METODOLOGÍAS PARA ACCESAR BASES DE DATOS A TRAVÉS DE SITIOS WEB

METODOLOGÍAS PARA ACCESAR BASES DE DATOS A TRAVÉS DE SITIOS WEB

JUSTO ARAGON MACHUCA

JACK ARAGON MACHUCA

CORPORACIÓN UNIVERSITARIA TECNOLÓGICA DE BOLÍVAR

FACULTAD DE INGENIERÍA DE SISTEMAS

CARTAGENA , D.T. Y C.

2001

METODOLOGÍAS PARA ACCESAR BASES DE DATOS A TRAVÉS DE SITIOS WEB

JUSTO ARAGÓN MACHUCA Código: 01-05-860

JACK ARAGÓN MACHUCA Código: 9305806

Trabajo de grado para optar al titulo de Ingeniero de Sistemas

Director

GIOVANNY VÁSQUEZ MENDOZA

Ingeniero de Sistemas

CORPORACIÓN UNIVERSITARIA TECNOLÓGICA DE BOLÍVAR

FACULTAD DE INGENIERÍA DE SISTEMAS

CARTAGENA, D.T. Y C.

2001

Articulo 105

La **Corporación Universitaria Tecnológica de Bolívar**, se reserva el derecho de propiedad intelectual de todos los trabajos de grado aprobados y no pueden ser explotados comercialmente sin su autorización.

Dedico esta tesis de grado a Dios, a mis hermanos, y de manera especial a mi madre; Por ser uno de los grandes motivos que han impulsado mi vida.

A mi hija Laura V. Aragón Mendoza, que dios te bendiga, y proteja toda tu vida.

A Enny I. Mendoza Pájaro, que este logro sea un motivo que alimente tus objetivos.

JUSTO ARAGÓN MACHUCA

Le doy gracias a Dios, a mis padres Rosa Maria Machuca y Justo Aragon Carpio, a mis hermanos y a todas aquellas personas que me apoyaron a lo largo de esta labor, colaborándome y dándome ánimos para seguir adelante.

JACK ARAGÓN MACHUCA

AGRADECIMIENTOS

Los autores expresan sus agradecimientos a:

- Giovanny Vásquez Mendoza. Ingeniero de Sistemas y Director del proyecto, por su valiosa colaboración, orientación y motivación.

- José Yi Romaní. Ingeniero de Sistemas, por la orientación oportuna prestada en el momento indicado.

- Gonzalo Garzón. Ingeniero de Sistemas, Decano de la Facultad de Ingeniería de Sistemas de la Corporación Universitaria Tecnológica de Bolívar, por la valiosa colaboración prestada.

- Y a todas las personas que de una u otra forma, hicieron su aporte para la realización de este trabajo.

Nota de aceptación

Presidente del Jurado

Jurado

Jurado

Cartagena, 16 de Octubre de 2001

CONTENIDO

pág.

pág.

pág.

pág.

pág.

pág.

LISTA DE TABLAS

pág.

LISTA DE CUADROS

LISTA DE FIGURAS

pág.

GLOSARIO

ACTIVEX: tecnología creada por la empresa *Microsoft* que brinda un entorno de programación para permitir la interacción y la personalización de los aplicaciones.

APPLET: pequeño programa hecho en lenguaje Java.

API: *Application Program Interface*. Interfaz para la programación de aplicaciones.

APLICACIÓN DE SERVIDOR: software implementado para responder a las solicitudes realizadas por los clientes y que se ejecuta como proceso de fondo.

CGI: conjunto de medios y formatos para permitir y unificar la comunicación entre el Web y otros sistemas externos, como las bases de datos.

CIBERESPACIO: es la denominación del espacio virtual (no físico) donde las personas se reúnen en Internet. También denomina a la cultura, usos y costumbres de la comunidad electrónica.

CLIENTE: es una aplicación o un proceso que solicita un servicio de otro proceso o componente.

CLIENT SIDE CGI SCR**P**T: programa interpretado CGI que se ejecuta en el cliente.

COMPONENTE: unidad discreta de código que ofrece u conjunto de servicios bien definidos a través de interfaces bien especificadas. Estos proporcionan los objetos que clientes requieren en tiempo de ejecución.

COM: *Component Object Model*. Tecnología desarrollada por Microsoft, cuyo objetivo consiste en dotar a los desarrolladores de un ámbito en y mediante el cual se puedan crear componentes que puedan comunicarse con otros componentes o aplicaciones, proporcionándoles servicios, incluso, aunque estos hallan sido creados en otros lenguajes por otros desarrolladores.

DLL: *Dinamic Link Library*, librería dinámica enlazada. Archivo que contiene datos y funciones que pueden ser usados por una aplicación de MS Windows.

GATEWAY: dispositivo de comunicación entre dos o más redes locales *(LAN)* y remotas, usualmente capaz de convertir distintos protocolos, actuando de traductor para permitir la comunicación. Como término genérico, es utilizado para denominar a todo instrumento capaz de convertir o transformar datos que circulan entre dos medios o tecnologías.

HIPERTEXTO: es una forma diferente de organizar información. En lugar de leer un texto en forma continua, ciertos términos están unidos a otros mediante relaciones (enlaces o links) que tienen entre ellos. Este esquema permite saltar de un punto a otro en un texto, y a través de los enlaces, permite que los navegantes busquen información de su interés en la Red, guiándose por un camino distinto de razonamiento.

HIPERDOCUMENTO: documento que tiene estructura de hipertexto; pero contiene además referencias a objetos multimediales.

HIPERENLACE: es un elemento en un documento electrónico que apunta a otro lugar en el mismo documento o a otro documento diferente.

HOST: sinónimo del servidor que contiene el contenido Web de uno o varios sitios.

HTML: *HyperText Markup Language*, Lenguaje de Marcado de Hipertextos. Lenguaje que define textos, subgrupo del SGML, destinado a simplificar la escritura de documentos estándar. Es la base estructural en la que están diseñadas las páginas de la World Wide Web

HTTP: *HyperText Transfer Protocol*, Protocolo de Transferencia de Hipertexto. Es el mecanismo de intercambio de información que constituye la base funcional de la *World Wide* Web.

IIS: *Internet Information Server*. Servidor Web y de aplicaciones de la plataforma *MS Windows NT.*

INTERFAZ: cara visible de los programas. Interactúa con los usuarios. La interfaz abarca las pantallas y su diseño, el lenguaje usado, los botones y los mensajes de error, entre otros aspectos de la comunicación computadora / usuario. Elemento de transición o comunicación entre datos que facilita el intercambio de datos entre aplicaciones generalmente incompatibles.

INTERFAZ NATIVA DEL GESTOR: API suministrado con el motor de base de datos y que sirven para comunicar las aplicaciones de servidor creadas por los desarrolladores y/o interfaces clientes del gestor y el motor de base de datos.

INTRANET: red corporativa que ofrece servicios Web, la cual es solo accesible solo para sus empleados.

ISAPI: *(Internet Server Application Programming Interface)* Interfaz de programación de aplicaciones del servidor de Internet. Es un conjunto de funciones para servidores de Internet, como las que ejecuta *Windows NT Server* con *Microsoft Internet Information Server* (IIS).

MAINFRAME: estructura principal. Computadora de gran tamaño de tipo multiusuario, utilizada en grandes corporaciones.

METODOLOGÍA: modo ordenado de proceder para llegar a un resultado o fin determinado.

MIDDLEWARE: módulo intermedio que actúa como conductor entre dos módulos de software. Para compartir datos, los dos módulos de software no necesitan saber cómo comunicarse entre ellos, sino cómo comunicarse con este.

MULTIMEDIA: combinación de varias tecnologías de presentación de información (imágenes, sonido, animación, video, texto) con la intención de captar tantos sentidos humanos como sea posible.

NAVEGADOR: aplicación cliente que permite leer documentos con formato en el Web, seguir enlaces de documento en documento de hipertexto y presentar imágenes en línea.

NCSA: *National Center for Supercomputer Applications*, Centro Nacional de Aplicaciones para Supercomputadoras, situado en Illinois, allí se desarrollo el primer Navegador.

PLATAFORMA CRUZADA: programa o dispositivo que puede utilizarse sin inconvenientes en distintas plataformas de hardware y sistemas operativos.

SCRIPT: programa que se interpreta y ejecuta en la memoria.

SERVIDOR: sistema que pone sus recursos (datos, impresoras, accesos) al servicio de otro a través de una red.

SERVER SIDE CGI SCRIPT: programa interpretado CGI que se ejecuta en el servidor.

TECNOLOGÍA CLIENTE DEL GESTOR: interfaz o pieza de software que accede a las funciones nativas proporcionadas por el gestor de base de datos.

UNICODE: código de 16 ligamentos para exponer signos en el computador, parecido al ASCII pero que posee un mayor numero de signos lo que permite el uso alfabético de todos los idiomas del mundo.

USUARIO: persona a la que se le permite el acceso a un computador o una red.

WAN: *Wide Area Network,* red de area amplia. Red de comunicaciones que conecta computadores dispersos en una amplia área geográfica.

WORLD WIDE WEB (WWW): conjunto de servidores que proveen información organizada en sitios, cada uno con cierta cantidad de páginas relacionadas. El Web es una forma de organizar toda la información existente en Internet a través de un mecanismo de acceso común de fácil uso, con la ayuda del hipertexto y la multimedia.

RESUMEN

El presente trabajo tiene como objetivo el de generar una propuesta metodológica representada en una vasta documentación, en la cual estén plasmados los resultados obtenidos de las diferentes evaluaciones, realizadas a las técnicas o metodologías, existentes en el entorno informático para la manipulación de datos residentes en bases de datos centralizadas, mediante tecnologías basadas en el Web.

Dicha documentación, estará dirigida a la comunidad de la Corporación Universitaria Tecnológica de Bolívar (CUTB), con el objeto de que sea una fuente de conocimientos para aportar los suficientes conceptos que ayuden a la formación de criterios bien definidos, para poder discernir acerca de los diferentes pasos y procesos, que se deben observar para llevar a cabo dichas labores de manipulación de datos.

Se exploraran todos los conceptos técnicos básicos y los propios de las diversas metodologías para acceder una fuente de datos, remotamente a través de Internet, además de cómo obtener los resultados esperados de acuerdo a los requerimientos de la implementación, haciendo uso de las herramientas de software apropiadas para este propósito.

Para realizar este trabajo se utilizo un estudio descriptivo de diferentes técnicas y herramientas que son utilizadas para tal fin en la actualidad.

Para conseguir los resultados se recurrio al software necesario, como lo son las siguientes plataformas computacionales:

- Windows NT 4.0
- Windows 98 SE
- Linux SuSE 7.0 Kernel 2.2

Utilizando los sistemas gestores de base datos:

- MySQL
- Microsoft SQL Server 7.0
- Sybase ASE 12.0
- PostgreSQL 7.0
- InterBase6
- Oracle 8.0
- Access 2000

INTRODUCCIÓN

Para nadie es un secreto de las posibilidades que ofrece tanto el Web como Internet, los grandes logros tecnológicos al nivel de las telecomunicaciones han logrado cambiar muchos aspectos o costumbre tradicionales en la vida cotidiana del hombre, por ejemplo en la actualidad el auge del correo electrónico ha escalado posiciones insospechadas, ha incursionado en el medio de las formas tradicionales de comunicación como el telegrama o el fax, esto lógicamente conlleva a la demanda de mas y mejores servicios. Como se puede observar las posibilidades que ofrece la red son casi impredecibles lo cual es consecuencia directa de causas como: la masificación del servicio y su bajo costo, las nuevas tecnologías aplicadas al comercio electrónico.

Por otra parte la red en conjunto, ha abierto la posibilidad de que mucha información relacionada con temas estudiados o investigaciones en vía de desarrollo, además de todo tipo de información no especializada sean publicadas en los servidores que la conforman. Esto implica que se destinen dispositivos especializados para dar un manejo optimo a la disposición final de dicha información como también sugiere que técnicas o metodologías se usaran para su manipulación.

Afortunadamente el hombre cuenta hoy en día con una de las herramientas en software más apreciable: Los Sistemas Manejadores de Bases de Datos (DBMS).

Hoy por hoy las organizaciones modernas tienen un concepto más fluido y contemporáneo de las bases de datos, es decir, saben que información debe incluirse en una base de datos, cual debe ser su estructura, como se deben almacenar los datos producidos por su desempeño, y como deben recuperarse y procesarse dichos datos para su aprovecho máximo.

Pero esto de las bases de datos no es algo nuevo, siempre han estado presentes en casi todas las actividades del hombre, desde sus inicios prehistóricos y a lo largo de toda su historia, las bases de datos modernas no son mas que la evolución de los métodos utilizados en otras épocas. A continuación se relacionan ciertas ventajas que brindan las bases de datos.

- La forma de recuperación de la información, utilizando bases de datos en la actualidad goza de un alto grado de eficiencia.
- Las capacidades de almacenamiento son bondadosas.
- Los medios de escritura para tal información son mas seguros, rápidos, y de precio asequible y esto se puede aplicar a casi cualquier componente de *hardware.*
- Los datos son altamente seleccionados, se registra lo esencial.
- La información de entrada goza de cierta depuración.

- La relación costo - beneficio (aplicados a las variables almacenamiento y recuperación), hace que las bases de datos sean una opción ideal y optima para el manejo de la información en los negocios modernos.

El diseño de una aplicación basado en bases de datos es un proceso altamente técnico, la cual debe garantizar: Eficiencia, Seguridad, Respaldo, Integridad, entre otros aspectos relevantes. La información almacenada en una base de datos puede ser utilizada como un arma estratégica de negocios para la empresa.

Por todos los aspectos antes mencionados y algunos no menos importantes que se escapan, se puede decir que las bases de datos, son una de las maravillas de la cultura informática.

¿Pero que relación guarda con la red de redes? Ó mejor ¿Por qué usar el Web para acceder información desde una base de datos?.

Bien, la relación Internet - bases de datos hoy día es casi imprescindible. Las empresas comerciales, universidades, centros de investigación entre otras entidades dependen de este par de elementos por obvias razones. El Web es una aplicación que ofrece ciertos estándares tales como:

- Soporte a lenguajes estándar de composición de documentos.
- Manejo de una Interfaz Grafica de Usuario *(GUI).*

- Independencia de la plataforma.
- Soporte de red.

Además el conjunto Internet, Web y las bases de datos ofrecen valores como:

- Estrategia comercial para relegar a sus competidores.
- Expansión a nuevos mercados (sin limite: local, regional, continental).
- Atención a los clientes 24 horas al día y 365 al año, en situaciones ideales.
- Posibilidades para realizar fusiones estratégicas con otras empresas.
- Explotación de mercados que no se exploraban.
- Buenas expectativas en el aspecto de comunicaciones.
- Posicionar la empresa a la vanguardia.
- Reafirmar la confianza que los clientes tienen en la empresa.

Teniendo en cuenta los antecedentes expuestos anteriormente, con este proyecto de grado se pretende conducir una investigación la cual tiene como objetivo la elaboración de una propuesta metodológica que facilite la implementación de proyectos de sistemas para acceder bases de datos centralizadas basadas en el modelo relacional, a través de tecnologías Web para tal efecto.

Para la elaboración de este trabajo se recopilaron y documentaron varios temas que se plasmaron en 15 capítulos.

El capitulo uno contiene las generalidades que se suceden y obvian en el intercambio de información a través del Web. Se expone la especificación OSI, el protocolo TCP/IP, los sistemas multiusuarios y el modelo Cliente–Servidor, Internet, el Web y sus componentes.

El segundo capitulo esta dedicado al tema de las bases de datos. En este se describen las bases de datos y sus componentes. Los sistemas de bases de datos, arquitecturas, características, ventajas y desventajas; el modelo relacional y el lenguaje SQL, Los gestores de bases de datos y por ultimo las bases de datos en el Web.

En el capitulo tres se describen las tecnologías de servidor que permiten el intercambio de información el Web, tales como: CGI características componentes ventajas y desventajas e ISAPI, comparación entre estas dos tendencias, y termina con los lenguajes que permiten crear aplicaciones Web.

A continuación, en el capitulo cuatro, se realiza una exhaustiva descripción de los componentes tecnológicos que permiten a las aplicaciones acceder a los gestores de bases de datos.

En el capitulo cinco se hace una exposición sobre la evolución y los factores que inciden en la implementación y uso de las bases de datos en la actualidad, y el porque utilizar la tecnología Web para accesar las bases de datos.

También se hace una breve introducción sobre las tecnologías que permiten acceder a los datos a través del Web, las metodologías evaluadas y el software utilizado en la evaluación metodológica de estas.

Los capítulos 6, 7, 8, 9, 10, 11 y 12 están dedicados a la evaluación de algunas de las técnicas que permiten crear las aplicaciones para acceder a los datos. Cada capitulo trata el marco teórico, filosofía, características y elementos de la tecnología en cuestión y termina con la evaluación de la técnica y los respectivos resultados de la evaluación.

Los capítulos 13, 14 y 15 contienen los resultados generales, las conclusiones y las recomendaciones respectivamente.

1. GENERALIDADES

Hoy día, Internet es uno de los términos más utilizados y mencionados, ya sea como medio de difusión de información, como tecnología, como tendencia futurista, como medio de diversión y entretenimiento, o como medio de comunicación. Mucha gente ha hecho de Internet una herramienta útil para diferentes propósitos ya sea laborales, educativos o personales.

Internet nació como una red de computadoras de la defensa de los Estados Unidos la cual se le llamó *ARPANET*. Posteriormente se unieron a dicha red, algunas importantes universidades de los Estados Unidos. La idea era estar enlazados para compartir información y experiencias. En esos inicios la interfaz de la comunicación entre los miembros de Internet era muy rudimentaria por lo que era necesario utilizar herramientas y métodos muy técnicos para realizar la conexión entre las computadoras. Es por eso que la utilización de Internet estuvo reducida a algunos pocos investigadores, especialistas y estudiantes universitarios del área de informática. La conexión de las computadoras se realizaba a través del protocolo de comunicación TCP/IP *(Transfer Control Protocol / Internet Protocol)*, el cual era el mecanismo mediante el cual las computadoras podían estar conectadas y comunicándose entre sí. Este protocolo de comunicación resultó ser a la postre un elemento clave para el crecimiento posterior de Internet.

Otro elemento importante en el desarrollo de Internet fue la estandarización de la información a través del lenguaje HTML, este lenguaje hizo posible la estandarización de la información mediante documentos sencillos, breves y que podían ser perfectamente presentados en pantallas visualmente entendibles para la gente no especialista en informática. Ello fue mediante el concepto del *Browser* o Navegador Web, el cual es una herramienta que formatea el código HTML en pantallas sencillas con información de muy diversos tipos (texto, imágenes, tablas; etc.).

Utilizando todos esos conceptos, se volvió altamente deseable que la información que ya muchas redes tenían fuera compartida y difundida mediante la nueva forma de presentar la información. Una de las causas que ha generado un crecimiento aún más espectacular de Internet, de lo que originalmente se pensó que podía tener, es la capacidad de las páginas y sitios Web de acceder datos, ya sea para consultarlos o para modificarlos.

El acceso a bases de datos, ha causado que los sitios Web tengan la capacidad de comportarse como los sistemas de información tradicionales; pero con la enorme ventaja de tener una interfaz común y estándar: el navegador. Actualmente existen gran cantidad de utilidades y programas que ayudan a la tecnología de las páginas Web a acceder información contenida en bases de datos tradicionalmente sólo accedidas vía lenguajes de programación.

1.1 MODELO DE INTERCONEXIÓN DE SISTEMAS ABIERTOS

En la actualidad, se utilizan muchos tipos distintos de computadoras. Básicamente se diferencian entre sí por sus sistemas operativos, CPU, interfaces de red y muchas otras variables. Al tratar de comunicar todas estas arquitecturas, las diferencias plantean un importante problema. En respuesta a este inconveniente la Organización Internacional para la Normalización (ISO), crea un subcomité con objeto de proporcionar un estándar de comunicación entre diversos fabricantes. El resultado fue la creación del modelo de referencia para el diseño de protocolos de Interconexión de Sistemas Abiertos *(OSI, Open Systems Interconection).*

El modelo propuesto OSI no especifica ningún estándar o protocolo de comunicaciones si no que, en su lugar dicta una serie de normativas a seguir en la transmisión de datos. Para llegar a este resultado el subcomité ISO adopto el enfoque de divide y vencerás. Si se divide el complejo proceso de transmisión de datos en tareas más pequeñas, el problema se hace más manejable y cada tarea puede optimizarse individualmente. El modelo se ha convertido en un estándar internacional. Esta dividido en siete capas o niveles, cada uno de las capas del modelo define una sección específica del total de la arquitectura.

Diferentes organismos de estandarización (ISO, IEEE, ANSI; etc.) han definido diversos protocolos sobre esos niveles para adaptar las implementaciones finales a variados entornos y requisitos. (Véase la Tabla 1). A cada capa se le asigna un

conjunto determinado de funciones. Cada una de estas utiliza los servicios de la capa inferior y proporciona servicios a la capa inmediatamente superior, ocultando los detalles de cómo se han implementado los servicios realmente.

Tabla 1. Servicios prestados en cada capa OSI

Capa	Servicio
Física	Suministra la conexión física entre un sistema informático y la red.
Enlace de datos	Esta capa empaqueta y desempaqueta los datos para su transmisión. Compone la información en tramas y ofrece un control de errores al nivel de red.
Red	Esta proporciona el enrutamiento de los datos por la red.
Transporte	Esta proporciona el guión y aceptación de la transmisión.
Sesión	Esta capa establece y finaliza los enlaces de comunicación.
Presentación	Efectúa la conversión de datos y se asegura que los datos se intercambien en un formato universal.
Aplicación	Proporciona una interfaz a la aplicación que ejecuta un usuario, es decir una pasarela entre las aplicaciones de usuario y el proceso de comunicación en red.

Cada capa cree que se comunica con la misma capa del otro sistema, cuando en realidad cada capa se comunica única y exclusivamente con las capas adyacentes de su propio sistema, es decir la capa tres en un sistema A, se comunica con la capa tres de otro sistema B pero en realidad se están comunicando con las capas cuatro y dos de sus respectivos sistemas. (Véase la Figura 1).

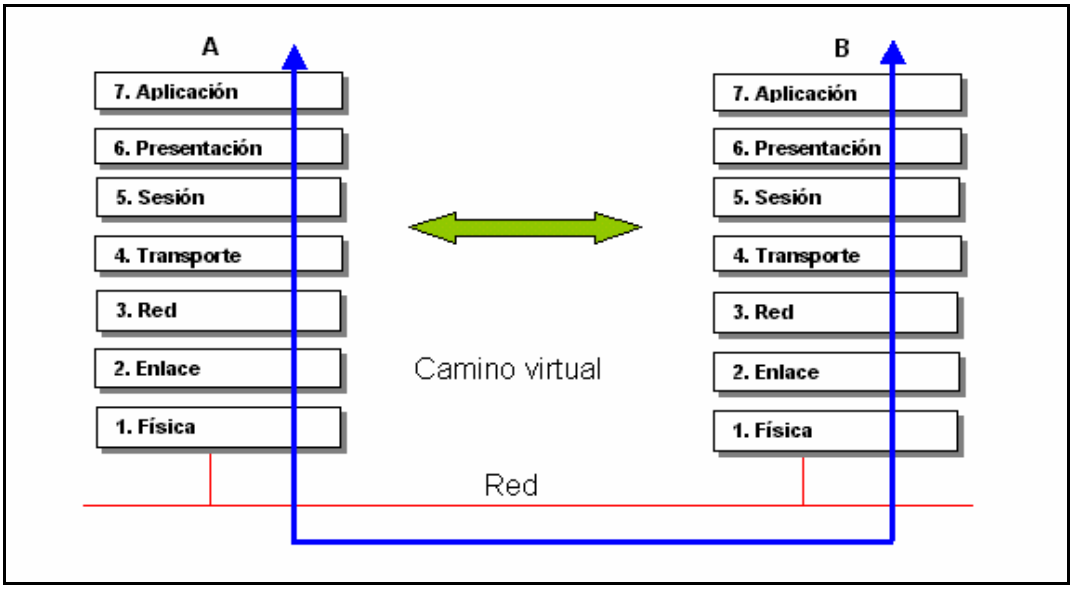

Figura 1. Comunicación entre capas OSI

Al pasar la información desde una capa superior a otra inferior, se añade a un encabezado a los datos para indicar de donde procede la información y adonde se dirige. Las unidades de información reciben nombres distintos en cada capa. (Véase la Tabla 2).

1.2 EL PROTOCOLO TCP/IP

El conjunto de protocolos TCP/IP representa una arquitectura de red similar al modelo de red OSI de ISO. No es un protocolo, sino en realidad una familia de protocolos reunidos bajo las siglas TCP/IP. La Figura 2, muestra la relación de las capas TCP/IP en el conjunto de protocolos ISO.

Tabla 2. Términos utilizados por las capas OSI para hacer referencia a unidades de información

Capa OSI	Nombre de la unidad de información
Aplicación	Mensaje.
Transporte	Segmento.
Red	Data grama.
Enlace de datos	Trama (paquete).
Física	Bit.

TCP/IP no establece tantas distinciones como OSI entre las capas superiores del conjunto de protocolos. Las tres capas superiores OSI equivalen a los protocolos de proceso de Internet, por ejemplo: La capa de transporte del modelo OSI es el responsable de la entrega fable de los datos. En el conjunto de protocolos de Internet, esta tarea corresponde a los protocolos TCP y UDP.

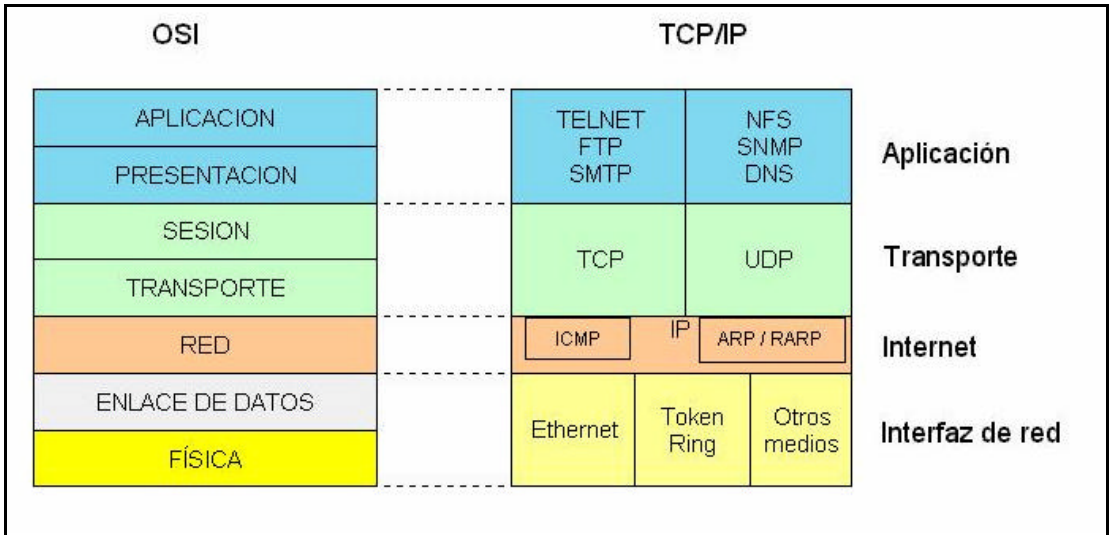

Figura 2. Comparación entre las capas OSI y TCP/IP

1.2.1 TCP. *TCP (Transmisión Control Protocol).* El protocolo de control de transmisión es un servicio orientado a la conexión, Se utiliza para traducir mensajes de longitud variable procedentes de los protocolos de capa superior, y supervisa la recepción y conexión del flujo de datos fiables entre sistemas remotos.

1.2.2 UDP. UDP *(User Datagram Protocol).* El protocolo del datagrama de usuario, es similar a TCP, salvo que no esta orientado a conexiones (no demanda conexión directa entre el remitente y el destinatario), y no supervisa la recepción de los datos. UDP solo recibe mensajes y los transmite a los protocolos de nivel superior.

1.2.3 IP. El Protocolo Internet *(Internet Protocol),* se ocupa de las comunicaciones sin conexiones entre sistemas. Dentro del modelo OSI forma parte de la capa de Red. Esta capa es la que gestiona el movimiento de la información en la red. La comunicación se lleva a cabo examinando la dirección en el mensaje de la capa de Red, que determina los sistemas y la ruta que debe utilizarse para enviar el mensaje; pero no garantiza la entrega de dicho mensaje.

1.2.4 Direcciones IP. Para comunicar dos o más computadoras en una red utilizando el protocolo TCP/IP se hace necesario asignarles a cada una dirección que las identifique dentro de dicha red. Esta dirección se conoce como *dirección IP*.

Esta consta de una serie de cuatro octetos, estos definen una dirección única, en la que una parte representa una red y opcionalmente una subred, y la otra parte representa un nodo especifico de la red.

1.3 SISTEMAS MULTIUSUARIOS

Un sistema multiusuario utiliza dos conceptos importantes: servicios de multitarea y multiusuario.

Los servicios multitarea se refieren a la capacidad que tenga la plataforma de ejecutar varios procesos al mismo tiempo y de forma transparente al usuario. Por ejemplo puede navegarse en la Internet a través de un *browser* o navegador, escuchar archivos de sonido, imprimir un archivo mientras se compila un programa, todo a la vez. A diferencia de los sistemas operativos de un solo usuario, caso *DOS*, en donde lo utiliza una sola persona a la vez, todos los recursos les son asignados a la aplicación que en ese momento el usuario este ejecutando, los sistemas multiusuarios permiten el acceso a los recursos de la maquina a varios usuarios al tiempo.

Para que esto se dé, los usuarios deben estar conectados a la maquina, esto puede hacerse por medio de una terminal o una computadora ubicados físicamente cerca del servidor y comunicados por medio de un cable o bien estar al otro lado del planeta conectado por medio de líneas de datos de alta velocidad o de la línea telefónica. El uso de un terminal o un computador personal y la forma

de estar conectados al servidor determinan la distribución o centralización de los recursos. Los sistemas multiusuarios hacen uso de los modelos de *arquitectura distribuida* y *centralizada* para dar servicio a muchos usuarios al mismo tiempo.

1.3.1 La arquitectura centralizada. Nace en torno a una concepción tradicional de la organización, con estructura centralizada y jerárquica, dividida en departamentos. Cada departamento tiene actividades muy concretas. Las relaciones que pueda establecer con otros departamentos están muy definidas y limitadas y suelen realizarse a través de la jerarquía. En un entorno de proceso centralizado, muchos usuarios acceden a los recursos de una computadora, es decir al almacenamiento, impresión, memoria y procesamiento de la misma.

1.3.1.1 Propiedades.

- El computador central es el único computador de la organización.
- El computador contiene todos los datos y es el responsable de la consolidación de la información.
- Desde el computador central se controla el acceso a múltiples terminales conectados a través de productos integrados en la arquitectura de red del suministrador.
- Los terminales funcionan como "esclavos" del computador central.
- Cada usuario tiene un número asignado, y derechos y prioridades de ejecución en la máquina, de sus programas o peticiones.

1.3.1.2 Características físicas.

- Un computador corporativo dimensionado para soportar todos los procesos de la organización, todos los datos y las posibles comunicaciones con las delegaciones.
- Una gran base de datos donde residen todos los datos del organismo.
- Impresoras y terminales (u computadores personales con emulación de terminal) como puestos de trabajo conectados en grupos *(clusters)*, al computador central.

1.3.1.3 Características lógicas.

- Ejecución de todos los procesos en el computador corporativo.
- Si la empresa está dispersa geográficamente y dispone de comunicaciones, todos los puestos de trabajo están conectados al computador formando una "estrella".

1.3.1.4 Ventajas.

- Alto rendimiento transaccional.
- Alta disponibilidad.
- Entorno probado y personal experimentado.

- Control total del computador, al ser éste único y residente en un único Centro de Proceso de Datos.
- Concentración de todo el personal de explotación y administración del sistema en un único Centro de Proceso de Datos.

1.3.1.5 Inconvenientes.

- Alto precio del computador, al requerirse mucha potencia de tratamiento para dar servicio a todos los usuarios que estén conectados y gran espacio en disco para albergar todos los datos del organismo.
- Arquitecturas propietarias.
- Alta dependencia de las comunicaciones, si existen. En caso de caída de una línea, todos los puestos de trabajo dependientes de dicha línea quedan inoperantes.
- Interfaces de usuario de caracteres (no gráficos) y, por lo tanto, poco amigables.

1.3.2 La arquitectura distribuida. Surge con los nuevos modelos organizativos, en los que la empresa se divide en unidades más o menos autónomas que establecen relaciones más definidas y directas entre sí. En el entorno de procesamiento distribuido el proceso puede realizarse en la estación de trabajo del usuario y utilizar el procesador central para distribuir aplicaciones y datos o pueden estar en el servidor central y ejecutarse desde allí. Las impresoras y los

sistemas de almacenamiento pueden estar conectados a la estación de trabajo del usuario o al servidor principal.

1.3.2.1 Propiedades.

- Nombre global: el mismo nombre es válido en todo el sistema.
- Acceso global: los mismos métodos actúan en objetos, en cualquier parte del sistema.
- Seguridad global: autenticación y acceso uniformes en todo el sistema.
- Disponibilidad global: funcionamiento correcto en presencia de fallos parciales.
- Cada usuario trabaja con su terminal local inteligente, con lo que obtiene mejores tiempos de respuesta.
- Los recursos necesarios que no estén disponibles sobre el terminal local (computador personal o estación de trabajo), pueden tomarse del computador central a través de la red de telecomunicaciones.
- Transparencia del sistema: los usuarios no conocen la ubicación de un componente en el sistema distribuido o nada de su fabricante, modelo, sistema operativo local, velocidad o tamaño. Todos los servicios se piden por su nombre.

1.3.2.2 Características físicas.

- Sistemas informáticos distribuidos en los que los ordenadores, a través de la

organización, están conectados por medio de una red de telecomunicaciones.

- Cada computador sobre la red tiene capacidad de tratamiento autónomo que permite servir a las necesidades de los usuarios locales.

1.3.2.3 Características lógicas.

- Cada tarea individual puede ser analizada para determinar si puede distribuirse o no. En general, las tareas más complejas o de carácter estratégico para la organización se mantienen en el computador central. Las tareas de complejidad media o específicas para un determinado grupo de usuarios, se distribuyen entre las máquinas locales de ese grupo.
- La plataforma física seleccionada puede ajustarse a las necesidades del grupo de usuarios, con lo que surgen los ordenadores especializados para determinados tipos de tareas.

1.3.2.4 Ventajas.

- Funcionamiento autónomo de los sistemas locales, lo que origina un buen tiempo de respuesta.
- Abre posibilidades de trabajo mucho más flexibles y potentes.

1.3.2.5 Inconvenientes.

- Requiere un intenso flujo de informaciones (muchas veces no útiles, como pantallas y datos incorrectos) dentro de la red, lo que puede elevar los costos de comunicaciones.
- Si los sistemas no están integrados, pueden producirse problemas de inconsistencia de datos.

1.4 EL MODELO CLIENTE - SERVIDOR

La arquitectura Cliente - Servidor es un modelo para el desarrollo de sistemas de información, en el que las transacciones se dividen en procesos independientes que cooperan entre sí para intercambiar información, servicios o recursos. Se denomina cliente al proceso que inicia el diálogo o solicita los recursos y servidor, al proceso que responde a las solicitudes. Es el modelo de interacción más común entre aplicaciones en una red. No forma parte de los conceptos de la Internet como los protocolos IP, TCP o UDP, sin embargo todos los servicios estándares de alto nivel propuestos en Internet funcionan según este modelo.

1.4.1 Componentes del modelo. Los principales componentes del modelo son los *Clientes*, los *Servidores* y la *Infraestructura de Comunicaciones*. En este modelo, las aplicaciones se dividen de tal forma que el servidor contiene la parte que debe ser compartida por varios usuarios, y en el cliente permanece sólo lo particular de cada usuario.

1.4.1.1 El Cliente. Los clientes interactúan con el usuario, usualmente en forma gráfica. Frecuentemente se comunican con procesos auxiliares que se encargan de establecer conexión con el servidor, enviar el pedido, recibir la respuesta, manejar las fallas y realizar actividades de sincronización y de seguridad.

Las funciones del cliente son:

- Manejo de la interfase del usuario.
- Captura y validación de los datos de entrada.
- Generación de consultas e informes sobre las bases de datos.

En el caso de las aplicaciones Web, el cliente debe cumplir unas especificaciones para que sea considerado como tal, y son las siguientes:

- El cliente deberá ser *ligero* para poder llegar a cualquier usuario potencial.
- *Estándar* ya que en el modelo Web, no tiene sentido plantear la distribución de un *cliente propietario.*
- Debe facilitar el acceso a virtualmente cualquier tipo de aplicación.

Si bien existen varias alternativas, el navegador Web es la solución más adecuada hasta la fecha, constituyendo el ***cliente universal.***

1.4.1.2 El Servidor. El servidor ofrece un servicio al cliente y devuelve los resultados. En algunos casos existen procesos auxiliares que se encargan de recibir las solicitudes del cliente, verificar la protección, activar un proceso servidor para satisfacer el pedido, recibir su respuesta y enviarla al cliente. Por las razones anteriores, la plataforma computacional asociada con los servidores es más poderosa que la de los clientes. Los servidores realizan, entre otras, las siguientes funciones:

- Gestión de periféricos compartidos.
- Control de accesos concurrentes a bases de datos compartidas.
- Enlaces de comunicaciones con otras redes de área local o extensa.
- Siempre que un cliente requiere un servicio lo solicita al servidor correspondiente y éste, le responde proporcionándolo. Normalmente; pero no necesariamente, el cliente y el servidor están ubicados en sistemas diferentes.

1.4.1.3 La infraestructura de comunicaciones. Para que los clientes y los servidores puedan comunicarse se requiere una infraestructura de comunicaciones, la cual debe proporcionar los mecanismos de direccionamiento, transporte y fiabilidad.

1.4.2 Características del modelo Cliente – Servidor.

- El servidor presenta a todos sus clientes una interfaz única y bien definida.
- El cliente no necesita conocer la lógica del servidor, sólo su interfaz externa.
- El cliente no depende de la ubicación física del servidor, ni del tipo de equipo físico en el que se encuentra, ni de su sistema operativo.
- Los cambios en el servidor implican pocos o ningún cambio en el cliente.

1.4.3 Arquitectura del modelo Cliente - Servidor. La gran parte de los programas de aplicación en la actualidad tienen tres capas, que son:

- ***Capa de presentación***
- ***Lógica de aplicación***
- ***Servicios***

La capa de presentación es la interfase que utiliza el usuario para poder comunicarse con la máquina. Para lograr esto la capa de presentación maneja los dispositivos de entrada como son el ratón y teclado, y los dispositivos de salida .

La lógica de aplicación contiene todos los procedimientos, reglas y tareas que realiza basado en los datos de entrada obtenidos por la primera capa, facilitando que estos datos puedan almacenarse, asegurando el cumplimiento de ciertas premisas.

La capa de servicios proporciona la información que necesitan las otras capas para poder hacer sus funcionales, entre esto están los servicios de archivos, de impresión, de comunicación y él más importante el servicio de base de datos.

Con base en esto, esta arquitectura se puede clasificar por lo estrecho o separado en que se encuentra la interactividad de las capas de los programas y según las funciones que asumen el cliente y el servidor. (Véase la Figura 3).

1.4.3.1 Primer Nivel.

Capa de presentación	± Servidor
Lógica de aplicación	Servidor
Servicios	Servidor

En el primer nivel el cliente asume parte de las funciones de presentación de la aplicación, ya que siguen existiendo programas en el servidor, dedicados a esta tarea. Dicha distribución se realiza mediante el uso de productos para el "maquillaje" de las pantallas del *mainframe*. Esta técnica no exige el cambio en las aplicaciones orientadas a terminales; pero dificulta su mantenimiento. Además, el servidor ejecuta todos los procesos y almacena la totalidad de los datos. En este caso se dice que hay una *presentación distribuida* o “embellecimiento”.

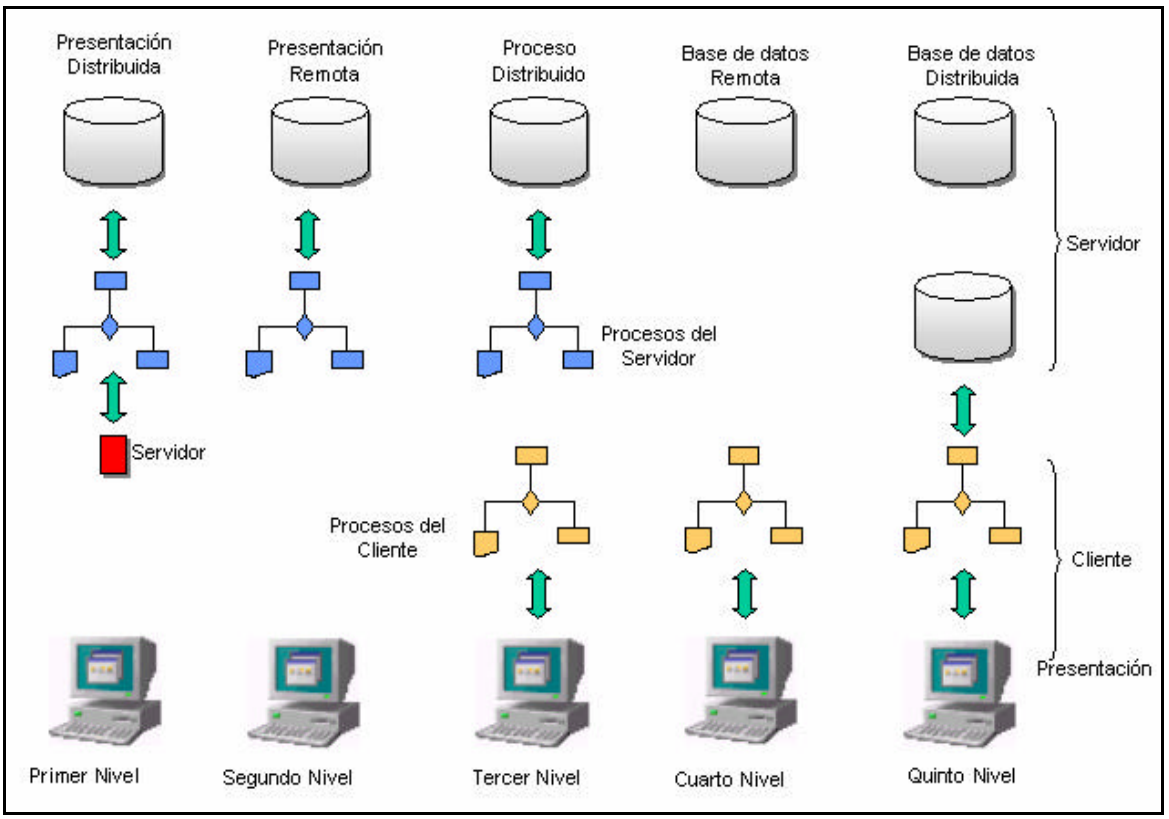

Figura 3. Niveles de la arquitectura Cliente – Servidor.

1.4.3.2 Segundo nivel.

Capa de presentación	Cliente
Lógica de aplicación	Servidor
Servicios	Servidor

En el segundo nivel, la aplicación está soportada directamente por el servidor, excepto la presentación que es totalmente remota y reside en el cliente. Los terminales del cliente soportan la captura de datos, incluyendo una validación

parcial de los mismos y una presentación de las consultas. En este caso se dice que hay una *presentación remota.*

1.4.3.3 Tercer nivel.

Capa de presentación	Cliente
Lógica de aplicación	Servidor / Cliente
Servicios	Servidor

En el tercer nivel, la lógica de los procesos se divide entre los distintos componentes del cliente y del servidor. Ahora las tres capas se encuentran separadas. La diferencia esta en que la capa de lógica aplicación se comporta como un servicio y por lo tanto se ejecuta en un servidor aunque también se puede ejecutar en el mismo cliente. Este servicio nuevo, bajo un servidor se llama *servidor de aplicaciones*.

El diseñador de la aplicación debe definir los servicios y las interfaces del sistema de información, de forma que los papeles de cliente y servidor sean intercambiables, excepto en el control de los datos, que es responsabilidad exclusiva del servidor. Cuando la capa de lógica de aplicación se encuentra mas presente en el cliente se dice que el cliente es pesado, de modo contrario se denomina que el cliente es ligero (como en el caso del navegador). En este tipo de situaciones se dice que hay un *proceso distribuido o cooperativo.*

1.4.3.4 Cuarto nivel.

Capa de presentación	Cliente
Lógica de aplicación	Cliente
Servicios	Servidor

En el cuarto nivel el cliente realiza tanto las funciones de presentación como los procesos. Por su parte, el servidor almacena y gestiona los datos que permanecen en una base de datos centralizada. En esta situación se dice que hay una *gestión de datos remota.*

1.4.3.5 Quinto nivel.

Capa de presentación	Cliente
Lógica de aplicación	Cliente
Servicios	Servidor

En el quinto y último nivel, el reparto de tareas es como en el cuarto nivel y además el gestor de base de datos divide sus componentes entre el cliente y el servidor. Las interfaces entre ambos, están dentro de las funciones del gestor de datos y por lo tanto, no tienen impacto en el desarrollo de las aplicaciones. En este nivel se da lo que se conoce como *bases de datos distribuidas.*

Existe también desarrollo de aplicaciones en *n-niveles*, que es cuando el servidor de aplicaciones requiere servicios de otros servidores para completar las

peticiones del cliente. Para este caso existe él termino *Middleware* que se aplica a todo lo existente entre la capa de presentación y la capa de servicios.

1.4.4 Ventajas de la arquitectura Cliente – Servidor. La arquitectura Cliente-Servidor ha acabado de imponiéndose gracias a que posee numerosas ventajas:

- Economía: es más económica que el *mainframe computing*, ya que el actual estado del mercado ha provocado el descenso del precio de los PC.
- Velocidad: la separación de los procesos reduce los cuellos de botella.
- Adaptabilidad: Esta arquitectura es abierta, por lo tanto pueden utilizarse diversos componentes de diversos proveedores.
- Facilidad de acceso a los datos: se han desarrollado en esta arquitectura herramientas visuales y sencillas para acceder a los datos del servidor.

1.5 INTERNET

Internet es el conjunto que contiene la mayor cantidad de personas, computadores, software y datos, funcionando de manera cooperativa, recopilando y publicando información a escala mundial. Internet esta basada en el modelo Cliente - Servidor y es la aplicación que mejor explota este modelo. En este gran conjunto de información y comunicación, se ofrecen una gran cantidad de servicios, que son ofrecidos en el ámbito global. Algunos de estos servicios son:

- *FTP:* servicio de transmisión de ficheros.
- *IRC:* canales de charla en vivo.
- *Listas de correo:* servicios de mensajería entre grupos de personas, mantenidas por correo electrónico.
- *Buscadores:* sistemas que organizan la información en Internet.

Dentro de esto servicios se destacan el *WWW (World Wide Web)*, tal vez la punta de lanza de Internet.

1.5.1 WWW (World Wide Web). El *World Wide Web* es un sistema multimedia de información distribuida, independiente de la plataforma y enmarcado en Internet, usado por millones de personas para compartir informaciones mucho mas complejas que las estrictamente textuales, concretamente imágenes, sonido, video y datos.

1.5.1.1 El WWW es un sistema de Información multimedia. El WWW es un sistema de información estructurada según el modelo hipertextual; pero no limitado al contenido textual, ya que a los contenidos a los que se pueden acceder incluyen gráficos, sonidos, video; pero la manera de acceder a estos es a través de vínculos hipertextuales.

1.5.1.2 El WWW es independiente de la plataforma. Puede accederse a los contenidos de WWW desde diferentes tipos de maquinas y en el ámbito de

sistemas operativos diversos. El WWW supone una revolución en la manera en que se acceden a los datos. A diferencia de otras aplicaciones, donde se usa una para cada tipo de documento, este sistema se accede a los datos a través de un único medio: *el navegador*. El navegador es una aplicación cliente que se comunica con el servidor Web mediante el protocolo conocido como *HTTP (HyperText Transfer Protocol).*

1.5.1.3 El WWW contiene información distribuida. La información accesible en el WWW se halla distribuida en numerosos computadores de todo el mundo. Cada pagina de información se encuentra identificada por una dirección o *URL (Uniform Resource Locator)* y ubicada en un lugar físico que es conocido como una sede o sito Web.

1.5.2 Uniform Resource Locator (URL). Un URL o localizador uniforme de recursos es un identificador único que identifica un documento o ubicación de almacenamiento de información en el WWW. Este documento puede ser, por ejemplo una pagina Web, un archivo FTP, una dirección de correo electrónico, un programa ejecutable o la respuesta de una transacción comercial.

1.5.3 HTML. *HTML* (*HyperText Markup Language)* Lenguaje de Marcado de Hipertextos, define una serie de normas que permiten especificar las características y funcionalidades de los documentos manejables por el WWW. El objetivo primordial de este lenguaje es crear una forma universal de almacenar y desplegar la información. Este permite agregar atributos a los documentos de

texto simple, crear documentos de hipertexto para publicar en el Web, así como implementar hipermedios. Un documento HTML es un conjunto de párrafos caracterizados por una serie de rasgos que determinan su estilo.

HTML no es la única manera, mediante la cual se pueda 'imprimir' los datos en el cliente, existen otras revolucionarias tecnologías que permiten hacerlo, tal es el caso de *XML (Extensible Markup Language)* Lenguaje de Marcación Extensible y el *WML (Wireless Markup Language),* lenguaje de marcación para comunicaciones inalámbricas, este ultimo usado principalmente en teléfonos celulares, *notepads* y dispositivos similares.

1.5.4 Extensible Markup Language (XML). XML Es una sintaxis universal para la descripción y estructuración de los datos independiente de la lógica de aplicación. Nació para simplificar y aplicar las características del *SGML (Standard Generalized Markup Language)* al Web. Lo que se propone con XML es crear un lenguaje (mas bien un metalenguaje), muy general que sirva para muchas cosas. HTML está diseñado para presentar información directamente a los humanos, pero es un lenguaje complicado de procesar para los programas informáticos, HTML no es bueno porque no indica lo que está representando, se preocupa principalmente de que tal texto tiene que ir en azul, o con un tipo de letra determinada; pero no dice que lo que está mostrando es el título de un libro o el precio de un artículo, XML hace precisamente esto: describe el contenido de lo que etiqueta.

XML promete simplificar y bajar los costos de la publicación e intercambio de datos en el Web. Posee una sintaxis basada en texto que puede ser legible entre los humanos y los computadores. XML ofrece portabilidad y reusabilidad del código entre diferentes plataformas y dispositivos. Es también flexible y extensible permitiendo que nuevas etiquetas sean añadidas a un documento sin que se rompa su estructura. Esta basado en *Unicode*, esto le permite ser soportado globalmente por todos los idiomas.

1.5.5 Hypertext Transfer Protocol (HTTP). El protocolo HTTP es el utilizado para transferir los contenidos Web, es decir, los documentos HTML a través de una conexión TCP/IP. La comunicación entre el servidor Web y los clientes se lleva acabo mediante peticiones y respuestas HTTP en el marco de una sesión HTTP. Una sesión HTTP se establece sobre una conexión TCP/IP entre un servidor y un cliente situado en una ubicación de *Internet* o de una *Intranet*.

2. EL ALMACENAMIENTO DE LA INFORMACIÓN

Pocas herramientas de software son tan necesarias como las bases de datos. No hay empresa o institución que pueda prescindir de ellas; son el alma de los programas contables, de la nómina y de los inventarios en las empresas tradicionales; pero de igual manera alojan la información que se consume en todo el mundo gracias a Internet.

2.1 BASE DE DATOS

Una base de datos es una colección de información, estructurada según un cierto modelo. Los contenidos de una base de datos, se organizan basándose en un modelo de datos. Este modelo de datos, o esquema, define las unidades mínimas de información que podrán almacenarse y las relaciones que entre ellas existen.

El modelo se explica al crear la base de datos, abstrayendo las características de cada entidad de datos y las relaciones y dependencias entre las mismas. Los datos se almacenan de modo que resulten independientes de los programas que los usan; y se emplean métodos bien determinados para incluir nuevos datos y para modificar o extraer los datos almacenados.

2.1.1 Objetos básicos. Las bases de datos, independientemente de cuales sean los mecanismos para almacenar y obtener su información, están compuestas fundamentalmente por tablas. Los siguientes son los elementos que definen la estructura de una base de datos.

2.1.1.1 Tabla. Una tabla es una colección de elementos de información agrupados en filas y columnas. (Véase la Figura 4). Una base de datos es por lo tanto una colección de tablas relacionadas.

2.1.1.2 Fila. Cada fila, registro o tupla, representa una instancia del objeto o entidad que abstrae la tabla. Una tabla resulta ser entonces una colección de filas.

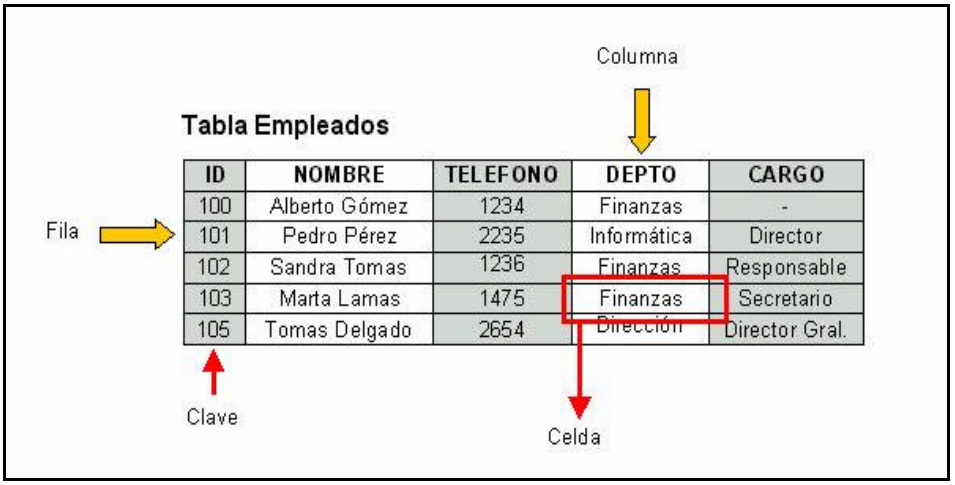

Figura 4. Estructura de una tabla

2.1.1.3 Columna. Las columnas de la tabla, o campos, representan los atributos o características de la entidad que representa los registros de la tabla. Una fila o registro resulta ser una colección de columnas o atributos. Cada columna almacena información homogénea.

2.1.1.4 Celda. Una celda es la intersección de una fila y una columna, y representa el valor de un determinado atributo para un registro especifico de la tabla.

2.1.1.5 Clave. Es un atributo o conjunto de atributos que identifican un registro de manera única.

2.2 SISTEMA DE BASES DE DATOS

Un sistema de bases de datos es una serie de recursos para manejar grandes volúmenes de información. Sin embargo no todos los sistemas que manejan información son bases de datos. Un sistema de bases de datos debe responder a las siguientes características:

- Independencia de los datos: es decir, que los datos no dependan del programa y por tanto cualquier aplicación puede hacer uso de los datos.
- Control de la redundancia: se llama redundancia a la existencia de duplicación de los datos; al reducir ésta al máximo se consigue un mayor aprovechamiento del espacio y además se evita que existan inconsistencias entre los datos.

Las inconsistencias se dan cuando se encuentran datos contradictorios.

- Seguridad: un sistema de base de datos debe permitir que se tenga un control sobre la seguridad de los datos.

Debido al gran desarrollo de las redes y al auge corporativo de llevar la información a los usuarios al lugar donde estén, hoy día los sistemas de bases de datos siguen dos arquitecturas o modelos de implantación: las bases de datos centralizadas y las bases de datos distribuidas.

2.2.1 Bases de Datos Centralizadas (BDC). Una base de datos centralizada es una base de datos almacenada en su totalidad en un solo lugar físico, o nodo de una red. Los componentes de la bases de datos centralizadas son: los datos, el Sistema Gestor de Base de Datos (SGBD) y los dispositivos de almacenamiento secundarios asociados.

2.2.1.1 Características. Entre las características más resaltantes de las bases de datos centralizadas se encuentran las siguientes:

- Se almacena completamente en una localidad central, es decir, todos los componentes del sistema residen en un solo computador o sitio.
- No posee múltiples elementos de procesamiento ni mecanismos de intercomunicación como las *bases de datos distribuidas (BDD).*
- El problema de seguridad es inherentemente fácil en estos sistemas.

2.2.1.2 Ventajas. Entre las ventajas de utilizar bases de datos centralizadas se pueden mencionar las siguientes:

- Se evita la redundancia: en sistemas que no usan bases de datos centralizadas, cada aplicación tiene sus propios archivos privados o se encuentran en diferentes localidades.
- Se evita la inconsistencia: ya que si un hecho específico se representa por una sola entrada (es decir, si la redundancia se elimina), la no-concordancia de datos no puede ocurrir.
- Pueden hacerse cumplir las normas establecidas: con un control central de la base de datos, el administrador puede garantizar que se cumplan todas las formas aplicables a la representación de los datos.
- Restricciones de seguridad: al tener jurisdicción completa sobre los datos de operación, el administrador del sistema puede:

    - Asegurar que el único medio de acceder la base de datos sea a través de los canales establecidos.
    - Definir controles de autorización para que se apliquen cada vez que se intente el acceso a datos sensibles.

- Conservación la integridad: el objetivo de la integridad es garantizar que los datos de la base de datos sean exactos. El control centralizado de la base de datos, ayuda a evitar la inconsistencia de los datos.

Es conveniente señalar que la integridad de los datos es un aspecto muy importante en una base de datos, porque los datos almacenados se comparten y porque sin procedimientos de validación adecuados es posible que un programa con errores genere datos incorrectos que afecten a otros programas que utilicen esa información.

- Equilibrar requerimientos contradictorios: cuando se conocen los requerimientos globales de la empresa, en contraste con los requerimientos de cualquier usuario individual, el administrador puede estructurar el sistema para brindar un servicio que sea el mejor para la empresa en términos globales.

2.2.1.3 Desventajas. Algunas de las desventajas que presentan los sistemas de bases de datos centralizadas son:

- Los *Mainframe* no ofrecen mejor proporción precio-rendimiento que los microprocesadores de los sistemas distribuidos.
- Por lo general, cuando un sistema de base de datos centralizada falla, se pierde toda la disponibilidad de procesamiento y sobre todo de la información confiada al sistema.
- En caso de un desastre o catástrofe, la recuperación es difícil de sincronizar.
- Las cargas de trabajo no se pueden difundir entre diferentes computadoras, ya que los trabajos siempre se ejecutarán en la misma máquina.
- Un *Mainframe* en comparación con un sistema distribuido no tiene mayor poder de cómputo.

- No se puede añadir poder de cómputo en pequeños incrementos, debido a lo complicado de esta operación.

2.2.2 Bases de Datos Distribuidas (BDD). Una base de datos distribuida es una base de datos construida sobre una red computacional y no por el contrario en una máquina aislada, la información que constituye la base de datos esta almacenada en diferentes puntos o nodos en la red, y las aplicaciones que se ejecutan, accesan datos en distintos sitios.

2.2.2.1 Ventajas. Las BDD tienen múltiples ventajas, en primer lugar los datos son localizados en lugar más cercano, por tanto, el acceso es más rápido, el procesamiento es rápido debido a que varios nodos intervienen en el procesamiento de una carga de trabajo, nuevos nodos se pueden agregar fácil y rápidamente. La comunicación entre nodos se mejora, los costos de operación se reducen, la probabilidad de que una falla en un solo nodo afecte al sistema es baja y existe una autonomía e independencia entre los nodos.

Las razones por las que compañías y negocios migran hacia bases de datos distribuidas incluyen razones organizacionales y económicas, para obtener una interconexión confiable y flexible con las bases de datos existente, y por un crecimiento futuro. El enfoque distribuido de las bases de datos se adapta más naturalmente a la estructura de las organizaciones, además, la necesidad de desarrollar una aplicación global (que incluya a toda la organización), se resuelve fácilmente con bases de datos distribuidas.

Si una organización crece por medio de la creación de unidades o departamentos nuevos, entonces, el enfoque de bases de datos distribuidas permite un crecimiento suave, los datos se pueden colocar físicamente en el lugar donde se accesan frecuentemente, haciendo que los usuarios 'tengan' control local de los datos con los que interactúan. Esto resulta en una autonomía local de datos permitiendo a los usuarios aplicar políticas locales respecto del tipo de accesos a sus datos.

Mediante la replicación de información, las bases de datos distribuidas pueden presentar cierto grado de tolerancia a fallas haciendo que el funcionamiento del sistema no dependa de un solo lugar como en el caso de las bases de datos centralizadas.

2.2.2.2 Desventajas. La principal desventaja se refiere al control y manejo de los datos, dado que éstos residen en nodos diferentes y se pueden consultar por nodos diversos de la red, la probabilidad de violaciones de seguridad es creciente si no se toman las precauciones debidas. La habilidad para asegurar la integridad de la información en presencia de fallos no predecibles tanto de componentes de hardware como de software es compleja. La integridad se refiere a la consistencia, validez y exactitud de la información, debido a que los datos pueden estar replicados, el control de concurrencia y los mecanismos de recuperación son mucho más complejos que en un sistema centralizado.

2.3 EL MODELO RELACIONAL

El modelo relacional representa a los datos y las relaciones entre los datos como una colección de tablas, cada una de las cual tiene un numero de columnas con nombres únicos. (Véase la Figura 5). Una relación consiste en enlace que puede establecerse entre dos tablas que poseen algunos campos con valores idénticos.

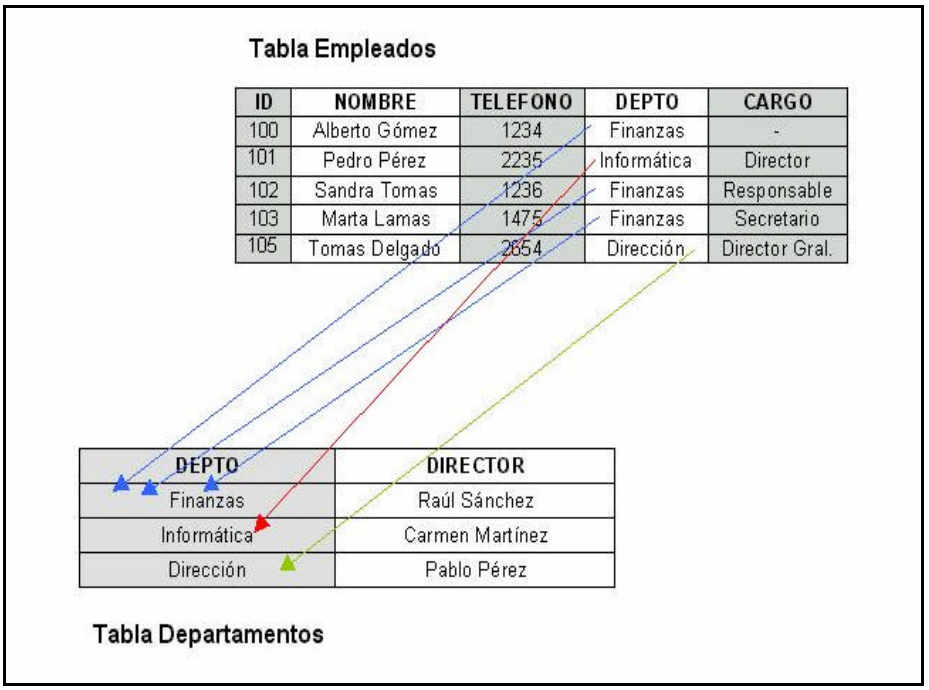

Figura 5. Ejemplo de relación entre tablas

2.3.1 Bases de datos relaciónales. El nombre proviene de su gran ventaja sobre las bases de datos de texto plano: la posibilidad de relacionar varias tablas de datos entre sí, compartiendo información y evitando la duplicidad y los problemas que ello conlleva: espacio de almacenamiento y redundancia. Sin

embargo, tienen un punto débil: la mayoría de las bases datos relacionales del momento no admite la incorporación de objetos multimedia, tales como sonidos, imágenes o animaciones.

2.3.2 El lenguaje SQL. Las bases de datos almacenan los datos con cierta inteligencia, y, por lo tanto, es necesaria para diseñarlas realizar un considerable esfuerzo de abstracción y normalización en el establecimiento de lar relaciones que las estructuren. La siguiente etapa consiste en disponer de un medio de acceso ágil y natural a los mismos, la herramienta existente, comúnmente aceptada como norma, para tal propósito es el lenguaje *SQL*.

SQL cuyo nombre es una abreviatura de *Structured Query Language,* es una es una herramienta para organizar, gestionar y recuperar datos almacenados en una base de datos. Este lenguaje informático es el que se utiliza para interactuar con una base de datos, más concretamente con las bases de datos relaciónales, SQL es a la vez un lenguaje fácil de aprender y una herramienta completa para gestionar datos, las peticiones sobre los datos se expresan mediante sentencias, que deben escribirse de acuerdo con unas reglas sintácticas y semánticas de este lenguaje.

2.3.3 Elementos básicos del lenguaje SQL. El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado, estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

2.3.3.1 Comandos. Existen dos tipos de comandos SQL:

- Los *DDL (Data Definition Languaje),* Lenguaje de Definición de Datos*,* que permiten crear y definir nuevas bases de datos, campos e índices.
- los *DML (Data Manipulation Languaje),* Lengua.je de Manipulación de Datos, que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.

Tabla 3. Comandos DDL

Comando	Descripción
CREATE	Utilizado para crear nuevas tablas, campos e índices.
DROP	Empleado para eliminar tablas e índices.
ALTER	Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.

Tabla 4. Comandos DML

Comando	Descripción
SELECT	Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado.
INSERT	Utilizado para cargar lotes de datos en la base de datos en una única operación.
UPDATE	Utilizado para modificar los valores de los campos y registros especificados.
DELETE	Utilizado para eliminar registros de una tabla de una base de datos.

2.3.3.2 Cláusulas. Las cláusulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular.

Tabla 5. Cláusulas

Cláusula	Descripción
FROM	Utilizada para especificar la tabla de la cual se van a seleccionar los registros.
WHERE	Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar.
GROUP BY	Utilizada para separar los registros seleccionados en grupos específicos.
HAVING	Utilizada para expresar la condición que debe satisfacer cada grupo.
ORDER BY	Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico.

2.3.3.3 Operadores. Estos especifican condiciones y realizan operaciones lógicas y numéricas.

Tabla 6. Operadores lógicos

Operador	Descripción
AND	Es el "y" lógico. Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.
OR	Es el "o" lógico. Evalúa dos condiciones y devuelve un valor de verdad si alguna es cierta.
NOT	Negación lógica. Devuelve el valor contrario de la expresión.

Tabla 7. Operadores numéricos

Operador	Descripción
=	Igual.
<>	Diferente de.
<	Menor que.
>	Mayor que.
<=	Menor o igual que.
>=	Mayor o igual que.
+	Adición.
-	Sustracción.
/	División.
*	Multiplicación.
BETWEEN	Utilizado para especificar un intervalo de valores.
LIKE	Utilizado en la comparación de un modelo.
IN	Utilizado para especificar registros de una base de datos.

2.3.3.4 Funciones de Agregado. Las funciones de agregado se usan dentro de una cláusula *SELECT* en grupos de registros para devolver un único valor que se aplica a un grupo de registros.

Tabla 8. Funciones de agregado

Función	Descripción
AVG	Utilizada para calcular el promedio de los valores de un campo determinado.
COUNT	Utilizada para devolver el número de registros de la selección.
SUM	Utilizada para devolver la suma de todos los valores de un campo determinado.
MAX	Utilizada para devolver el valor más alto de un campo especificado.
MIN	Utilizada para devolver el valor más bajo de un campo especificado.

2.4 SISTEMAS GESTORES DE BASES DE DATOS (SGBD)

Actualmente las bases de datos se diseñan a partir de determinados modelos estándares de diseños para bases de datos), donde estos modelos buscan una independencia total de los datos y del modo físico concreto en el que estos, están almacenados en dicha base de datos.

2.4.1 Gestor o motor de base de datos. En los sistemas de bases de datos convencionales (modelo relacional), existe un elemento imprescindible y relevante como es el gestor de base de datos, este gestor o motor de base de datos es el encargado de manejar la información contenida en la base de datos, facilitado el acceso a esta y evitando que se produzcan errores o incoherencias en el momento de adicionar, eliminar o modificar datos en la base de datos.

Dentro de los gestores relevantes en la actualidad se pueden citar *Access, MS SQL Server, Interbase, Oracle,* o *Paradox*.

En el medio de la informática y más exactamente en lo que concierne a las bases de datos, hoy día se impone el modelo de base de datos relacional, estas se diseñan a partir de un modelo que facilita el entendimiento de ver los datos como algo independiente de su modo de almacenamiento y manipulación. Esto plantea en teoría que se puede utilizar cualquier aplicación para acceder a cualquier base de datos.

La anterior coyuntura permite crear una misma base de datos en diversos gestores, es decir, puede diseñarse una base de datos que funcione con cualquier gestor, lógicamente para que esto sea una realidad se deberán utilizar aquellas estructuras comunes a todos los gestores de base de datos, para lograrlo. De acuerdo a la conceptualización expuesta se puede indicar que existen al menos dos modelos de gestores para bases de datos, los cuales se describen a continuación.

2.4.1.1 Gestores basados en el modelo Cliente - Servidor. Los gestores que utilizan esta arquitectura separan la aplicación que maneja los datos (servidor), de las aplicaciones (cliente), que muestran y actualizan la información para cada usuario. Las principales labores de un sistema gestor de este tipo son:

- Creación y mantenimiento de la base de datos.
- Mantenimiento de la seguridad y control de accesos.
- Generación de una relación *(log)* de las operaciones solicitadas al gestor para construir la base de datos a partir de una copia de seguridad en caso de un error del sistema.
- Mantenimiento de la integridad referencial.
- Manejo de concurrencia sobre los datos.
- Interpretación de peticiones de los clientes y devolución de datos como respuesta.

Una de las principales ventajas de este sistema radica en que, es el propio gestor quien lleva a cabo todas estas labores de manera que no debe añadirse el código para realizarlas, a cada una de las aplicaciones que solicitan datos al gestor de base de datos.

2.4.1.2 Gestores tradicionales de bases de datos. También son conocidos con la descripción de “gestores de bases de datos de sobremesa” o simplemente *desktop*, se caracterizan porque poseen un formato propio para los ficheros que almacenan las tablas y otros objetos, como los índices de la base de datos. Estos sistemas funcionan sobre computadores personales, con pocos requerimientos de hardware. La principal diferencia que existe entre los gestores de escritorio y los gestores cliente – servidor estriba en que la información no se distribuye de manera tan sencilla, debido a que no están claramente separadas la parte de

aplicación y la gestión de la base de datos. El uso de este tipo de gestor se da mas en el ámbito de las oficinas, dentro de los cuales podemos citar *dBase o paradox,* estos combina sus características de motores de bases de datos con la incorporación en su *kernel* de un cierto tipo de interprete que ejecuta el código de la aplicación. En ellos se crea un archivo para cada tabla (por ejemplo para el caso de dBase *.DBF*), lo cual tiene desventajas como por ejemplo el detrimento de la seguridad de la base de datos. Además en la mayoría de este tipo de gestor se hace necesario escribir el código necesario para preservar la integridad referencial.

Dentro de este tipo de gestor se encuentra uno que se puede denominar de tipo mixto como es *Microsoft Access,* este utiliza un único archivo para albergar la base de datos completa (incluyendo informes o procedimiento varios) y se encarga de mantener la integridad referencial ade más del acceso seguro y concurrente de los usuarios a los datos que están contenidos en la base de datos.

2.5 LAS BASES DE DATOS EN EL WEB

El Web es un medio para localizar / enviar / recibir información de diversos tipos, en el ámbito competitivo, es esencial ver las ventajas que esta vía de comunicación electrónica proporciona para presentar la información, reduciendo costos y el almacenamiento de ésta, aumentado la rapidez de difusión de la misma.

El Web provee un formato de presentación dinámico para ofrecer campañas y mejorar negocios, además de permitir el acceso a cada sitio alrededor del mundo, en la actualidad se ha tomado como el principal medio para realizar negocios y la principal fuente de información. Una gran parte de ésta información requiere de un manejo especial y puede ser provista por bases de datos.

Con estos propósitos, los usuarios de *Internet* o una *Intranet* pueden obtener un medio que puede adecuarse a las necesidades de intercambio de información de su ámbito, con un costo, inversión de tiempo y recursos mínimos, asimismo, las bases de datos serán usadas para permitir el acceso y manejo de la variada información que se encuentre en la red. Muchas instituciones se han dado cuenta de la importancia que el espacio virtual tiene en el desarrollo de sus potencialidades, ya que con ello pueden lograr una mejor comunicación con personas o instituciones situadas en cualquier lugar del mundo.

Una de las ventajas de utilizar el Web para este fin, es que no hay restricciones en el sistema operativo que se debe usar, permitiendo la conexión entre sí de las paginas desplegadas en el navegador del cliente basado en una plataforma diferente a las plataformas donde residen el servidor Web que la emite y el servidor que contiene la base de datos. Además de que no hay que cambiar el formato o estructura de la información dentro de la base de datos.

3. TECNOLOGÍAS PARA INTERCAMBIAR INFORMACIÓN A TRAVÉS DEL WEB

Las aplicaciones Web, conllevan necesariamente a que tengan que implementarse métodos que de alguna manera procese la información que el usuario envía al servidor, normalmente a través de formularios. El procesamiento de los datos puede suponer la comunicación con las interfaces cliente de bases de datos para acceder a estas o la ejecución de rutinas que extraigan información o provoquen respuestas personalizadas. La Figura 6, muestra una vista general de los elementos requeridos para acceder una base de datos desde el Web.

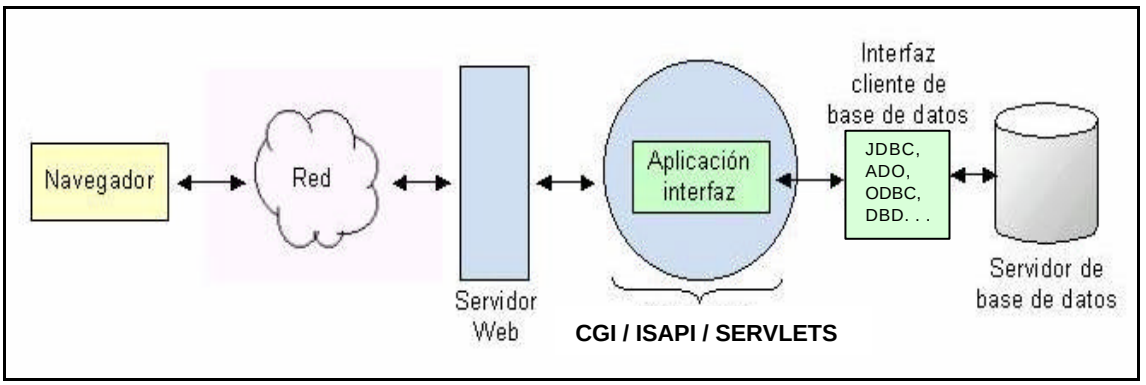

Figura 6. Elementos que intervienen en el intercambio de datos a través del Web

La arquitectura mas extendida para la realización de estos programas ha venido siendo el ***Common Gateway Interface (CGI)***; pero últimamente se estan imponiendo las propuestas tecnológicas ***Internet Server Applicaton Programming Interface (ISAPI)*** de *Microsoft* (y las variantes de este, como es el de la empresa *Netscape* denominada *NSAPI),* y la tecnologia basada en ***Servlets*** de la empresa *Sun Microsystems,* desarrollada en Java, para las aplicaciones hechas en Java.

3.1 TECNOLOGÍA CGI

CGI *(Common Gateway Interface)* es un método estándar para la comunicación entre diferentes procesos, esta técnica es utilizada en servidores Web para procesar las solicitudes obtenidas a través de los navegadores que presenten paginas Web de dicho servidor. Este proceso puede conllevar la consulta de una base de datos, o el simple tratamiento de la información que el cliente ha introducido para la elaboración de una respuesta.

CGI no es un lenguaje de programación. Es simplemente un protocolo que puede ser usado para comunicar formularios Web y un programa. Un *script CGI* puede ser escrito en cualquier lenguaje que pueda leer de la entrada estándar o *STDIN*, escribir en la salida estándar o *STDOUT*, y leer variables de entorno como cualquier lenguaje de programación. Algunos de los lenguajes comúnmente utilizados para escribir programas o *scripts CGI* son:

- C
- PERL
- Java
- C Shell
- Python
- TCL/Tk
- VisualBasic
- PHP

El CGI es un programa que se ejecuta en un servidor como respuesta a una petición de usuario a través de un cliente. (Véase la Figura 7).

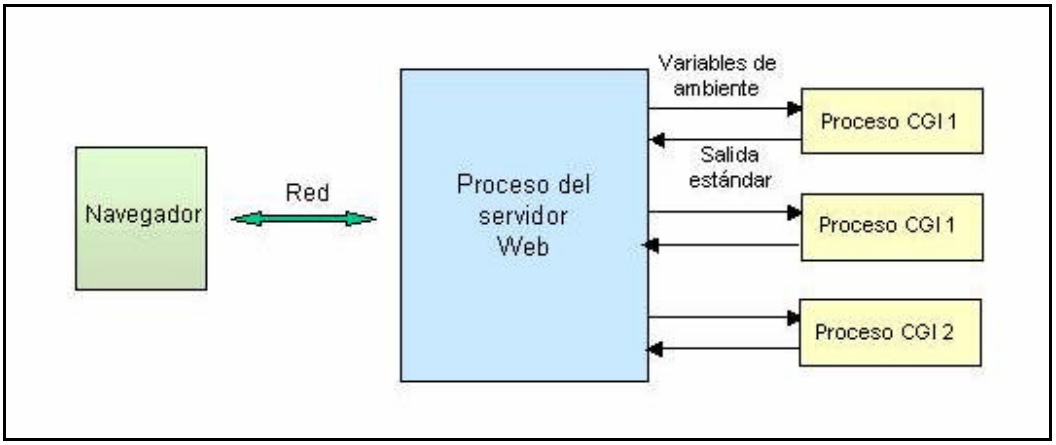

Figura 7. Esquema da la tecnología CGI

Los CGI son llamados por el servidor basándose en la información que le proporciona el navegador, cuando esto sucede el servidor ejecuta un proceso hijo

que recibirá los datos que envía el usuario (en caso de que los haya), pone a disposición del mismo algunos datos en forma de variables de ambiente y captura la salida del programa para enviarlo como respuesta al navegador. Permitiendo generar páginas HTML dinámicas, esto es, páginas HTML cuyo contenido puede variar y que por lo tanto no pueden almacenarse en un fichero en el servidor.

El proceso podría ser el siguiente:

1. El navegador demanda un *URL* (el del *CGI*) al servidor.
2. El servidor ejecuta el *CGI*.
3. El *CGI script* envía la información resultante de la ejecución al servidor.
4. El servidor envía esta información al navegador.
5. El navegador le da el formato y la presenta al usuario.

3.1.1 Formularios HTML. Los formularios son un mecanismo para permitir que las paginas con contenidos Web sean interactivas, recogiendo información de los usuarios para ser transmitida al servidor Web. (Véase la Figura 8). En este paso de transmisión de información al servidor para su proceso existen dos pasos: la presentación en el documento HTML de la interfaz a través de la cual el usuario enviara la información, y la tarea del procesado de la misma. Para la creación de los formularios se utiliza la etiqueta HTML *<FORM>*, esta etiqueta en su versión básica posee usualmente dos atributos: *METHOD* y *ACTION.*

Figura 8. Ejemplo de un formulario en un navegador Web

El siguiente es un ejemplo del uso de esta etiqueta:

```
<FORM METHOD = método ACTION = “URL del script a ejecutar> ...
</FORM>
```

3.1.2 Método de envió de datos al servidor. El atributo *METHOD* permite especificar la manera en como los datos serán enviados desde el navegador al servidor, este puede tomar varios valores: *GET*, *POST, HEAD, PUT, DELETE, TRACE, CONNECT* y *OPTIONS*.

Generalmente se utiliza los métodos *GET* y *POST*, estos son dos métodos diferentes definidos en el protocolo *HTTP* que hacen cosas bastante diferentes; pero ambos son capaces de enviar remisiones de formularios al servidor.

3.1.2.1 GET. Normalmente, *GET* es usado para obtener un archivo u otro recurso, posiblemente con parámetros, especificando más exactamente lo que se necesita. Al utilizar este valor, la información se enviara al servidor adherida al final del *URL* del *script,* como si se le hubiera añadido al final del mismo, por ejemplo:

```
URL_del_programa?variable1=valor1&variable2=valor2 ...
```

Cuando la petición de datos llegue al servidor, este examinara el *URL* y colocara los argumentos que le siguen en una variable denominada *QUERY_STRING*. *GET*, es la manera, como el navegador baja la mayoría de los archivos, como archivos HTML e imágenes.

El método *GET* es *ídem potente*, lo cual significa que el efecto lateral de muchas peticiones *GET* idénticas, es el mismo que para una sola petición *GET*, en particular, los navegadores y *proxies* pueden obtener respuestas *GET* del caché, así que dos remisiones de formas idénticas podrían no llegar al *script CGI*. Una ventaja de *GET* es que la remisión completa del formulario puede ser encapsulada en un *URL*, como un hiperenlace o un marcador.

3.1.2.2 POST. En este caso la información se envía separadamente al servidor, no incluida en el *URL* del *script.* Cuando un formulario HTML se remite usando *POST*, Los datos del formulario se atan al final de la petición *POST* en su propio objeto, esto no es tan rápido ni tan fácil como al usar *GET*; pero es mucho más versátil y seguro, porque no existe ningún tipo de limitación sobre la cantidad de información que puede enviarse.

Cuando la petición de datos llegue, el servidor colocara los argumentos que le siguen en una variable de entorno denominada *CONTENT_LENGTH*.

Algunas ventajas de *POST* son que no hay limite en los datos que se van a remitir, los datos a enviar no se muestran en URL del destino y que el *script* será llamado cada vez que el formulario sea remitido.

3.1.3 Elementos del formulario. Un formulario esta compuesto por un conjunto de elementos, cajas de edición de texto, listas desplegables, botones; etc. Cada uno de estos elementos es codificados en *HTML* con la etiqueta <*INPUT*>, aunque existen otras etiquetas que también permiten definir elementos, el siguiente ejemplo muestra el uso de esta etiqueta:

```
<INPUT TYPE = “tipo” NAME = “Nombre”> ...</INPUT>
```

La etiqueta acepta básicamente dos atributos:

- *TYPE:* determina el tipo de elemento que se creara, se pueden definir cajas de texto, botones de pulsación, botones de radio; etc.
- *NAME:* indica el nombre del elemento, a efectos de identificarlo en el envío al servidor de la información generada, de hecho, el navegador enviara al servidor un conjunto de parejas del tipo <*nombre, valor*>.

Otros atributos son: *ALIGN, CHECKED, MAXLENGTH, SIZE, SRC, VALUE*.

3.1.4 Programa que procesara el formulario. El atributo *ACTION* contiene el nombre *(URL)* del *script* que procesara en el servidor los datos obtenidos a través del formulario.

Por ejemplo, si el *script* asociado al formulario fuese el programa **proceso.cgi** situado en el directorio ***cgi-bin*** (donde generalmente residen los *scripts*), del servidor se deberá escribir la siguiente etiqueta:

```
<FORM METHOD="get"  ACTION="http://www.cutb.edu.co/cgi-bin/
proceso.pl">
```

3.1.4.1 Procesando la entrada del formulario. Cuando el usuario envía la forma, el *script* toma los datos como un conjunto de pares *nombre-valor*, de las variables de ambiente *CONTENT_LENGTH* o *QUERY_STRING*, dependiendo del valor asumido por el atributo *METHOD*. Este conjunto de pares nombre-valor la recibe como una sola larga cadena, la cual debe ser procesada. Por ejemplo:

URLprograma?**variable1=valor1&variable2=valor2&Variable3=valor3**

Los pasos del *script* o programa, para realizar esta tarea serian:

1. Separar las parejas dividiendo los *ampersands* (signo &).
2. Separar los pares de los signos igual.
3. Para cada nombre y valor se hacen dos cosas más:
    - Convertir todos los caracteres "+" en espacios, y
    - Convertir todas las secuencias "%xx" en el carácter cuyo valor ASCII sea "xx", en hexadecimal. Por ejemplo, convierte "%3d" en "=".

3.1.5 Variables de entorno CGI. Los *scripts* y programas *CGI* tienen acceso a cerca de 20 variables de entorno, como *QUERY_STRING* y *CONTENT_LENGTH* mencionadas anteriormente. La siguiente tabla contiene las variables de entornos definidas en *NCSA.*

Tabla 9. Variables de entorno

Variable de Ambiente	Descripción
AUTH_TYPE	Método de autentificación usado por el servidor.
CONTENT_LENGTH	Longitud del contenido enviado por el cliente.
CONTENT_TYPE	El tipo de datos del contenido, tal como "text/html" .
DOCUMENT_ROOT	El directorio donde se encuentran los documentos HTML. Por ejemplo: /web/members/htdocs.

GATEWAY_INTERFACE	La versión o especificación del CGI que usa el servidor. Ejemplo: CGI/1.1
HTTP_ACCEPT	Una lista de los tipos de datos que el cliente puede aceptar.
HTTP_FROM	Correo electrónico del cliente que envía el formulario.
HTTP_REFERER	El URL origen del formulario qué el cliente remite antes de acceder el programa CGI. No siempre se usa.
HTTP_USER_AGENT	Agente de usuario para realizar la solicitud.
PATH_INFO	Información extra de ubicación. No siempre se usa.
QUERY_STRING	Contiene la cadena de consulta, si la consulta del cliente corresponde al método GET.
REMOTE_ADDR	La dirección IP de la maquina que ejecuta el navegador.
REMOTE_HOST	El nombre de la maquina que realiza la petición.
REMOTE_IDENT	Identidad del usuario que realiza la solicitud. No siempre se usa.
REMOTE_USER	Contiene el *Userid* de la persona que esta usando el navegador.
REMOTE_METHOD	El método usado para hacer la petición (normalmente GET o POST). No siempre se usa.
REQUEST_METHOD	El método HTTP con el que el script fue llamado. Generalmente *GET, POST, o HEAD*
SCRIPT_MAP	Fragmento base del URL.
SCRIPT_NAME	*Path* virtual del script. El URL local del script que está siendo ejecutado

SERVER_NAME	Nombre de la maquina donde se ejecuta el servidor. Ejemplo: cutb.edu.co.
SERVER_PORT	Indica el numero del Puerto del servidor al que se hizo la petición (por defecto es el puerto 80).
SERVER_PROTOCOL	Indica el protocolo en uso. Por ejemplo HTTP/1.1
SERVER_SOFTWARE	Nombre y versión del software del servidor.

3.1.6 Ventajas y desventajas de la tecnología CGI. La acción de ejecutar programas en el servidor en respuesta a una solicitud hecha por un usuario en el cliente, compromete varios puntos:

- El usuario no se tiene que preocupar de si el programa funciona o no, ya que no lo ejecuta el cliente.
- No hay que tener en cuenta los aspectos de compatibilidad de su sistema con el sistema del servidor Web.
- Es un programa listo para ejecutar, no hay que traducirlo ni compilarlo. Sólo ejecutarlo, lo que le da mas velocidad, y por lo tanto mayor sensación de interactividad.
- El usuario no va ejecutar ningún programa del que no sabe qué efectos nocivos podría tener en su maquina, por tanto para él, no conlleva problema de seguridad en este punto.
- No hay problemas de mantenimiento y distribución por parte del servidor Web, ya que la comunicación es estándar, a través del protocolo HTTP.

- Un inconveniente de ejecutar los programas en el servidor, es que por cada cliente que accede a la página y ejecuta el programa, hay que hacer una copia del mismo en memoria, ya que no puede hacer uso de recursos compartidos.
- Al ser *CGI* simplemente un punto de encuentro entre dos extremos, no se tiene control directo sobre la comunicación.
- Tener un programa *CGI* en el servidor significa que cualquier persona desconocida que acceda a la pagina, puede ejecutar un programa en este. Un simple programa situado en cualquier sitio puede ser una puerta abierta para cualquier intruso. La manera más fácil y empleada por casi todos los sistemas para remediar esto, es guardar todos los programas en un directorio llamado *cgi-bin*, en el que los usuarios solo van a tener permiso de ejecución.

3.2 TECNOLOGÍA ISAPI

ISAPI (Internet Server Application programming interface), es el nombre del protocolo de extensión que admite el *IIS (Internet Information Server)* de *Microsoft*. Esta tecnología permite crear aplicaciones que se comuniquen con el servidor Web par realizar diversas tareas. ISAPI es, como su nombre lo indica, un API, es decir un conjunto de funciones y elementos de programa que pueden usarse para crear aplicaciones que colaboraran con el servidor *IIS*.

3.2.1 Extensiones. Una extensión *ISAPI* es un modulo de programa que contiene código que puede ejecutarse desde el servidor Web cuando el usuario,

mediante una petición formulada desde su navegador, lo solicita. El código de una extensión se encuentra en un archivo *DLL,* con un conjunto de funciones de interfaz bien definido, que es cargada por el servidor cuando es solicitada y es mantenida en memoria. Las extensiones son la alternativa a CGI, siguiendo la tecnología ISAPI. La entrada de parámetros se realiza a través de los propios parámetros de la función de comunicación y lo mismo sucede con la respuesta.

3.2.2 Filtros. Un filtro ISAPI, por otra parte es una *DLL* que se carga junto con el servidor para filtrar los datos que este recibe y envía, cuando ciertas situaciones, caracterizadas por ciertos eventos, se producen. Los filtros a diferencia de las extensiones no se ejecutan bajo demanda. El sistema sabe que filtros deben cargarse examinado una clave del registro, esta clave contiene una lista de todas los archivos de extensión DLL a cargar cuando se inicie el servidor.

La aplicación típica para los filtros es la de procesar los datos que se envían a los usuarios, para obtener y anotar información de utilización, para comprimir o encriptar datos o cualquier otra operación.

3.3 ISAPI vs. CGI

La principal diferencia entre el modelo de programación CGI y el modelo de programación ISAPI, se centra en que el modelo CGI crea un único proceso para cada solicitud, mientras que el modelo ISAPI no.

Con CGI, cada vez que el servidor HTTP recibe una solicitud, debe iniciar un nuevo proceso, el mantenimiento de estos procesos consume grandes recursos. Esta inherente limitación del modelo CGI, dificulta el respectivo desarrollo de aplicaciones en Internet.

En el modelo ISAPI cada, solicitud recibida por un servidor HTTP inicia la creación de una estructura de datos ECB. La creación y mantenimiento de una estructura de datos es mucho más fácil y rápido que iniciar un nuevo proceso, en consecuencia, puesto que la ECB y la extensión usualmente corren en el mismo proceso de servidor, el servidor puede procesar solicitudes rápidamente y alojar un mayor volumen de estas. CGI es un estándar multiplataforma mientras ISAPI solo funciona en sistemas *MS Windows.* En la Figura 9, se ilustra la diferencia entre el modelo CGI y el modelo ISAPI.

Finalmente, mejor que utilizar el aislamiento de procesos, el modelo ISAPI usa hilos para aislar el procesamiento de los diferentes trabajos. Usar múltiples hilos para sincronizar los trabajos, permite al servidor utilizar de una manera más eficiente los recursos del sistema, que el modelo CGI o cualquier otro proceso basado en procesos aislados.

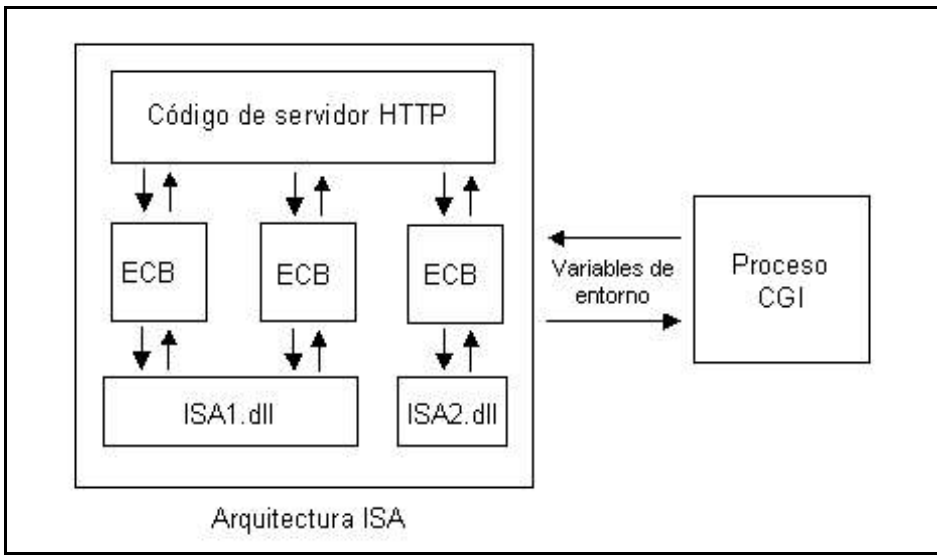

Figura 9. Diferencia entre el modelo CGI y el modelo ISAPI

3.4 LENGUAJES DE PROGRAMACIÓN PARA INTERNET

El objetivo principal de los lenguajes de programación, es el de proveer una herramienta mediante la cual los desarrolladores, puedan implementar las aplicaciones que permitan a los usuarios interaccionar con los sistemas. Los lenguajes de programación que permitan la creación de aplicaciones Web no se apartan de este objetivo, por lo contrario, en estos se ve un énfasis en el mismo.

Los lenguajes de programación para Internet se comportan de la misma manera que los lenguajes de programación tradicionales y en algunos casos, existe cierta integración de los lenguajes tradicionales con las páginas Web.

Como en casi todo lo relacionado con Internet, existen los lenguajes de programación para sistemas *Microsoft Windows* y otros para sistemas *UNIX*. A continuación se presentan algunos lenguajes de programación.

3.4.1 Active Server Pages (ASP). ASP es un marco de trabajo independiente del lenguaje, diseñado por *Microsoft* para codificar *scripts* de *server-side* que sean desarrollados para ser ejecutados por un servidor en respuesta a una requisición de un usuario. El programador puede desarrollar *scripts* con ASP codificando ya sea en *VBScript*, *Jscript* o *JavaScript.* También puede codificar en *PerlScript*, utilizado mayormente para compatibilidad con *scripts* Perl. Además con ASP se pueden acceder bases de datos, principalmente las bases de datos de *Microsoft: SQL Server* y *Access*. Sin embargo puede acceder cualquier otro gestor de base de datos, a través de cualquier interfaz cliente soportada.

ASP es de hecho un lenguaje de *scripts* que se ha convertido en el estándar para servidores *Windows NT* con *IIS*. Tiene la real capacidad de generar aplicaciones tradicionales, tales como financieras, de producción, de administración; etc., todo con interfaz estilo páginas Web.

3.4.2 Visual InterDev. *Visual InterDev*, más que un lenguaje de programación, es un *software* de administración de proyectos de desarrollo de software, el cual integra una gran cantidad de herramientas *Microsoft* orientadas al desarrollo de sistemas de información para Internet. Entre sus principales características se destacan:

- Soporta ASP para generar páginas Web en línea.
- Soporta acceso a bases de datos.
- Soporta programación *VBScript* y *JScript* en las páginas Web.
- Se integra con herramientas de desarrollo visual, formatos preestablecidos *(templates)* y ayudas *(wizards)*, desarrollo orientado a objetos; etc.
- Se integra con una versión *WYSIWYG (What you see is what you get)* de la herramienta de desarrollo de páginas HTML de *Microsoft, FrontPage*.
- Soporta herramientas de administración de proyectos de desarrollo Web.

Visual InterDev es una valiosa herramienta de desarrollo de sistemas para Internet con capacidades para desarrollar aplicaciones muy complejas. Pero ha mostrado tener dos limitaciones:

- El tamaño de la herramienta hace que ésta sea demasiado lenta de ejecutar.
- El desarrollo está orientado, de manera natural, para el servidor *Internet Information Server* por lo que queda descartada para ambientes UNIX.

3.4.3 Perl. Perl significa *Practical Extraction and Report Language*, lenguaje práctico de extracción y de informes. Es un lenguaje creado por *Larry Wall* con el objetivo principal de simplificar las tareas de administración de un sistema UNIX; hoy en día se ha convertido en un lenguaje de propósito general, y una de las principales herramientas de los desarrolladores del Web. No establece ninguna filosofía de programación concreta.

No se puede decir que sea orientado a objetos, modular o estructurado aunque soporta directamente todos estos paradigmas y su punto fuerte son las labores de procesamiento de textos y archivos.

No es ni un compilador ni un interprete, esta en un punto intermedio, cuando se ejecuta un “programa” en Perl, se compila el código fuente a un código intermedio en memoria que se optimiza como si se fuera a elaborar un programa ejecutable pero es ejecutado por un motor, como si se tratara de un interprete. Sus características son:

- Extensibilidad: gran parte del crecimiento de Perl se debe por el creciente uso de bibliotecas y módulos. Esto permite a los desarrolladores escribir porciones de código contenidas en si mismas que pueden incorporarse en una aplicación Perl.
- Seguridad: al emplear para la formulación de *scripts* en un servidor Web, se puede evitar con facilidad que los usuarios intenten emplear comandos furtivamente para que el servidor los ejecute a su beneficio.
- Se pueden acceder la mayoría de bases de datos relacionales existentes.
- Es gratuito y de código libre.
- Soporte multiplataforma.

3.4.4 PHP. PHP *(Hypertext Preprocessor)* es un lenguaje de programación de *scripts* cuyas características principales son:

- Está orientado a *scripts server-side.*
- El lenguaje tiene la capacidad de ser inmerso dentro del código HTML de páginas Web, es decir, los *scripts* están escritos dentro del código HTML que define la página. En ese sentido difiere de otros como Perl ó C, y es mucho más parecido a ASP.
- PHP es un lenguaje de programación muy poderoso. Al nivel de ASP, sólo que es mayormente utilizado en plataforma UNIX por lo que su desempeño es muy superior a ASP bajo programas de la misma complejidad.

Además tiene otra característica que lo hace más atractivo y es que es un lenguaje de *fuente abierta*. PHP maneja el mismo estilo de programación que el lenguaje C y que Perl. Es una excelente alternativa a Perl para la interfaz con las páginas Web. Tiene la ventaja sobre Perl, que se programa dentro del código HTML y que al estar instalado como un componente del software de servidor Web, no tiene que levantar varias copias del mismo programa una vez que éste se ejecute.

3.4.5 Java. Java es un lenguaje de programación desarrollado por *Sun Microsystem* y utilizado para crear contenido ejecutable que puede distribuirse a través de redes computacionales como Internet. Usado de manera genérica, el nombre Java se refiere a un grupo de herramientas de software que permite crear e implantar contenido ejecutable utilizando el lenguaje de programación Java.

Aunque la máxima potencialidad de Java se presenta para *scripts* o programas centrados en el cliente o navegador *(scripts client-side)*, Java también tiene la capacidad de generar código ejecutable para el servidor *(scripts server-side)*.

La gran ventaja de Java sobre todos los demás lenguajes de programación es precisamente esta característica de manejar tanto el lado del navegador como del servidor. El problema principal que Java ha tenido, es que ninguna de las grandes tendencias de Internet *(UNIX y Windows NT)* lo ha tomado como lenguaje estándar de procesamiento ni en el cliente ni en el servidor.

4. INTERFACES ENTRE LAS APLICACIONES Y LOS GESTORES DE BASES DE DATOS

La gran cantidad de motores de bases de datos, de arquitecturas y sintaxis muy diversas, conlleva en que las aplicaciones que acceden a los datos deban, en principio, someterse a las particularidades del motor de base de datos al momento de interactuar con este. Esto motivaría a que las aplicaciones sirvieran exclusivamente para un gestor en particular y fuese necesario para el desarrollador conocer una gran cantidad de sintaxis de acceso para los mismos.

Para evitar estos problemas se han creado variadas y diferentes interfases que permiten un acceso a las diferentes plataformas de datos utilizando medios comunes.

Esta interfaz, que también se conoce como *tecnología cliente del gestor de base de datos*. Suele estar conformada por un *software* controlador que estandariza las llamadas a los motores de bases de datos. De esta manera se definen funciones que pueden ser utilizadas para acceder a los datos de cualquier gestor que disponga de dicho controlador. Este traduce las llamadas al lenguaje y sintaxis propia del gestor.

La cuestión de cuando y donde usar esta tecnología puede causar mucha confusión, a menos que se conozca muy bien cada tecnología y su relación con las demás.

El propósito de la interfaz cliente de base de datos, es proporcionar principalmente un medio de abstracción. Un gestor de bases de datos es una pieza de *software* muy compleja, escribir programas para comunicarse con una base de datos a través de su interfaz nativa puede ser muy complicado, las tecnologías cliente de bases de datos simplifican este proceso.

Estas tecnologías proporcionan una interfaz, un medio para el desarrollador, menos compleja que la API proporcionada por el gestor de base de datos. Estas posibilitan que se escriban aplicaciones relativamente sencillas que apalanquen una enorme cantidad de código (código que reside en el gestor), para ejecutar operaciones realmente complejas. Una buena interfaz es como un lente de aumento que maximiza el código, como se muestra en la Figura 10.

Escribir programas que se comuniquen con el gestor de base de datos a través de su *interfaz nativa*, no solo puede ser complejo, también pueden resultar en aplicaciones limitadas e inflexibles. Una aplicación que use una interfaz nativa es limitada a interactuar con una base de datos en particular. Habilitar que esta aplicación interactué con otro gestor de base de datos puede ser muy laborioso sino imposible.

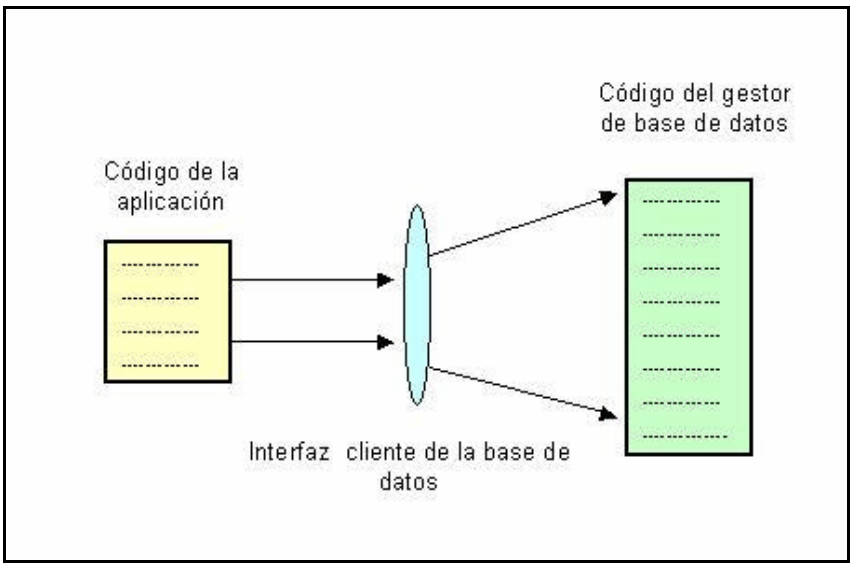

Figura 10. Una interfaz es un amplificador de código

Las tecnologías clientes de bases de datos facilitan una interfase para comunicarse con diferentes y dispares sistemas de bases de datos. Con las modernas interfases clientes de bases de datos, se puede escribir un programa que realice operaciones complejas usando diferentes tipos de bases de datos. (Véase la Figura 11).

Una buena interfaz de base de datos amplia el código y proporciona un medio uniforme para acceder a los diferentes sistemas de bases de datos, en el pasado reciente, varias interfaces de gestores de bases de datos han sido desarrolladas, estas interfases difieren de otras, en la manera en que llevan a cabo la tarea que se les asigna.

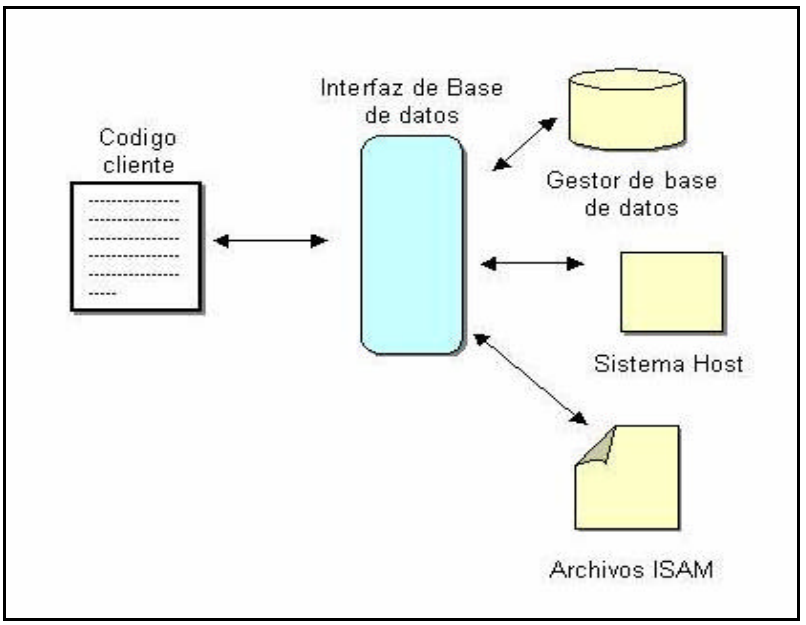

Figura 11. Una interfaz uniforme para diversos tipos de bases de datos

Las interfases de bases de datos más conocidas son:

- *ODBC (open database connectivity).*
- *MFC (Microsoft Foundation Classes) ODBC classes.*
- *DAO (Data Access Objects).*
- *RDO (Remote Data Objects).*
- *OLE DB (object-linking and embedding database).*
- *ADO (ActiveX Data Objects).*
- *JDBC*
- *DBI (Data Base Interface)*

4.1 OPEN DATA BASE CONNECTIVITY (ODBC)

ODBC, Conectividad Abierta de Base de Datos, define un método de conexión a fuentes de datos, de manera que cualquier desarrollador pueda acceder a los datos desde cualquier plataforma que disponga de esta funcionalidad. Del lado de la base de datos, esta conectividad abierta se obtiene mediante el uso de controladores contenidos en archivos *DLL* (en las plataformas *Microsoft*), los cuales transforman las funciones ODBC API en llamadas a funciones soportadas por el gestor de datos, de esta forma se puede acceder a diferentes motores de bases de datos cargando el controlador apropiado.

La interfaz ODBC define lo siguiente:

- Una librería de funciones de llamadas ODBC que permiten a una aplicación conectarse a un gestor de base de datos, ejecutar sentencias SQL, y recuperar resultados.
- Una sintaxis SQL basada en la especificación *X/Open and SQL Access Group (SAG) SQL CAE (1992).*
- Un conjunto estándar de códigos de error.
- Una forma estándar de conectarse y registrarse un sistema manejador de base de datos.
- Una representación estándar de los tipos de datos.

4.1.1 Arquitectura. La arquitectura esta constituida por cuatro componentes:

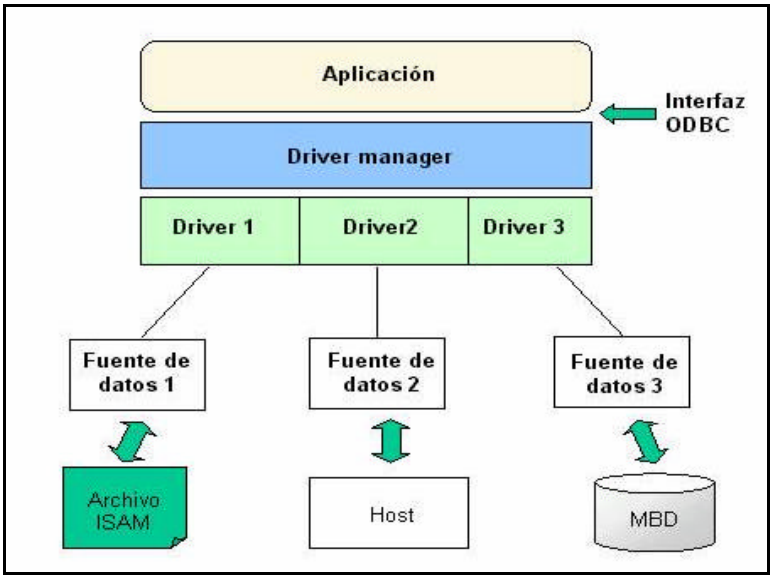

Figura 12. Arquitectura de ODBC

1. La aplicación: lleva a cabo el procesamiento y las llamadas a las funciones ODBC para el envió de consultas SQL y recepción de los resultados.
2. El manejador de controladores *(driver manager)*: carga los controladores en nombre de una aplicación.
3. Controlador *(driver)*: procesa los llamados a las funciones ODBC, envía las solicitudes SQL a un determinado gestor, y retorna el resultado a la aplicación. Si es necesario transforma la solicitud de una aplicación, de tal manera que la solicitud este conforme con la sintaxis soportada por el gestor de base de datos asociado.

4. Fuente de datos: consiste en la información que el usuario quiere acceder y esta asociado con el sistema operativo, el gestor de base de datos y la plataforma de red (algunos casos) usada para accederla.

Múltiples controladores y gestores de datos pueden coexistir, esto permite que la aplicación pueda acceder simultáneamente a la información de varias fuentes de datos. El API ODBC se ubica en dos lugares: entre la aplicación y el manejador de controladores y entre el manejador de controladores y cada controlador. Esta ultima ubicación es llamada algunas veces *service provider interface*, o SPI. Para ODBC el API y el SPI son lo mismo, es decir el manejador de controladores y cada controlador tienen la misma interfaz para las mismas funciones.

4.1.2 Flexibilidad.

- Puede contener cadenas de sentencias SQL incluidas explícitamente en el código fuente o construidas en tiempo de ejecución.
- El mismo código objeto puede ser usado para acceder a diferentes gestores de base de datos.
- Una aplicación puede ignorar los protocolos de comunicación de datos entre este y el gestor.
- Los valores de los datos pueden ser enviados y recibidos en un formato conveniente para la aplicación.
- La interfaz ofrece dos tipos de llamadas de funciones:

- Funciones básicas basadas en la especificación *X/Open and SQL Access Group Call Level.*
- Funciones extendidas que soporten funcionalidad adicional, incluyendo cursores desplazables y procesamiento asíncrono.

4.2 OLE DB Y LA TECNOLOGÍA DE ACCESO UNIVERSAL A DATOS (UDA)

Universal Data Access (UDA) es una tecnología que *Microsoft* propone para el acceso a la información que puede modelarse como susceptible a ser presentada en conjunto de registros, esta tecnología propone utilizar mecanismos similares para acceder a este tipo de datos estén o no en bases de datos relaciónales.

UDA es una *filosofía* de actuación plasmada en la practica por *OLE DB (Object-Linking and Embedding DataBase)*. La idea es, extender el controlador ODBC a fuentes de datos no relacionales, y superar la naturaleza del API ODBC, escrito en C, y pensado para ser utilizado en C, proponiendo un estándar independiente del lenguaje de programación, siguiendo las directrices de la tecnología *COM* de *Microsoft*. OLE DB obtiene su funcionalidad de la existencia de un *servidor COM* que se comunica tanto con los clientes que quieren acceder a los datos, como de las propias fuentes en las que estos se almacenan. Lo que se denomina un *ODBC driver* pasa a denominarse un *OLE DB provider* de hecho, OLE DB se fundamenta en una abstracción que considera diferenciadas en el acceso a datos: los consumidores de datos *(Data Consumers)* y los proveedores de datos *(Data Providers)* como se observa en la Figura 13.

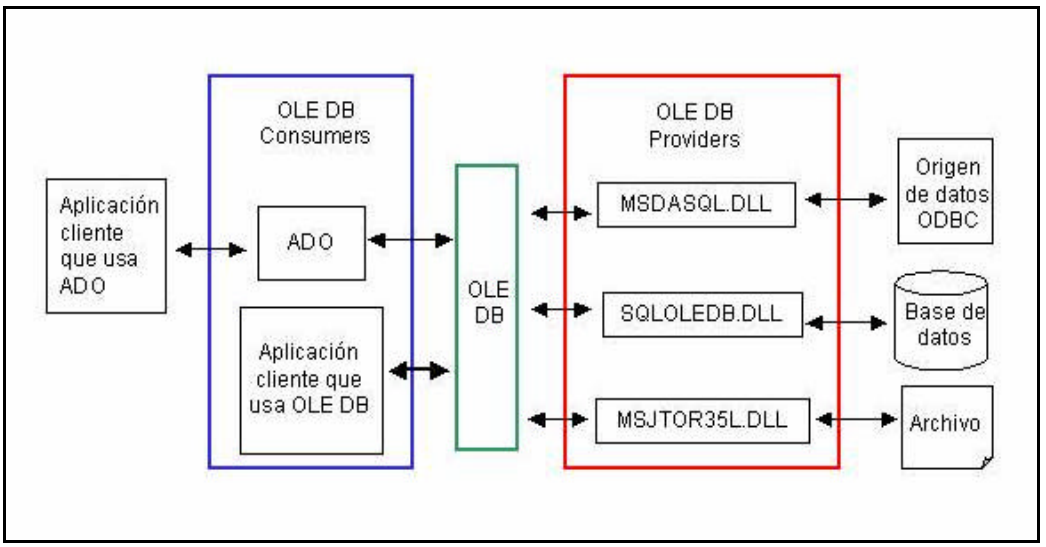

Figura 13. Interacción entre Consumidores y Proveedores

Un *OLE DB Consumer* es una aplicación que usa o consume interfaces OLE DB. Por ejemplo, una aplicación escrita en C++ y que usa OLE DB para conectarse a un servidor de base de datos puede ser considerado un *OLE DB Consumer.*

Los *OLE DB Provider* (proveedor de OLE DB) son las *DLL* que implementan las interfases OLE BD y que realizan y permiten el acceso nativo a los datos, como una base de datos *Informix u Oracle*. Son similares en funcionamiento a los controladores ODBC, excepto que los *OLE DB Provider* implementan interfaces con la tecnología *COM* en lugar de funciones de un API. Al utilizar un proveedor de OLE DB, la aplicación puede recuperar y manipular datos de una gran variedad de fuentes de datos, no sólo bases de datos relaciónales. Al crear un vínculo de datos, se especifica el proveedor de OLE DB designado para operar con los datos.

Por ejemplo, es posible tener acceso a un sistema de archivos planos o de correo electrónico mediante el proveedor de OLE DB adecuado y mostrar los datos en una aplicación cliente. Existe un puente entre OLE DB y ODBC por lo que para efectos prácticos se puede considerar OLE DB como un *súper conjunto* de ODBC.

4.3 MODELOS DE PROGRAMACIÓN PARA ACCESO A DATOS

Las tecnologías anteriormente señaladas son utilizables directamente a la hora de desarrollar aplicaciones; pero su naturaleza las hace poco adecuadas para lenguajes orientados a objetos, ya que estas (ODBC, OLE DB) son API de una u otra naturaleza.

Por este motivo se han desarrollado unos modelos de programación que se apoyan en estas u otras tecnologías exponiendo y definiendo una serie de jerarquía de objetos que pueden ser utilizados por los desarrolladores, de una manera sencilla y natural. (Vease el anexo B). Ejemplos de estos tipos de modelos son: DAO, RDO, ADO y JDBC

4.3.1 Data Access Objects (DAO). DAO, Objetos de Acceso a Datos, es un modelo de programación orientado a objetos que permite utilizar un lenguaje de programación para tener acceso y manipular datos en bases de datos locales o remotas y administrar bases de datos, los objetos y la estructura de estas. DAO es un conjunto de interfaces *(COM)* de automatizacion para el motor de base de datos *Acces/Jet*. *Jet* (Tecnología de motores unidos o *Joint Engine Technology*)

es el que proporciona el impulso a aplicaciones como *Access*, para permitir el acceso a diversos formatos de fuentes de datos tales como archivos de texto, archivos de aplicaciones de MS Office, archivos de Dbase, Paradox, etc. DAO tambien puede comunicarse con cualquier otra fuente de datos, pero a traves del motor *Jet.* DAO es facil de usar, como ODBC pero no ofrece el control de bajo nivel de este, por lo que se puede considerar una interfaz de alto nivel

4.3.2 Remote Data Objects (RDO). RDO, Objetos de Datos Remotos, Implementa un conjunto de objetos para conseguir requerimientos especiales de acceso a bases de datos. RDO se constituye como una delgada capa sobre el API ODBC, a la que accede directamente, y facilita el establecimiento de conexiones, la creación de resultados *(ResultSets)* y la ejecución de procedimientos complejos. RDO es fácil de usar como el API ODBC pero no ofrece el manejo de bajo nivel que este provee, por lo que RDO puede ser considerado como una interfaz de base de datos de alto nivel.

Debido a que RDO accede directamente al API ODBC, este puede proveer un buen rendimiento para las aplicaciones que accedan a los servidores de base de datos relaciónales. Otras funcionalidades en el uso de RDO son la flexibilidad en el uso de cursores, la ejecución de procedimientos almacenados *(stored procedures)* que opcionalmente devuelvan parámetros de salida o valores de retorno. También es posible delimitar el numero de filas devueltas y monitorizar los mensajes de error generados por la fuente de datos sin comprometer la consulta actual.

Permite además la ejecución asíncrona con lo que las aplicaciones no han de permanecer bloqueadas mientras esperan respuesta del servidor. A pesar de ser una interfaz que pretende realizar un acceso consistente a todas las bases de datos ODBC, su comportamiento depende sustancialmente de las características del controlador ODBC del fabricante. A su vez, no pueden realizarse tareas de administración de la base de datos mediante RDO, pues este modelo de objetos solo esta pensado para facilitar el acceso a datos.

Su funcionamiento óptimo se logra en redes LAN y algunos accesos transaccionales mediante Internet y CGI. No obstante, sus propias características (y las limitaciones de ODBC) no lo convierten en la mejor opción para aplicaciones Web *(Intranet, Extranet* e *Internet*), pues las conexiones deben ser permanentes.

4.3.3 ActiveX Data Objects (ADO). ADO es una tecnología de *Microsoft* para sustituir y ampliar el modelo de ínter conectividad que hasta ahora se había venido utilizando en sus plataformas de desarrollo. Se ha realizado este modelo de acceso a bases de datos orientado a objetos por encima de OLE DB, con el objetivo de implementar UDA. Sus ventajas principales son su facilidad de uso, su gran velocidad, el bajo consumo de memoria y la pequeña ocupación en disco. ADO es compatible con características claves en la creación de aplicaciones cliente - servidor y basadas en Web. ADO incluye, además, *Remote Data Service (RDS)*, con el cual se pueden mover datos de un servidor a una aplicación cliente o a una página Web, manipular los datos en el cliente y devolver las actualizaciones al servidor en un solo viaje de ida y vuelta.

RDS Se ha combinado con el modelo de programación ADO para simplificar el trabajo con datos remotos desde el lado del cliente.

4.3.3.1 Modelo de programación básica ADO. ADO proporciona los medios para realizar la siguiente secuencia de acciones:

- Conectarse a un origen de datos. Opcionalmente, puede asegurarse de que todos los cambios al origen de datos se realicen correctamente o no.
- Especificar un comando para tener acceso al origen de datos, opcionalmente con parámetros variables, o de forma opcional, optimizado para la ejecución.
- Ejecutar el comando.
- Si el comando hace que los datos se devuelvan en la forma de filas en una tabla, almacenar las filas en una caché que se pueda examinar, manipular o cambiar con facilidad.
- Si fuera apropiado, actualizar el origen de datos con los cambios de las filas de la caché.
- Proporcionar un medio general para detectar los errores (normalmente como resultado de realizar una conexión o ejecutar un comando).

Normalmente se emplearán estos pasos en el modelo de programación. Sin embargo, ADO es lo suficientemente flexible para que se pueda realizar un trabajo útil ejecutando simplemente una parte del modelo.

Por ejemplo, se podrían almacenar los datos a partir de un archivo directamente en un caché de filas y después utilizar los recursos de ADO simplemente para examinar los datos.

4.3.3.2 Arquitectura de ADO. ADO es un entorno orientado a objetos que facilita el acceso a bases de datos a las que pueda accederse bien desde OLE DB o ODBC. ADO se constituye como una capa entre la base de datos y la aplicación cliente. El lenguaje nativo que utilizan los objetos ADO es OLE DB, es decir ADO se comunica directamente con los OLE DB *providers.* Cuando la base de datos deba ser accedida mediante ODBC, las peticiones de datos serán traducidas por una DLL. (Véase la Figura 14).

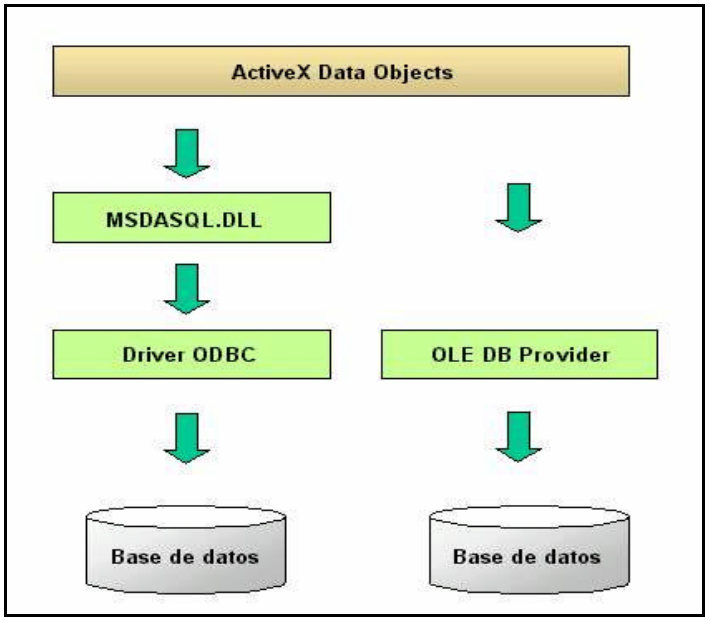

Figura 14. Arquitectura de ADO

4.3.3.3 Características. Algunas características más importantes de ADO son:

- ADO simplifica OLE DB: OLE DB es largo y complejo; un programa que use OLE DB debe usar complejas interfaces de la tecnología *COM.* ADO es más simple de usar y puede ser clasificado como una interfaz de alto nivel.
- Sencillez: ADO presenta una jerarquía con un pequeño numero de objetos, lo que hace esta tecnología fácil de aprender y de utilizar.
- Buen rendimiento: la necesidad de crear pocos ejemplares de objeto para acceder a los datos provoca una excelente funcionalidad en términos de potencia.
- Gestión completa de eventos: ADO proporciona un conjunto completo de eventos para los desarrolladores.
- Potencia: soporta cualquier operación de datos, incluyendo cursores complejos, ejecución de complejos almacenados e incluso actualizaciones por bloques mediante el uso de cursores en el cliente.

4.3.4 JDBC. JDBC es el estándar de Java para conectarse con bases de datos. JDBC está diseñado para ser independiente de la plataforma e incluso de la base de datos sobre la que se desee actuar. Para conseguir esta independencia, JDBC ofrece un sistema estándar de interconexión con las bases de datos, muy similar al SQL. JDBC es un API que consiste en una serie de clases e interfaces escritas en el lenguaje Java, que proporcionan un mecanismo para escribir aplicaciones en Java que accedan a bases de datos. (Véase la Figura 15).

Al igual que ODBC, JDBC permite crear aplicaciones de bases de datos con una gran independencia del gestor de base de datos al que se accederá. Al crear la aplicación se utilizan las clases y el API JDBC y será esta interfaz la que se encargara de enviar los comandos adecuados al gestor. JDBC es una interfaz de acceso a datos de bajo nivel, esto quiere decir, que se utiliza para ejecutar comandos SQL de manera directa. En cualquier caso se pueden crear interfaces de mas alto nivel que se apoyen en JDBC, y que resulten mas agradable para su utilización en entornos de desarrollo visuales, es mas, se esta utilizando SQL inmerso en Java *(Embebed SQL),* un API de alto nivel que se apoya en JDBC, y que se expresa en un conjunto de clases para la representación de bases de datos relacionales: en esta visión cada tabla es una clase y cada fila de la tabla es un ejemplar de la clase.

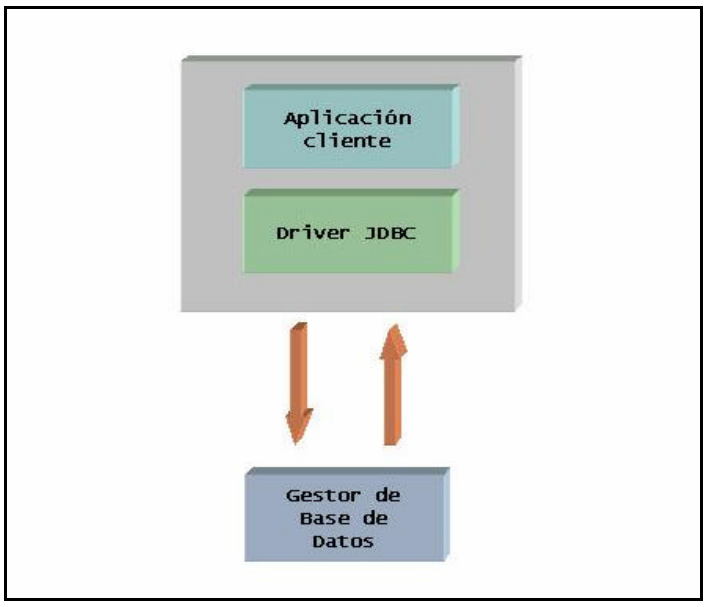

Figura 15. Arquitectura de la interfaz JDBC

4.3.4.1 JDBC vs. ODBC. Microsoft ODBC es el API mas utilizado para acceder a las bases de datos relacionales, pues su uso es realmente sencillo y permite la conexión virtualmente de todas las plataformas. Entonces ¿Por que utilizar JDBC?.

Existen numerosas razones por las cuales adoptar esta tecnología. Entre las cuales se incluyen las siguientes:

- ODBC no es apropiada para el uso directo con Java porque esta diseñada en C, esto motiva a que la aplicación Java pierda algunas de sus características de seguridad y portabilidad. Java no maneja punteros, por lo que la traducción del API ODBC a Java no es sencilla, de hecho, puede pensarse en JDBC como una traducción de ODBC de un modo orientado a objetos que es apropiado para la programación en Java.
- ODBC no es precisamente simple, en este API coexisten aspectos sencillos con otros muy complejos. JDBC ha sido diseñado para parecer sencillo, aunque pueden complicarse las cosas tanto como se desee.
- JDBC es necesario para una solución totalmente Java, ya que ODBC requiere de la instalación del manejador de controladores y los controladores en la maquina cliente, complicando la portabilidad.

4.3.4.2 Clasificación de los controladores JDBC. los controladores JDBC se pueden clasificar en cuatro niveles:

Puente JDBC-ODBC (Nivel 1): permite utilizar controladores ODBC como controladores JDBC. Esta solución traduce las llamadas a los métodos JDBC, a llamadas ODBC, y requiere que existan los controladores ODBC necesarios en la maquina, con lo que la portabilidad se resiente. Sin embargo facilita la utilización de JDBC para gestores de bases de datos que aun no proporcionan controladores JDBC puros; pero que si facilitan controladores ODBC. (Véase la Figura 16).

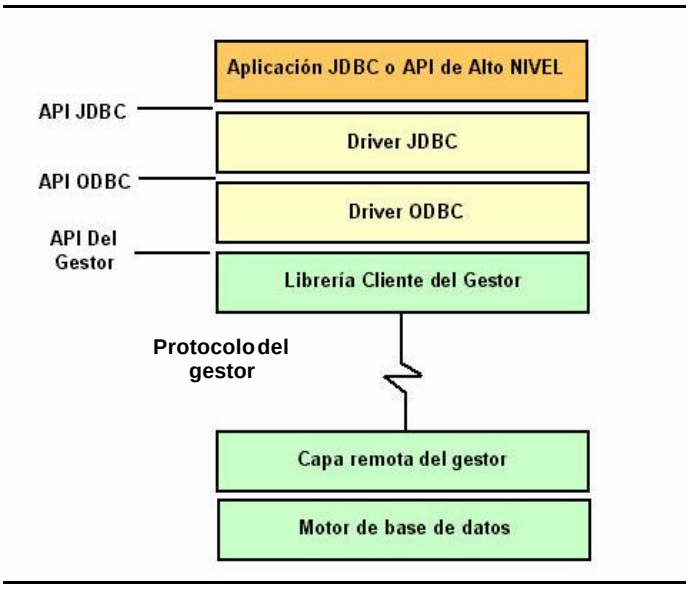

Figura 16. Puente JDBC-ODBC

Controladores JDBC de acceso al API del gestor de datos (Nivel 2): estos controladores habitualmente llamados *Native API partly java,* traducen las llamadas Java al API del gestor, con lo que se elimina la capa ODBC, con la consiguiente mejora de rendimiento; pero no se elimina la necesidad de código binario dependiente de la plataforma en el cliente. (Véase la Figura 17).

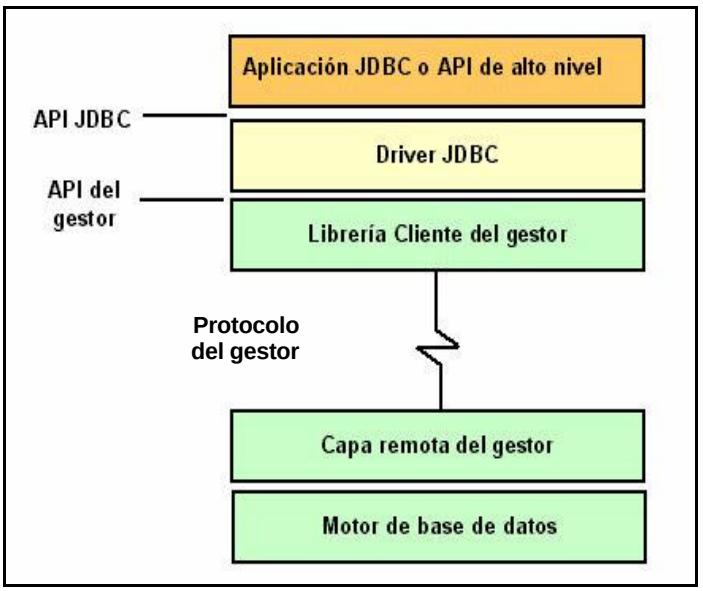

Figura 17. Controladores JDBC de acceso al API del gestor

Controladores JDBC de red (Nivel 3): este el primero de dos soluciones distribuidas, en este caso el controlador esta distribuido entre los componentes del cliente y el servidor. En este nivel las llamadas son dirigidas por el controlador a través de un protocolo de red al gestor de base de datos. Esto es conveniente cuando el protocolo de red y el del gestor no coinciden. Este tipo de controlador no requieren de ningún tipo de código binario en el cliente. (Véase la Figura 18).

Controlador JDBC nativo (Nivel 4): este tipo de controlador conforma una solución completamente java, que traduce las llamadas directamente al protocolo de red utilizado por el gestor. (Véase la Figura 19).

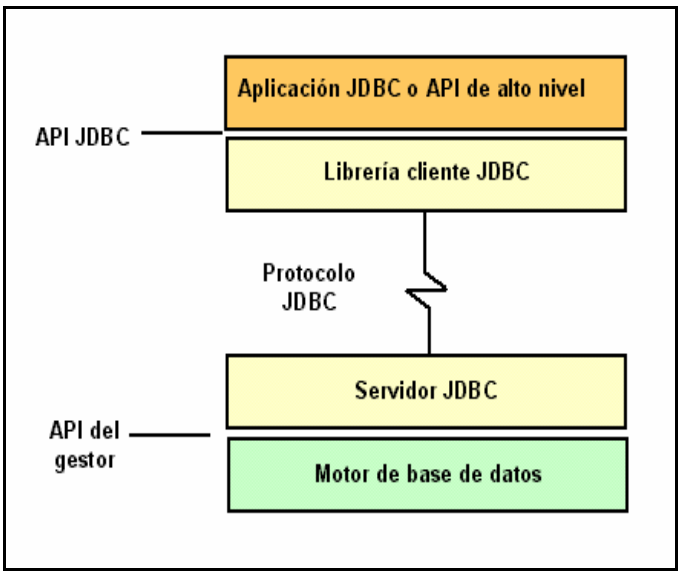

Figura 18. Controladores JDBC de red

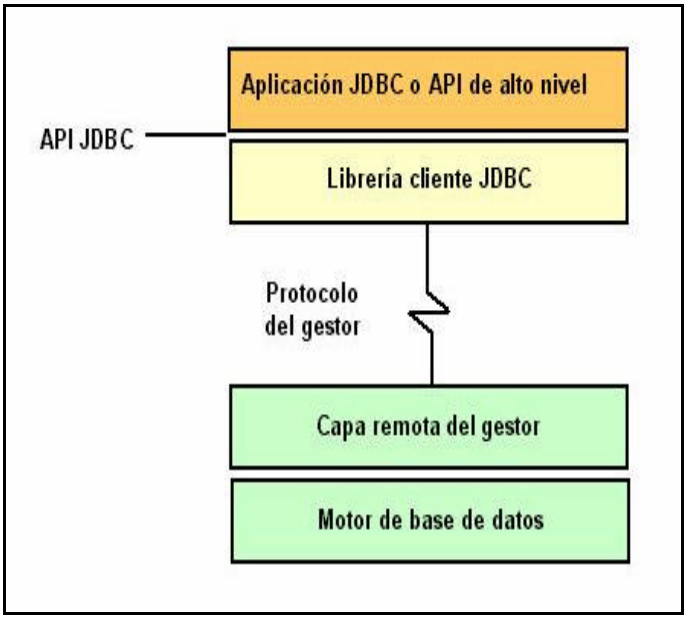

Figura 19. Controlador JDBC nativo

5. METODOLOGÍAS PARA EL ACCESO A LAS BASES DE DATOS CENTRALIZADAS A TRAVÉS DEL WEB

Una de las necesidades que la sociedad actual solicita es la de enviar y recibir informacion al instante. Esta sociedad produce información de manera no cuantificable, acerca de temas que van desde lo especializado a lo informal, de lo cultural a lo divertido, de noticias trascendentes en el campo del periodismo hasta la farándula, en fin cualquier tema imaginable.

Servicios vía Internet, teléfonos, celulares, buscapersonas y muchos más son sólo un ejemplo de situaciones que hacen necesario el acceso a la información de forma precisa y rápida. Uno de los servicios que más ha impulsado estas situaciones es Internet, puesto que cada vez son más las personas que requieren acceder a sus cuentas bancarias, registros personales e información corporativa al instante, implicando innumerables situaciones, donde la sincronización de datos en diversas fuentes requieren de procesos automáticos para minimizar el tiempo de administración y la posibilidad de errores.

El mecanismo por excelencia para realizar esto es la implementación de bases de datos.

Las bases de datos han estado presentes en la vida del hombre a lo largo de toda su evolución, se han encontrado vestigios de registros con escritura cuneiforme, hechos sobre las paredes de las cavernas, en eras posteriores se registraba la información correspondiente a los resultados de las faenas relacionadas con el agro, el comercio, los inventarios en tablas de arcilla.

Como se puede observar la información siempre se registraba, pero los métodos para el almacenamiento y recuperación de dicha informacion no eran los mas indicados por obvias razones: las tablillas tienden a romperse, las cavernas podían tener restricciones de espacio o condiciones ambientales.

Luego de las tablas de arcilla se paso al registro hecho al papel recluido en archivadores pero este modelo también tenia sus limitaciones, como la escritura deficiente en los datos, las faltas ortográficas, datos erróneos, etc. Con el paso de los años surgieron los sistemas informáticos los cuales permitieron manejar grandes volúmenes de datos de una manera rápida, eficiente y segura.

5.1 FACTORES QUE INFLUYEN EN EL USO DE LAS BASES DE DATOS

La información aparte de ser registrada en forma técnica y adecuada debe ser localizada, editada y manipulada con métodos adecuados, los gestores de bases de datos modernos permiten hacer esto, a su vez existen factores que impulsan el uso de tales sistemas, cuya combinación hace más fácil el almacenamiento de la información, más económica y disponible a más personas o entidades.

A continuación se relaciona y describen estos factores:

- **Bajo costo en el almacenamiento de la información:** los avances recientes en las tecnologías de almacenamiento de datos, han favorecido a los consumidores, de medios o dispositivos para el almacenamiento de datos, debido al abaratamiento de estos.
- **Avances en la tecnología de computo:** la potencia de los computadores día tras día tiende al alza. La capacidad de computo de los procesadores es alta y los tiempos de respuesta son muy bajos, estos factores favorecen la manipulación de los datos. Otro campo donde se ha avanzado es en la tecnología de memoria intermedia (RAM). Los servidores bases de datos suelen absorber grandes cantidades de esta para su funcionamiento continuo. Su costo disminuye progresivamente, lo que hace que los servidores de bases de datos sean preparados con grandes cantidades de memoria, para un desempeño optimo.
- **Acceso a la red Internet:** esto provee a las empresas, y usuarios o clientes un escenario ideal para llevar a cabo un sinnúmero de actividades que les reportaran beneficios ilimitados.
- **Avances en la tecnología de bases de datos:** los avances en la arquitectura de modelación de bases de datos han obtenido grandes logros, el modelo relacional es el mas usado debido a que se adaptan a los requerimientos de manejo de información de las empresas. Dentro de los avances en las tecnologías de bases de datos se tienen:

- **Los optimizadores de consultas:** son programas que analizan el enunciado de una consulta y deciden cual es la forma más eficiente de reunir los datos solicitados.
- **Interfaz de base de datos:** las bases de datos modernas vienen provistas con una Interfaces para Programación de Aplicaciones (API), que proporciona a los programadores llamadas a funciones para comunicarse con la base de datos.
- **Almacenamiento de datos:** los administradores de bases de datos en la actualidad están capacitados en el manejo del lenguaje SQL, los datos que se guardan en una base de datos, deben cumplir ciertas especificaciones formales, de lo contrario se incurre en contratiempos como: redundancia, inconsistencia, corrupción, entre otros.

Adicionalmente los gestores de bases de datos modernos poseen características que facilitan la integración de componentes y el máximo provecho de la plataforma computacional tales como:

- *Capacidad de multihilo* para realizar las tareas más rápidamente.
- División y balanceo de tareas o *multiprocesamiento simétrico (SMP).*
- Pueden utilizarse diversos protocolos de red: *TCP / IP, IPX/ SPX*, etc.
- Escalabilidad: el gestor puede adaptarse al tamaño del negocio.
- Replicación de datos y transacciones distribuidas: permite la distribución de los datos diseminados a lo largo de las corporaciones, para reducir riesgos de

perdidas de datos y mejorar el rendimiento para acercar la informacion al usuario.

- Gestión centralizada de servidores: esto permite tener varios servidores en la empresa gestionados de manera centralizada mediante un marco de trabajo distribuido.
- Mecanismos de seguridad avanzados de gestión de la seguridad: permite la protección de los datos como de la realización de copias de seguridad.

5.2 POR QUÉ USAR EL WEB PARA ACCESAR BASES DE DATOS

Existen fuertes razones que inciden en el uso de las tecnologías del Web para accesar la información que manejan los gestores de bases de datos, el principal de todos es el crecimiento escalar de las entidades o corporaciones, este crecimiento se puede dar como una expansión geográfica de la empresa (nuevas sucursales) ó un aumento en el numero de usuarios que interactúan con las bases de datos, ambas situaciones implica mas personal y por supuesto mas hardware y software, luego el volumen de información es proporcional al tamaño de la entidad, lógicamente se analizara mas información y se procesaran mayores cantidades de datos vitales para su normal funcionamiento. Esto implica un problema que la empresa debe solucionar. Las soluciones podrían incluir:

- Adquisición de nuevos equipos de diversas especificaciones y de que sean de determinada marca pues así lo determinan las necesidades.

- Para el funcionamiento de estos nuevos equipos tal vez se haga necesariamente adquirir diversos sistemas operativos.
- Desarrollar nuevas aplicaciones, clientes pesados *(front ends)* hechos a la medida en novedosos lenguajes visuales, que permitan acceder al gestor y que faciliten el intercambio de datos con este.
- Ampliación y/o modernización de la red interna y todo lo que ello signifique.
- La adquisición de un gestor de base de datos y software propietario.

Todas estas soluciones son validas pero acarrean costos y es mas, esto se podría repetir varias veces con el transcurrir de los años según se vaya presentando la expansión de la empresa. Pero lo que el negocio tal vez necesite sea la estandarización de la interfaz de usuario y los lenguajes, que funcione en cualquier maquina sin importar su marca y plataforma ó hacia donde se transfieren los datos, como tampoco que maquina trata de acceder la base de datos.

La solución podría estar en las tecnologías de servicios que ofrece el ***World Wide Web***, por las siguientes cuatro razones:

- **Uso de lenguajes multiplataforma:** con estos lenguajes es posible escribir aplicaciones que sean accesibles desde cualquier tipo de computador, los programadores tienen que aprender un único lenguaje, y los componentes necesarios para desarrollo están disponibles sin costo alguno, la administración y el mantenimiento de estas aplicaciones se realizan de una manera rápida.

Tal vez no se requieran costosos compiladores, pueden ser interpretados, pueden crearse o leerse con cualquier editor de texto.

- **Utilización de una interfaz grafica única de usuario:** las aplicaciones HTML, XML o basadas en lenguajes para programación de *scripts client-side,* son formateadas por una interfaz grafica cliente llamado navegador Web, este formatea el código y lo convierte en una aplicación hipermedia que se puede manejar con el ratón. La información puede ser digitada en campos, seleccionada de listas o al oprimir un botón. Al utilizar un navegador se evita tener una interfaz hecha a la medida que frecuentemente necesita mantenimiento. En vez de invertir tiempo y dinero en el desarrollo de una nueva interfaz, mejor se dedica tiempo en el desarrollo de las aplicaciones de servidor que se encargaran de la lógica comercial y de producir el código que el navegador formateara para la interacción con el usuario final.
- **Soporte a plataformas cruzadas:** la característica principal de los navegadores Web es que están disponible para la muchos tipos de sistemas operativos, esto hace posible tener un servidor de bases de datos en una maquina *Sun* y utilizar un *Palmtop 3Com* o un *PC 586,* para accesar la base de datos. No es necesario sustituir los sistemas y las maquinas que se tienen, lo cual ahorra tiempo, dinero y dificultades. Para disfrutar este soporte de plataformas cruzadas no es necesario generar aplicaciones para varias maquinas como sucedería si se tuviera un *front end* diseñado a la medida.
- **Posee soporte de red:** los navegadores y servidores Web tienen la capacidad de conectarse con redes interconstruidas, ya que están diseñados para enviar

y recibir información a través de Internet o una intranet, de un tipo de maquina u otro, eliminando la necesidad de utilizar programas propietarios.

5.3 TECNOLOGÍAS

Las técnicas de servidor que permiten el acceso a la información a las bases de datos utilizando las tecnologías Web, son numerosas y de diversas especificaciones. Muy pocas poseen una arquitectura homogénea, la gran mayoría consisten en una amalgama de diferentes tecnologías, compuestas por: plataformas computacionales, entornos de desarrollos, lenguajes de programación, API clientes de gestor, especificaciones y tecnologías de servidores Web; etc. Todas aplican de una manera general la misma filosofía para llevar acabo su función:

- Conectarse a la base de datos.
- Enviar la consulta.
- Captar el conjunto de resultados devuelto.
- Procesar la información según la lógica programada.
- Devolver la respuesta a la aplicación de nivel superior para su posterior envió al usuario.
- Cerrar la conexión.

El cómo se llevan a cabo estas operaciones y aspectos tales como seguridad, velocidad de procesamiento, velocidad de respuesta; etc., dependerán en sí de la tecnología escogida. No existe una verdad absoluta, todo depende de las necesidades y objetivos del proyecto y de las capacidades que se tengan para realizarlo. Sin embargo los elementos básicos que intervienen en la implementación de estos sistemas son:

- **La plataforma computacional:** proveerá los servicios de red y de servidor, dará soporte a las aplicaciones, los lenguajes, los gestores de bases de datos y las interfaces clientes de bases de datos.
- **El gestor de bases de datos:** el elemento central. Proveerá la informacion a modificar y la interfaz nativa.
- **El lenguaje de programación:** permitirá crear el código de las aplicaciones de la capa media lo que se traduce en aplicaciones de servidor, estas aplicaciones harán básicamente tres cosas:
  - Contener la lógica del negocio (lógica de aplicación).
  - Interactuar con la interfaz cliente de base de datos; a través de este enviaran las consultas y recibirán los conjuntos de resultados.
  - Traducir las respuestas en el lenguaje que sea soportado por el cliente (HTML, XML, WML; etc.).
- **La interfaz cliente del gestor:** que comunicara a la aplicación con la interfaz nativa del gestor.
- **El cliente:** que permitirá la interacción entre el usuario y el gestor.

5.3.1 Evaluación de las metodologías. Teniendo en cuenta todo lo anterior, para llevar acabo la evaluación de las metodologías, se escogieron técnicas representativas que permiten tal fin en la actualidad. Estas metodologías son:

- La aplicación de servidor ODBiC.
- La interfaz PHP para acceder bases de datos.
- La tecnología ASP-ADO.
- La tecnología Servlets de Java.
- La interfaz DBI de Perl.
- La técnica HTX-IDC.
- La interfaz ColdFusion.

Para la evaluación de estas tecnologías se utilizaron además utilidades y software adicional tales como: servidores Web, editores de documentos HTML entre otros. Las siguientes tablas contienen las especificaciones técnicas de los elementos de *software* utilizados durante esta etapa.

Tabla 10. Características técnicas de las plataformas utilizadas

Sistema	Plataforma
Servidor	• Microsoft Windows NT 4.0 Server - SP4 • Linux SuSE 7.0
Cliente(servidor de pruebas)	• Microsoft Windows 98 SE

Tabla 11. Software empleado

Plataforma	Servidor Web	Gestor(es) de Base datos	Lenguajes
Linux SuSE 7.0	Apache 1.3.12, Tomcat 3.1	MySQL 3.22.32, PostgreSQL 7.0, Oracle 8.0	Perl 5.005, PHP 3 Java 1.2
WindowsNT 4.0	IIS 4.0	MS Access 2000, MS SQL Server 7.0, InterBase6 Sybase ASE 12.0	PHP 4, Perl 5.6, Java 1.22, ASP, VBScript
Windows 98	PWS 4.0, JRUN 3.0	Microsoft Access 2000	

Las tecnologías con interfaz centradas en el servidor, evaluadas en Linux y Microsoft Windows están descritas en las siguientes tablas.

Tabla 12. Tecnologías evaluadas en Linux

Interfaz	Gestor de base de datos
DBI:DBD	MySQL versión 3.22.32
PHP (funciones de acceso a PostgreSQL)	PostgreSQL versión 7.0

Tabla 13. Tecnologías evaluadas en plataformas Microsoft Windows

Interfaz	Gestor de base de datos
ASP-ADO	Fuente de datos ODBC (MS SQL Server versión 7)
ODBiC 1.6	Fuente de datos ODBC (InterBase versión 6)
HTX-IDC	MS SQL Server versión 7
ColdFusion	Access 2000
Servlets-JDBC	InterBase versión 6

Para realizar la evaluación de las diferentes técnicas de acceso a bases de datos relacionales a través del Web, se llevaron a cabo varios pasos:

- Investigación y estudio previo de la tecnología.
- Aprendizaje y asimilación del lenguaje y componentes a emplear.
- Instalación del gestor de bases de datos.
- Estudio de las características del gestor, creación de la base de datos de prueba (tablas).
- Instalación de las utilidades o herramientas a utilizar en la metodología.
- Desarrollo e implementación de las aplicaciones de manipulación de datos. Estas aplicaciones realizaran las operaciones de:
    - Inserción
    - Actualización
    - Listado.
    - Eliminación de registros.
- Prueba y análisis de la técnica.
- Análisis de resultados.
- Documentación de la metodología.

6. LA INTERFAZ DBI DE PERL

DBI (Data Base Interface), Interfaz de Base de Datos, es un API de acceso a bases de datos para el lenguaje Perl. Las especificaciones del API DBI definen un conjunto de funciones, variables y convenciones que ofrecen una interfaz de base de datos consistente e independiente del gestor de base de datos que sé este utilizando.

6.1 MODULO DBI

DBI es un módulo de Perl para acceso a bases de datos, mediante DBI se puede acceder a bases de datos con *scripts* escritos en Perl. Lo cual no significa que no haya otros; pero, normalmente todo lo que se puede hacer con otro módulo (que no use DBI para acceder a bases de datos), se puede hacer con DBI, de forma más fácil y portable. DBI es independiente de la base de datos con la que se está trabajando, esto significa que se pude trabajar con bases de datos como *Oracle, Sybase, Informix, MySQL, mSQL,* bases de datos con soporte ODBC, ADO *(MS-Access, SQL Server)*; etc. En la actualidad DBI sólo trabaja con bases de datos relacionales y no con bases de datos orientadas a objetos.

6.2 MODULO DBD

DBD (Data Base Driver), es el que se encarga de traducir lo que se pide que se haga en el *script* Perl, usando DBI, en una base de datos específica. Existe un módulo DBD para cada motor de base de datos, dicho módulo se encarga de pasar las peticiones que se realizan al DBI a peticiones a la base de datos que se esta accediendo. Para trabajar con una base de datos determinada hace falta tener el controlador o módulo DBD correspondiente, por ejemplo, para trabajar con la base de datos *Oracle* se necesita el módulo DBD::oracle.

La siguiente es una lista de los módulos para motores de bases de datos existentes y sus versiones hasta la edición de este documento:

Tabla 14. Módulos de interfaz de DBD

Modulo	Versión actual del modulo
DBD::ADO	2.1
DBD:: ASAny	1.12
DBD::Adabas	0.2003
DBD::CSV	0.1027
DBD::Chart	0.41
DBD::DB2	0.75
DBD::DBMaker	0.13
DBD::DtfSQLmac	0.1201

DBD::Empress	0.52
DBD::EmpressNet	0.52
DBD::Excel	0.03
DBD::FreeTDS	0.02
DBD::Fulcrum	0.20
DBD::Illustra	0.04
DBD::Informix	0.97005
DBD::Informix	1.00.PC
DBD::Informix4	0.23
DBD::Ingres	0.30
DBD::InterBase	0.27
DBD::JDBC	0.64
DBD::NET	0.1
DBD::ODBC	0.28
DBD::Oracle	1.07
DBD::Ovrimos	0.12
DBD::Pg	1.00
DBD::QBase	0.03
DBD::RAM	0.072
DBD::SQLrelay	0.1
DBD::SearchServer	0.2
DBD::Solid	0.13
DBD::Sprite	0.19

DBD::Sybase	0.91
DBD::Teradata	1.12
DBD::Unify	0.20
DBD::XBase	0.173
DBD::mysql	2.0901
DBD::pNET	0.1003
DBD::SQLFLEX	8.2

Anteriormente, en Perl 4, las especificaciones de la interfaz se conocían como *DBPerl.* Algunos de estos módulos son *oraperl, isqlperl, ingperl*. Estas API no cuentan con una interfaz estándar y tampoco con mucho soporte. La Tabla 15. contiene una lista de algunos módulos DBperl y su contraparte DBI.

Para trabajar con bases de datos en Perl es necesario haber instalado Perl, la base de datos con la que se trabajara, el módulo DBI y el módulo DBD para la base de datos instalada. Para programar con Perl y acceder a la base de datos se necesita saber programar en Perl, saber usar el módulo DBI y saber o conocer SQL, porque DBI se comunica con la base de datos a través de SQL. Es de tener en cuenta que SQL es un lenguaje estándar en el acceso a bases de datos; pero cada base de datos utiliza su propio dialecto en SQL, se puede dar el caso, que el *script* en Perl, usando DBI, no sea portable por que las sentencias SQL utilizadas utilicen aspectos no estándar o aspectos que no estén soportados por la base de

datos con la que se vaya a trabajar.

Tabla 15. **Módulos de interfaz DBperl vs. DBD**

Nombre del módulo Dbperl	Gestor de base de datos requerido	Driver DBD
Sybperl	Sybase	DBD::sybase
Oraperl	Oracle 6,7 y 8	DBD::oracle
Ingperl	Ingres	DBD::ingres
interperl	Interbase	DBD::interbase
Uniperl	Unify 5.0	DBD::Unify
pgperl	Postgres	DBD::Pg
btreeperl	NDBM	SDBM
ctreeperl	C-Tree	Ninguno
cisamperl	Informix C-ISAM	Ninguno
duaperl	X.500 Directory	Ninguno

6.3 ARQUITECTURA DE DBI/DBD

La arquitectura de DBI se puede dividir en dos partes. (Véase figura 20), la propia interfaz DBI y los *drivers* (controladores) del gestor de base de datos, que se encargan de acceder a la base de datos correspondiente. Los controladores se implementan en el módulo DBD.

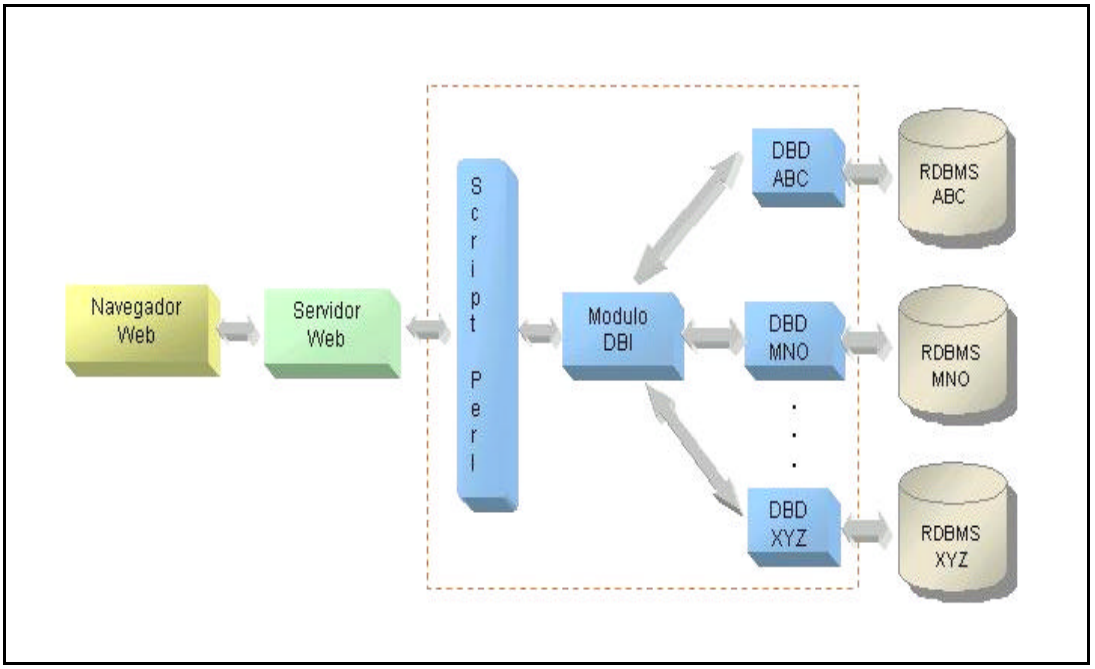

Figura 20. Arquitectura del modulo DBI

6.3.1 Manejadores (Handles). DBI define tres tipos principales de objetos para interactuar con las bases de datos, dichos objetos son denominados manejadores, existen manejadores para controladores, otros para conexiones con la base de datos y los que se crean para realizar sentencias individuales en la base de datos.

6.3.1.1 Manejadores de controladores. Estos manejadores representan los *drivers* que están cargados en DBI y se crean e inicializan cuando los controladores son cargados en el DBI, solo debería haber uno por cada base de datos con la que se está trabajando, por convención se les suele denominar usando como nombre de variable ***$drh***.

6.3.1.2 Manejadores de bases de datos. Los manejadores de base de datos encapsulan una conexión con una base de datos en particular dentro de un sistema gestor de bases de datos, por convención se les suele denominar usando como nombre de variable ***$dbh****.*

6.3.1.3 Manejadores de sentencias. Los manejadores para las sentencias encapsulan sentencias SQL que se envían a la base de datos. Para cualquier base de datos no hay límite en los manejadores de sentencias que se puedan crear y ejecutar.

El número de manejadores que se pueden ejecutar a la vez (concurrentemente) depende del *driver* de la base de datos (módulo DBD), por ejemplo, en ODBC sólo se puede ejecutar una sentencia a la vez, mientras que en MySQL se pueden ejecutar todas las que se deseen, por convención se les suele denominar usando como nombre de variable ***$sth****.*

6.4 ACCEDIENDO AL GESTOR DE BASE DE DATOS: EL API DBI

Las siguientes tablas contienen los métodos, atributos y funciones de la clase DBI.

Tabla 16. Métodos de la clase DBI

Método	Ejemplo de uso
connect	`$dbh = DBI->connect("DBI:oracle:$database:$hostname:$port", $username, $password);`
available_drivers	`@drivers = DBI->available_drivers;`
data_sources	`@sources = DBI->data_sources($driver);`
trace	`DBI->trace($trace_level);`

Tabla 17. Funciones de utilidades de DBI

Funciones	Ejemplo de uso
neat	`$neat_value = DBI::neat($value, $maxlen);`
neat_list	`$neat_list = DBI::neat_list($listref, $maxlen, $field_sep);`
dump_results	`$rows = DBI::dump_results($sth, $maxlen, $lsep, $fsep, $fh);`

Tabla 18. Atributos dinámicos de DBI

Atributo	Ejemplo de uso
$DBI::err	`Mirar $dbh->err`
$DBI::errstr	`Mirar $dbh->errstr`
$DBI::state	`Mirar $dbh->state`
$DBI::rows	`Mirar $dbh->rows`

Tabla 19. Métodos comunes a todos los manejadores

Método	Ejemplo de uso
err	`$err_code = $dbh->err;`
errstr	`$err_string = $dbh->errstr;`
state	`$state = $dbh->state;`
trace	`$trace = $dbh->trace;`
func	`$result = $dbh->func(@func_args, $func_name);`

Tabla 20. **Atributos comunes a todos los manejadores**

Atributo	Ejemplo de uso
Warn	`$dbh->{Warn} = 1;`
CompatMode	`$dbh->{CompatMode} = 1;`
InactiveDestroy	`$dbh->{InactiveDestroy} = 1;`
PrintError	`$dbh->{PrintError} = 1;`
RaiseError	`$dbh->{RaiseError} = 1;`
ChopBlanks	`$dbh->{ChopBlanks} = 1;`
LongReadLen	`$dbh->{LongReadLen} = 1;`
LongTruncOk	`$dbh->{LongTruncOk} = 1;`

Tabla 21. Métodos de los manejadores de bases de datos

Método	Ejemplo de uso
prepare	`$sth = $dbh->prepare($statement);`
do	`$sth->$dbh->do($statement);`
commit	`$dbh->commit;`
rollback	`$dbh->rollback;`
disconnect	`$dbh->disconnect;`
ping	`$return = $dbh->ping;`
quote	`$sql = $dbh->quote($string);`

Tabla 22. Métodos de los manejadores de sentencias

Método	Ejemplo de uso
bind_param	`$sth->bind_param($param_num, $bind_value, $bind_type);`
bind_param_inout	`$rv = $sth->bind_param_inout($param_num, \$bind_value, $max_len);`
execute	`$sth->execute;`
fetchrow_arrayref	`$row_array = $sth->fetchrow_arrayref;`
fetchrow_array	`@row_array = $sth->fetchrow_array;`
fetchrow_hashref	`$row_hash = $sth->fetchrow_hashref;`
fetchall_arrayref	`$row_all = $sth->fetchall_arrayref;`
finish	`$sth->finish;`
rows	`$rc = $sth->rows;`
bind_col	`$sth->bind_col($column_number, \$bind_var);`
bind_columns	`$sth->bind_columns(\%attr, @bind_var_refs);`

Tabla 23. Atributos de los manejadores de sentencias

Atributo	Ejemplo de uso
NUM_OF_FIELDS	`$num_fields = $sth->{NUM_OF_FIELDS};`
NUM_OF_PARAMS	`$num_params = $sth->{NUM_OF_PARAMS};`
NAME	`$names = $sth->{NAME};`
NULLABLE	`$nullables = $sth->{NULLABLE};`
CursorName	`$cursor_name = $sth->{CursorName};`

Tabla 24. Depuración

Método	Ejemplo de uso
DBI_TRACE	`DBI_TRACE = 2 perl test_script.pl`

6.4.1 Métodos de la clase DBI.

- **connect:** Se usa el método *connect* para crear una conexión con la base de datos a la fuente de datos.

  - *datasource***:** nombre de la fuente de datos, esto es una cadena que contiene los siguientes elementos, separados por dos puntos:

    - DBI
    - Nombre del *driver*: Nombre del *driver*, por ejemplo, odbc, sybase, oracle; etc.

- Nombre de la base de datos: Nombre de la base de datos a la que se accederá.

Ejemplo: `'DBI:oracle:basededatos'`

- *username*: Usuario que se va a conectar a la base de datos. Dentro de la base de datos hay distintos usuarios, cada uno, con sus propias tablas y con una serie de privilegios asociados.
- *password*: Clave de seguridad del usuario. Si el *username* (nombre de usuario) y/o el *password* están indefinidos, entonces DBI usará los valores de la variables de entorno *DBI_USER*, *DBI_PASS* respectivamente.
- *hostname:* Aquí se indica la máquina donde está la base de datos, es opcional.
- *port:* Aquí se indica el puerto que se va a usar para conectarse a la base de datos, es opcional.

- **available_drivers:** este método devuelve un arreglo con los controladores disponibles, buscándolos en los directorios *DBD::**.

- **data_sources:** este método devuelve un arreglo de bases de datos disponibles para un *driver* dado. Si dicha variable *$driver* se omite, se utiliza la variable de entorno *DBI_DRIVER*.

- **trace:** la información en *DBI* puede seguirse (*trace*) para todos los manejadores que usen éste método. Para habilitarlo para un manejador específico hay que usar el método *$dbh->trace* similar para el manejador. La información que devuelve *trace* estará disponible para todos los manejadores que usen este método.

  Poniendo la variable *$trace_level* en dos, se pude ver información más detallada, si se pone en cero desaparece la información de seguimiento. Si el fichero *$trace_file* está especificado, entonces toda la información se añade a dicho fichero.

6.4.2 Funciones de utilidades de DBI.

- **neat:** devuelve una representación limpia de la variable *($value)* que se le pasa, es decir, formateada para manejo. Esta función se usa internamente por el *DBI* para el seguimiento *(trace)*. No debe ser usada para formatear valores para usarlos en la base de datos.

- **neat_list:** Llama a *DBI::neat* para cada elemento de *@listref* y devuelve una cadena con los resultados unidos por *$field_sep*. No debe ser usada para formatear valores a usar en la base de datos.

- **dump_results:** esta función extrae todas las filas de un manejador de sentencia *($sth)*, llama a la función *DBI::neat_list* para cada fila, y escribe el resultado en el descriptor de fichero *($fh),* que por defecto es *STDOUT*. El separador de línea *($lsep)* es por defecto \n', el separador de campos *($fsep)* es por defecto ',' y la máxima longitud *($maxlen)* es 35.

6.4.3 Atributos dinámicos de DBI. Todos los atributos dinámicos descritos a continuación son volátiles *(volatile)*, es decir, tienen una vida corta debido a que siempre están asociados al último manejador usado. El motivo es que tienen que ser usados inmediatamente después de llamar el método que los activa. También se han encontrado problemas con multi hebras en Perl 5.005. Es mejor usar los atributos equivalentes que se mencionan en la siguiente tabla:

Tabla 25. Equivalencia de atributos dinámicos

Atributo	Equivalente
`$DBI::err`	`$dbh->err`
`$DBI::errstr`	`$dbh->errstr`
`$DBI::state`	`$dbh->state`
`$DBI::rows`	`$dbh->rows`

6.4.4 Métodos comunes a todos los manejadores.

- **err:** este método devuelve el código de error nativo de la base de datos, provocado por la última función llamada del *driver*.

- **errstr:** este método devuelve el mensaje de error nativo de la base de datos, debido a la última función llamada del controlador.

- **state:** devuelve el estado del manejador especifico de la base de datos.

- **trace:** similar a *DBI::trace* con una excepción. El seguimiento sólo está asociado con el manejador específico, con el que está siendo usado.

- **func**: este método se usa para llamar a métodos privados que están implementados por el controlador (DBD::*).

El siguiente ejemplo que usa alguno de los métodos anteriormente descritos:

```
#!/usr/bin/perl -w
use DBI;
use strict;
my $trace_level = 2;
my $dbh = DBI->connect("DBI:mysql:contact",undef,undef)
$dbh->trace($trace_level);
```

```
or die "No se puede conectar a la BD: $dbh->errstr\n";
my $sth = $dbh->prepare("SELECT * FROM contact");
$sth->execute or die "Imposible ejecutar consulta:$dbh->err,$dbh->
errstr\n";
$sth->finish;
@tables = $dbh->func('_ListTables')
or die "No se puede listar las tablas: $dbh->errstr\n";
foreach $table(@tables) {
    print "$table\n";
}
$dbh->disconnect;
exit;
```

6.4.5 Atributos comunes a todos los manejadores. La mayoría de estos atributos son heredables, es decir, cualquier manejador hijo los heredará de sus padres. Por ejemplo: los manejadores de sentencias los heredarán de los manejadores de la base de datos que los crea.

```
$dbh->{nombre_atributo} = 1;
$value = $dbh->{nombre_atributo};
```

- **Warn:** Booleano, heredado. Activa mensajes de aviso útiles. Activado por defecto.

- **CompatMode:** Booleano, heredado. Usado para emulación de capas, para activar el modo compatible en el *driver*.

- **InactiveDestroy:** Booleano. Este atributo es usado para inhabilitar los efectos de destruir un manejador. Está específicamente diseñado para aplicaciones en Unix que tienen procesos hijos creados con *fork()*. EL padre o el hijo deben poner este atributo en todos sus manejadores; pero no los dos.

- **PrintError:** Booleano, heredado. Este atributo se usa para forzar a los errores a generar mensajes de aviso además de los códigos de error, por defecto *DBI -> connect* tiene activado *PrintError*.

- **RaiseError:** Booleano, heredado. Este atributo es usado para obligar a los errores a crear excepciones justo antes de devolver un código de error. Por defecto, no está activado.

- **ChopBlanks:** Booleano, heredado. Este atributo se usa para controlar los espacios en los campos de anchura fijos. Por defecto es falso.

- **LongReadLen:** Entero, heredado. Este atributo se usa para controlar la máxima longitud de los campos *blob* (objetos binarios largos). Un valor de nueve significa que automáticamente no se extraen datos largos (la extracción debe devolver *undef* para el *blob* cuando está a cero). Por defecto es,

normalmente, cero; pero puede haber diferencias entre *drivers*. Este atributo debe ser activado antes que la sentencia esté en la etapa de preparación.

- **LongTruncOk:** Booleano, heredado. Este atributo se usa para controlar como se extrae manejadores con campos binarios que han sido truncados (debido a ser el valor del campo más largo que el atributo *LongReadLen*). Por defecto es falso y provocará que la extracción falle. Muchos controladores permiten continuar extrayendo más tuplas cuando este atributo esta puesto en *false*.

6.4.6 Métodos de los manejadores de bases de datos

- **prepare:** este método prepara una sentencia simple de SQL para su ejecución y devuelve una referencia al manejador de la sentencia *($sth),* para usarlo con el objetivo de coger los atributos de la sentencia cuando se ejecute la sentencia. Algunos *drivers* sólo guardan la sentencia en el manejador y sólo pueden dar información útil después de que el método *execute* haya sido llamado.

- **do:** este método prepara y ejecuta una sentencia. Devuelve el número de tuplas afectadas o -1 si no es conocido, o *undef* si hay un error. Este método es usualmente usado para sentencias *no-select* (que no son consultas), las cuales no necesitan ser preparadas o ejecutadas por separado.

- **Commit:** este método hace persistente una operación realizada en la base de datos. Puede no funcionar con algún sistema gestor de bases de datos.

- **rollback:** deshace un *commit*. Puede no funcionar con algunos sistemas gestores de bases de datos.

- **disconnect:** Este método cierra la conexión entre la base de datos y el manejador de la base de datos. Normalmente se suele usar justo antes de terminar la aplicación. Si se llama a este método mientras hay manejadores de sentencias activos se obtendrá un mensaje de aviso. Hay que usar el método *finish* para cada manejador de sentencia definida.

- **ping:** Este método mira si el servidor de base de datos y la conexión siguen funcionando. No se suele usar.

- **quote:** Este método quita caracteres especiales (comillas; etc.) en las cadenas y añade las comillas externas. No es aplicable a todos los tipos de entrada (ejemplo, datos binarios).

Ejemplo que utiliza los métodos anteriormente explicados:

```
#!/usr/bin/perl -w
use DBI;
use strict;
```

```
my $dbh = DBI->connect (“DBI:mysql:contactos:localhost:3306”
,”root”,”123456789”) or die "No se puede conectar a la BD:
$dbh->errstr\n";
my $sth = $dbh->prepare(“SELECT uid FROM contacto WHERE apellido1
=‘Gomes’”);
$sth->execute or die  “No se puede ejecutar consulta: $dbh->
errstr\n”;
my $row = $sth->fetchrow_arrayref;
my $uid = $row->[0];
$sth->finish;
my $nuevo_nombre = $dbh->quote(“Gomez”);
my $sentencia =qq(UPDATE contacto SET last_name = '$nuevo_nombre'
WHERE uid = $uid);
my $rc = $dbh->do($sentencia) or die “No se puede preparar /
ejecutar la sentencia: $dbh->errstr\n”;
print “$rc tuplas han sido actualizadas\n”;
$dbh->disconnect;
exit;
```

6.4.7 Atributos de los manejadores de bases de datos

AutoCommit: puede no funcionar con algún sistema gestor de bases de datos.

6.4.8 Métodos de los manejadores de sentencias

- **bind_param:** este método se usa para asignar valores a un *placeholder* (marcador de posición) en una sentencia preparada. *%attr* puede ser usado para especificar el tipo de dato del *placeholder*. Por ejemplo:

```
$sth->bind_param(1, $value, { TYPE => SQL_INTEGER });
```

Se usa un atajo para unir un tipo directamente en el lugar de una referencia de una memoria asociativa. Por ejemplo:

```
$sth->bind_param(1, $value, SQL_INTEGER);
```

- **bind_param_inout:**

```
$sth->bind_param_inout($param_num,\$bind_value,$max_len);
$sth->bind_param_inout($param_num,\$bind_value,$max_len,\%attr);
$sth->bind_param_inout($param_num,\$bind_value,$max_len,
$bind_type);
```

- **execute:** este método ejecuta una sentencia preparada y devuelve *true* si hay éxito y *undef* si ocurre un error. Para las sentencias que no son *SELECT (UPDATE*, *INSERT*; etc.*)* el valor de retorno es el número de tuplas afectadas. Cero tuplas son devueltas como *'0E0'* que Perl trata como '0'; pero que es

tratado como *true*. Para sentencias *SELECT* la ejecución comienza la maquinaria de consulta. Se necesitan usar uno de los métodos de extracción, para obtener datos. Si se omite algún argumento entonces el método *execute* llama a *bind_param* para cada valor, y pone el tipo a *SQL_VARCHAR*.

- **fetchrow_arrayref:** este método extrae la siguiente tupla con datos y devuelve una referencia a un arreglo con los valores de los campos. Si no hay más tuplas que obtener, devuelve *undef*. Cuando es usado con el método *bind_columns*, es el modo más rápido de extraer datos.

- **fetchrow_array:** es igual a *fetchrow_arrayref* excepto que devuelve un arreglo con los valores de los campos, en lugar de una referencia a un arreglo.

- **fetchrow_hashref:** otro método alternativo para extraer tuplas de datos. Este método devuelve una referencia a una memoria asociativa *(hash)* conteniendo para cada campo el nombre y el valor. Las claves de la memoria asociativa son los mismos que los nombres de los campos devueltos por el método *$sth->{NAME}*. Este método es menos eficiente que el anterior.

- **fetchall_arrayref:** este método es usado para obtener todos los datos (tuplas) devueltos por la sentencia SQL. Devuelve una referencia a un arreglo de arreglos de referencia.

- **finish:** Este método es usado cuando no se va a extraer más datos de un manejador de sentencia, antes de que sea preparado de nuevo o desechado. Este método es el más usado para mantenimiento interno (liberar recursos como cuando se bloquea la lectura). No hace falta llamar a este método si será desechado o volver a usar el manejador de sentencia.

- **rows:** este método devuelve el número de o filas afectadas por la última sentencia *no-select* ejecutada.

- **bind_col:** este método asigna una columna (campo) a una variable. Cuando una columna es captada *(fetched),* la correspondiente variable es automáticamente actualizada, esto hace que la extracción sea muy eficiente.

- **bind_columns:** Este método llama a *bind_col* para cada columna de la sentencia *SELECT*.

6.4.9 Atributos para manejadores de sentencias. La mayoría de estos atributos son de sólo lectura. Algunos *drivers* no proporcionan los valores hasta después de ejecutar el método que ha sido llamado.

- **NUM_OF_FIELDS:** este atributo guarda el número de campos que la sentencia ya preparada devolverá.

- **NUM_OF_PARAMS:** este atributo almacena el número de parámetros *(placeholders)* que están preparados en la sentencia.

- **NAME:** este atributo devuelve una referencia a un arreglo de nombres de campos para cada columna.

- **NULLABLE:** este atributo devuelve una referencia a un arreglo indicando cual es devuelto como *NULL (true / false).*

- **CursorName:** devuelve el nombre del cursor asociado con el manejador de sentencia. Puede no funcionar con algunos motores de bases de datos.

6.4.10 Depuración.

- DBI_TRACE: además del método de seguimiento *trace* se puede activar información de seguimiento activando la variable de entorno *DBI_TRACE* antes de ejecutar el *script*. Si se activa *DBI_TRACE* a un valor no numérico se asume que es el nombre de un fichero donde se añade toda la información de seguimiento.

6.5 EVALUACIÓN DE LA METODOLOGÍA

En la evaluación de esta metodología se utilizo la plataforma Linux SuSE 7.0, el motor de base de datos relacional MySQL 3.22.32, Perl versión 5.005 y el modulo DBD::mysql versión 2.01 y el servidor Web Apache versión 1.3.12

El siguiente formulario es enviado por el cliente. El código ejemplo muestra el uso del modulo DBD:mysql con DBI para realizar inserciones en la base de datos:

```
<html><body>
<form action="/cgi-bin/insertar.pl" method="get">
cedula<input type="text" name="cedula" size=8 maxlength=9>
nombre<inputtype="text" name="nombre" size=20 maxlength=21>
apellido1<input type="text" name="apellido1" size=11
maxlength=12>
apellido2<input type="text" name="apellido2" size=11 maxlength=12>
edad<input type="text" name="edad" size=2 maxlength=2>
<input type="submit" value="enviar datos"><input type="reset"
value="borrar campos">
</form>
</html></body>
```

Y ejecuta en el servidor el *CGI* correspondiente:

```
#!/usr/bin/perl -w
```

```
#insertar.pl
use DBI;
use CGI;
$tabla ="cliente";
$basededatos ="prueba";
#DBI:mysql es el driver
$origen_de_datos ="DBI:mysql:$basededatos";
$max_entradas = 100;
print "Content-type: text/html\n\n"; #tipo de datos a imprimir en
STDOUT
my $var1 = $ENV{'REQUEST_METHOD'};
my $var2 = $ENV{'CONTENT_LENGTH'};
if($var1 eq 'GET')
{ #divide el par nombre-valor
  @parejas = split( /&/, $ENV{"QUERY_STRING"})
}
else
{  &error('metodo'); }
# se quita el signo igual, y se escriben las parejas en un
#sencillo hash
foreach $par(@parejas)
{ ($nombre, $valor) = split(/=/, $par);
   # remplazar el signo mas por espacio en blanco y codigo
   #hexadecimal por su correspondiente en ASCII
    $valor=~tr/+/ /;
    $valor=~s/%([a-f A-F 0-9][a-f A-F 0-9])/pack("C", hex($1))/eg;
```

```
    $valor =~ s/<!--(.|\n)*-->//g;

   $FORM{$nombre} = $valor;
   }
# se requieren variables con valores introducidos por el usuario.
foreach $item (cedula, nombre, apellido1, apellido2, edad)
{  if( $FORM{$item} eq "" )
  { &error('buscando_datos');   }
}
# se realiza la conexion a la base de datos
$login ="root";
$password ="12345678”;
$dbh = DBI->connect("$origen_de_datos:localhost:3306", $login,
            $password) or &error('conectandose_a_basededatos');

$sentencia = "select * from $tabla";
$sth = $dbh->prepare($sentencia) or &error('prepara_db');
$sth->execute or &error('ejecutando_en_db');
$filas_retornadas = $sth->rows;
$sth->finish;
if($filas_retornadas >= $max_entradas)
{ &error('max_entradas'); }
$sentencia="INSERT INTO $tabla (cedula, nombre, apellido1,
apellido2, edad)".
“VALUES  (\'$FORM{cedula}\'
         ,\'$FORM{nombre}\'
```

```
          ,\'$FORM{apellido1}\'
          ,\'$FORM{apellido2}\'
          ,\'$FORM{edad}\')";

$rc = $dbh->do($sentencia) or &error("ejecutando_sentencia");
# finaliza el manejador de sentencias
# desconecta de la base de datos
$dbh->disconnect;
# manejando los errores
sub error
{ #funcion para los mensajes de los diferentes errores que se
   #puedan presentar
}
exit(0);
```

6.6 RESULTADO DE LA EVALUACIÓN

6.6.1 Lenguaje. Para desarrollar aplicaciones de acceso a bases de datos utilizando el API DBI, es indispensable poseer un buen conocimiento del lenguaje Perl, ya que, según sea el grado de conocimiento que se posea del lenguaje, así de potentes y eficientes serán los resultados. Perl es sustancioso: un poco de código hace mucho. El lenguaje es mas bien funcional que elegante, es legible y no es difícil adquirir fluidez en la escritura del código, es necesario conocer y manejar las variables de ambiente del protocolo HTTP para la implementación de los *scripts CGI*

6.6.2 Soporte a plataformas. En la actualidad Perl es soportado por casi todas las plataformas computacionales existentes. Debido a la transparencia que brinda el API DBI, este permite desarrollar aplicaciones de acceso a bases de datos, sin preocuparse por entender como los mecanismos que acceden el API se conectan con el gestor de datos. La portabilidad del lenguaje admite implementar una aplicación en una plataforma y portarla a otra (aunque la portabilidad y eficiencia del código, puede verse afectada al pasarla de plataformas *Microsoft* a plataformas *Unix* y viceversa).

6.6.3 Soporte a bases de datos. Posee un buen soporte para múltiples gestores de bases de datos, ya este soporte no corresponde al API DBI sino al modulo o *driver* DBD correspondiente, el cual esta diseñado para un gestor especifico, estos además son constantemente actualizados y desarrollados. También permite acceder a los datos a través de otras interfaces tales como ADO, ODBC y JDBC usando los módulos respectivos.

6.6.4 Mantenimiento. El mantenimiento de las aplicaciones puede ser algo complicado, debido a que en este lenguaje se puede realizar la misma cosa de distintas maneras, un problema puede tener múltiples soluciones y diferentes programadores pueden interpretar y desarrollar el mantenimiento de estas, desde diferentes puntos de vistas.

6.6.5 Soporte. La interfaz de base de datos para Perl5 es un software gratis; pero este viene sin ninguna garantía de cualquier tipo. Posee una excelente documentación y buen soporte gratis en el Web.

7. LA INTERFAZ ODBiC

ODBiC (Open DataBase Internet Conector) proporciona una interfaz de servidor entre las paginas *Web* y la interfaz ODBC. Esta interfaz provee comandos para conectarse a la base de datos y ejecutar sentencias SQL soportadas por ODBC, además de comandos útiles que facilitan la generación de paginas *Web.*

ODBiC lee plantillas o *scripts* como entradas. Las plantillas generalmente contienen código HTML y texto, que es directamente enviado a la salida estándar. (véase la Figura 21). Estas plantillas contienen además sentencias de comandos ODBiC y variables referenciadas.

Cuando ODBiC encuentra una sentencia de comandos la ejecuta inmediatamente. Cuando encuentra una variable referenciada, el valor de la variable es insertado en su lugar en el texto de salida.

7.1 PLANTILLAS ODBiC

Las plantillas son la entrada para ODBiC y estos controlan la salida. Estos archivos son muy similares a un archivo HTML normal, de hecho se puede partir

de una pagina *Web* y luego embeber los comandos ODBiC. También es posible generar una respuesta sin una plantilla de entrada; pero los mejores resultados se obtienen con una plantilla.

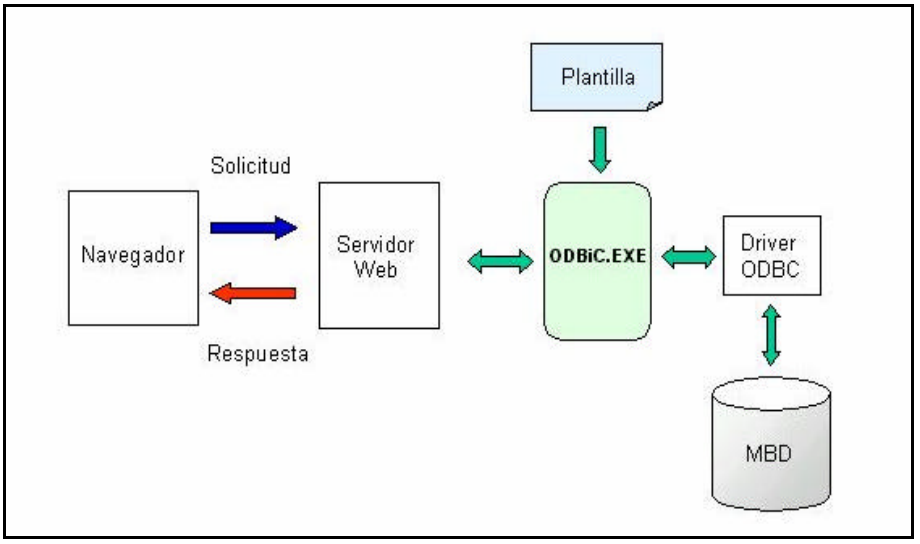

Figura 21. Arquitectura de funcionamiento de la interfaz ODBiC

OBDiC se puede utilizar de dos maneras: como programa CGI para acceder los datos en tiempo real o puede ser usado como un programa de DOS para generar paginas *Web* y guardarlas en disco.

Cuando se utiliza ODBiC como un programa CGI, las plantillas se pueden especificar de dos maneras. Puede incluirse como parte del URL que se usa para invocar o ejecutar ODBiC, si se usa en la función *ACTION* de una declaración *FORM*, el URL debe apuntar hacia el programa ODBiC (Este debe estar en un directorio conocido por el servidor, como *scripts* o *cgi-bin*). Ejemplo:

```
<FORM METHOD= “post” ACTION = “http:// www .sitio.com /scripts
/odbic.exe”>
```

La interfaz permite incluir una “ruta extra” en el URL. Esta información “extra” se utiliza para indicarle al programa, el nombre de la plantilla a leer y la ruta hacia esta, por ejemplo:

```
<FORM METHOD= “post” ACTION = “http:// www.sitio.com /scripts
/odbic.exe/directorio_plantillas/plantilla1.odb”>
```

Esto le indicara a ODBiC procesar un archivo llamado “plantilla1.odb” y este se encuentra en un directorio llamado “directorio_plantillas”, este directorio debe ser un subdirectorio del directorio principal o *root* del servidor *Web* que contenga los archivos HTML. Es posible que las plantillas se encuentren directamente en directorio *root* y no en un subdirectorio, entonces se puede indicar directamente el nombre del archivo en el URL.

Se puede utilizar la interfaz desde la línea de comando de *DOS* para generar la salida en un archivo HTML en disco. Si no se hace necesario acceder la base de datos en tiempo real (por ejemplo, cuando los datos no cambian con mucha frecuencia); pero se desea tener información actualizada de esta, en el sitio. Con este método se puede evitar la sobrecarga de la base de datos por cada pagina de acceso.

7.2 VARIABLES Y COMANDOS

Las variables son referenciadas en un archivo o plantilla ODBiC como, un nombre de variable encerrada entre signos de peso ($), de la siguiente manera: `$var$.`

Las variables siempre deben comenzar con caracteres alfabéticos, en minúsculas o mayúsculas después del primer carácter se pueden incluir números, letras, guión de piso (_) o un espacio en blanco.

Las variables se pueden definir por cinco razones:

- Pueden ser variables de entradas para un formulario HTML o URL cuando ODBiC se ejecuta.
- Pueden contener datos de columnas devueltas por una base de datos.
- Pueden ser declaradas explícitamente en una plantilla.
- Pueden ser recibidas como "*cookies*" que previamente se han enviado al navegador.
- Y pueden ser variables predefinidas de la interfaz ODBiC.

Los comandos son embebidos en un archivo ODBiC usando las etiquetas de comentario del lenguaje HTML. Los navegadores ignoran estos comentarios; pero para ODBiC los caracteres "<! - -", indican el comienzo de un nombre de comando. Las sentencias de comando terminan con los caracteres " - - >" .

7.2.1 Comandos de ODBiC. Los siguientes son los comandos de la interfaz, se ofrece una descripción breve, pues una información detallada sobre estos y las variables predefinidas del lenguaje, esta fuera del alcance del documento.

- **BREAK:** termina el proceso de un ciclo.

- **DATABASE:** define una conexión de base de datos ODBC. Los argumentos típicos de este comando son "DSN" (*Data Source Name*), "UID" (usuario), "PWD" (clave de usuario) estos dos últimos si la base de datos admite seguridad y “DRIVER" (especificación explicita de un manejador).

- **DEFAULT:** asigna valores por defecto para variables.

- **EXEC:** ejecuta un comando de sistema o corre un programa de DOS.

- **EXIT:** termina el procesamiento de una plantilla.

- **EACHMULTI, ENDMULTI:** define un ciclo de procesamiento para múltiples variables. Todo el código entre *EACHMULTI* y *ENDMULTI* se repite para cada instancia de variables multi-definidas (por ejemplo: múltiples instancias de valores con el mismo nombre).

- **EACHROW, ENDROW:** define el formato para cada fila resultante después de ejecutar una sentencia *SELECT*.

- **FORM:** genera un formulario HTML con el atributo *ACTION* enlazado con ODBiC.

- **FORMAT:** define un formato de salida para variables.

- **HIDDEN:** este comando es una simple conveniencia para generar campos ocultos de formularios HTML, tomados de una lista de variables dada.

- **IF, ELSE, ENDIF:** sentencias de condiciónales.

- **IFNEW:** comprueba si el valor de una variable ha cambiado desde la ultima vez que fue testada por una sentencia *IFNEW*.

- **IMPORT, ENDIMPORT:** lee y procesa variables de datos provenientes de un archivo de texto.

- **INCLUDE:** lee y procesa una plantilla adicional (externa).

- **INSERTFORM:** genera un formulario HTML para realizar inserción o modificación de datos e una tabla.

- **NOTE:** comentario en una plantilla, no se envía al navegador cuando la plantilla es procesada.

- **OPTIONLIST:** crea una lista de selección HTML para una consulta o devuelve una lista de valores. Puede ser usado para generar un formulario HTML de selección de campos.

- **OUTPUT:** escribe la salida a un archivo de disco. Especifica que los datos procesados van a ser escritos en un archivo.

- **QBE:** genera y ejecuta automáticamente una sentencia SQL SELECT examinando los campos de entrada (columnas) especificados.

- **REDIRECT:** redirecciona el navegador del usuario a un URL diferente.

- **SEARCH:** genera y ejecuta una consulta o búsqueda de cadena en una o mas columnas en una tabla de una base de datos.

- **SENDMAIL, ENDMAIL:** este comando envía el resultado como un mensaje de correo electrónico.

- **SET:** crea y asigna un valor a una variable.

- **SETCOOKIE:** envía una *"cookie"* al navegador del usuario.

- **SETMULTI:** define una nueva instancia para un arreglo. Es como *SET* solo que crea múltiples ocurrencias de la variable.

- **SHOWINPUT:** muestra una lista del nombre y valor de todas las variables de entrada CGI.

- **SQL:** ejecuta una sentencia SQL ODBC.

- **TABLE:** formatea el resultado de un *SQL SELECT*, todas las columnas y todas las filas, como una tabla HTML.

- **TRANSLATE:** el comando *TRANSLATE* permite sustituir el valor de una variable por otro.

- **UPDATEFORM:** genera un formulario HTML conteniendo valores obtenidos de la base de datos, para realizar una actualización.

- **VALIDATE:** este comando valida que la variable dada contenga la información que se desea, compara esta contra un patrón o expresión regular tipo UNIX. Si el patrón no esta especificado, este simplemente chequea que la variable contenga un valor.

- **WHILE, ENDWHILE:** define un ciclo que se repetirá hasta que la condición sea verdadera.

7.3 EVALUACIÓN DE LA INTERFAZ

Para la evaluación de la metodología se empleo la plataforma Windows 98, el servidor Web Apache versión 1.3.20, la interfaz ODBiC versión 1.6 y el motor de base de datos InterBase6, que provee el controlador EasySoft IB6 ODBC versión 1.00.01.58.

El siguiente código HTML, crea un formulario que permitirá la búsqueda de times conociendo uno o varios campos.

```
<html><body>
<form method="get" action="/scripts/odbic.exe/odbic/
consulta.odb">
idfruta<input name="idfruta" type="text" size="12
nombre<input name="nom_fruta" type="text" size="50">
caracteristica<input name="caracteristica" type="text"
```

```
size="50">
precio<input name="precio" type="text" size="8">
existencias<input name="existencia" type="text" size="12">
numero de resultados a mostrar <input name="num_filas" type="text"
value="5" size="3">
<b>ordenado por:</b>
<select name="ordenadopor" size="1">
      <option>idfruta</option><option>nombre</option>
      <option>precio, idfruta</option>
      <option>precio, nombre</option><option>precio</option>
      <option>existencia</option>
</select>
<input type="submit" value="buscar">
<input type="reset" value="borrar">
</form>.
</body></html>
```

Este formulario ejecutara el programa *odbic.exe* que se encuentra en el directorio *scripts* del servidor *Web*, a este se pasa como entrada la plantilla *consulta.odb,* que recibe los datos del formulario y cuyo código es el siguiente:

```
<html>
<!-- (consulta.odb - plantilla para "Busqueda de productos") -->
<!-- (el criterio de seleccion proviene del formulario html) -->
<body>

<!--DEFAULT num_filas = 10, ordenadopor = idfruta -->
```

```
<!--FORMAT precio="$###,##0.00" -->
Resultados de la busqueda de productos

<!--SET usuario=SYSDBA, clave=masterkey -->
<!--DATABASE DSN=interfrutas; UID=$usuario$; PWD=$clave$ -->
<!--QBE   TABLE=FRUTA,   idfruta,   'nom_fruta',   'caracteristica',
precio, existencia, ROWS=$num_cols$, ORDER=$ordenadopor$ -->
<!--IF $row$ = 0 -->
    <h3><I>No hay resultados</I></h3>
    <!--IF $sql_status$ = -1 --> $sql_error$<P>
    <!--ENDIF-->
<!--ELSE-->
   <table border cellspacing="0" cellpadding="5">
   <tr><th>Idfruta</th><th>Nombre</th><th>Caracteristica</th>
   <th>Precio</th><th>Existencia</th></tr>
   <!--EACHROW-->
      <tr><td>$idfruta$</td><td>$nom_fruta$</td>
      <td>$caracteristica$</td><td>$precio$</td>
      <td>$existencia$</td></tr>
   <!--ENDROW-->
   </table>
<!--ENDIF-->
</body></html>
```

7.4 RESULTADO DE LA EVALUACIÓN

La versión que se evaluó de esta herramienta consistió en un ejecutable *odbic.exe* de 129 kilo bites, es un claro ejemplo de la tecnología CGI y por lo tanto conlleva todo lo que ello significa. Como esta herramienta existen muchas en el mercado, las hay para la plataforma Unix pero la mayoría esta dirigida principalmente a la plataforma *Windows*. Este tipo de herramienta no es recomendable para

operaciones que demanden grandes volúmenes transacciones a la base de datos.

7.4.1 Lenguaje. No es un lenguaje de programación orientado a la programación de aplicaciones. Aunque el conjunto de etiquetas es relativamente pequeño, versátil y eficiente, lo que permite en cierta manera que su aprendizaje sea relativamente practicable; el manejo de los comandos, funciones y variables predefinidas, y la generación de las plantillas a partir de estos es sencillo no deja de ser tedioso. Como lenguaje de etiquetas es muy completo por que soporta ciclos y sentencias de decisión.

7.4.2 Soporte a plataformas. El soporte multiplataforma es verdaderamente limitado ya que esta interfaz solamente puede trabajar sobre las plataformas de *Microsoft*: Windows 98 y Windows NT 4.0

7.4.3 Soporte a bases de datos. La única manera de acceso a las fuentes de datos es a través de la interfaz ODBC. La característica de trabajar con este estándar permite que se puedan desarrollar aplicaciones que se puedan conectar virtualmente con cualquier motor de base de datos que posea un controlador ODBC, y por lo tanto esta limitada a las ventajas y desventajas de dicha interfaz.

7.4.4 Flexibilidad. Es una herramienta que posee funciones, variables predefinidas y comandos que simplifican el tratamiento de requerimientos comunes y la hacen muy versátil. Por ejemplo:

- Generación automática de tablas con información proveniente de consultas

a bases de datos.

- Generación automática de consultas booleanas de palabras clave, creadas a partir de las entradas de usuario.
- Creación dinámica de formularios y sentencias SQL para la operaciones de inserción *(INSERT)* y actualización *(UPDATE)* de datos.

7.4.5 Soporte. No es una herramienta de uso muy extendido y poco conocida, ya que hace parte de un grupo de productos *pequeños* que son en realidad "*propuestas de solución".* No posee una comunidad de desarrolladores conocida hasta el momento de la edición de este trabajo. Se debe pagar una suma módica por licencia.

8. FUNCIONES DE PHP PARA EL ACCESO A BASES DE DATOS

PHP es un lenguaje interpretado de alto nivel, embebido en páginas HTML y ejecutado en el servidor. El siguiente ejemplo nuestra como se aplica lo anterior:

```
<html>
<head>
<title>Ejemplo PHP</title>
</head>
<body>
<?php echo "Hola, este es un ejemplo con PHP!"; ?>
</body>
</html>
```

Se puede notar, que no es lo mismo que un *script CGI* escrito en otro lenguaje de programación como Perl o C. En vez de escribir un programa con muchos comandos para crear una salida en HTML, se escribe el código HTML con cierto código PHP embebido en el mismo, que producirá cierta salida (en el ejemplo, producir un texto). El código PHP se incluye entre etiquetas especiales de comienzo ” <? ” y final “?> ” que permitirán entrar y salir del modo PHP.

Si se tuviera un *script* similar al del ejemplo, en el servidor, el cliente solamente recibirá el resultado de su ejecución en el servidor, sin ninguna posibilidad de determinar que código ha producido el resultado recibido. (Véase la Figura 22). Un servidor Web puede ser incluso configurado para que procese todos los ficheros *HTML* con PHP.

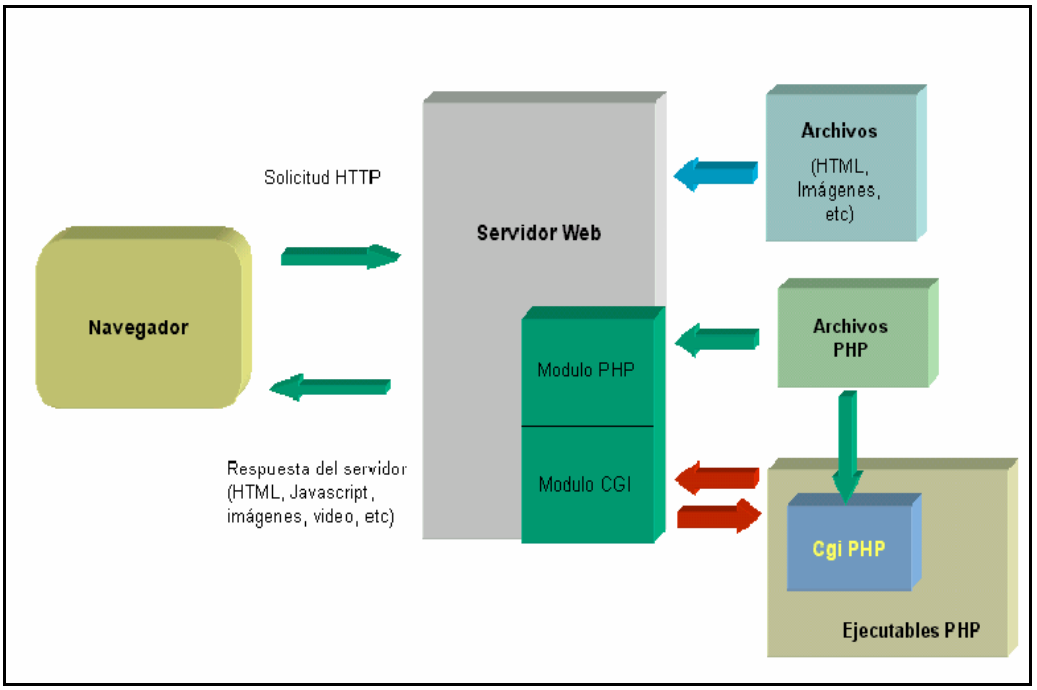

Figura 22. Arquitectura de PHP

8.1 UTILIDAD DE PHP

A un nivel más básico, PHP puede hacer cualquier cosa que se pueda hacer con un *script CGI*, como procesar la información de formularios, generar páginas con contenidos dinámicos, o mandar y recibir *cookies*. La característica más potente y

destacable de PHP es su soporte para una gran cantidad de bases de datos (ver Tabla 26). Diseñar y escribir una interfaz para acceder una base de datos vía Web, es una tarea simple.

Tabla 26. Bases de datos soportadas por PHP

Gestores de bases de datos soportados		
Adabas D	Ingres	Unix dbm
DBase	InterBase	Velocis
Empress	FrontBase	Sybase Solid
FilePro	MSQL	PostgreSQL
IBM DB2	ODBC	Oracle 7
Informix	MySQL	Oracle 8

PHP posee funciones para acceder a estas bases de datos. Estas funciones son independientes y adaptadas a las características propias de cada motor de base de datos, es decir, las funciones de interfaz de Informix son diferentes e independientes de las funciones diseñadas para Sybase.

Para evaluar el método de acceder a bases de datos a través del lenguaje PHP (versión 3.03), se hizo uso de las librerías de funciones de PHP para PostgreSQL, que sirven de interfaz entre la aplicación y la base de datos.

8.2 FUNCIONES PHP PARA ACCEDER A PostgreSQL 7.0

- **pg_Close:** `bool` **`pg_close`** `(int connection);`

Cierra una conexión PostgreSQL. Devuelve *false* si *connection* no es un índice de conexión valido y *true* en cualquier otro caso. Cierra la conexión a la base de datos PostgreSQL asociada con el índice de conexión pasado como parámetro.

- **pg_cmdTuples:** `int` **`pg_cmdtuples`** `(int result_id);`

Devuelve el número de tuplas afectadas. *pg_cmdTuples()* devuelve el número de tuplas (instancias o filas) afectadas por consultas *INSERT, UPDATE* y *DELETE*. Si no hay ninguna tupla afectada la función devolverá cero.

- **pg_Connect**

```
int pg_connect (string host, string port, string options, string
tty, string dbname);
```

Abre una conexión. Devuelve un índice de conexión en caso de éxito, o falso si la conexión no se puede realizar. Cada uno de los argumentos debe ser una cadena entrecomillada, incluyendo el número de puerto.

Los parámetros *options* y *tty* son opcionales y pueden ser omitidos. Esta función devuelve un índice de conexión que se necesitará para otras funciones PostgreSQL. Se puede tener múltiples conexiones abiertas al mismo tiempo.

Una conexión también se puede establecer con el siguiente comando: *$conn = pg_connect("dbname=marliese port=5432")*. Otros parámetros, aparte de *dbname* y *port* son *host, tty, options, user* y *password*.

- **pg_Dbname:** `string pg_dbname (int connection);`

Nombre de la base de datos. Devuelve el nombre de la base de datos a la cual es el índice de conexión con PostgreSQL está conectado, o *false* si *connection* no es un índice de conexión válido.

- **pg_ErrorMessage:** `string pg_errormessage (int connection);`

Mensaje de error. Devuelve una cadena que contiene el mensaje de error, o *false* en caso de fallo. Probablemente no se podrán obtener los detalles del error a través de la función *pg_errormessage()* si ocurre un error en la última acción de base de datos para la cual existe una conexión valida, esta función retornará una cadena conteniendo el mensaje de error generado por el servidor.

- **pg_Exec:** `int pg_exec (int connection, string query);`

Ejecuta una consulta (*query*). Devuelve un índice de resultado si se pudo ejecutar la consulta, o *false* en caso de fallo o si *connection* no es un índice de conexión válido. Se pueden obtener detalles acerca del error mediante la función *pg_ErrorMessage()* siempre que *connection* sea válido. Envía una sentencia SQL a la base de datos PostgreSQL especificada por el índice de conexión, *connection* debe ser un índice válido devuelto por *pg_Connect().* El valor de devuelto por esta función es un índice para ser usado al acceder a los resultados de la consulta desde otras funciones PostgreSQL.

- **pg_Fetch_Array:**

```
array pg_fetch_array (int result, int row [, int result_type]);
```

Obtiene una fila en la forma de un arreglo. Devuelve un *array* que se corresponde con la fila obtenida, o *false* si no hay más filas.*pg_fetch_array()* es una versión extendida de *pg_fetch_row().* Además de almacenar los datos en los índices numéricos del arreglo resultante, también almacena los datos usando índices asociativos, empleando para ello el nombre del campo como la llave o índice.

El tercer parámetro opcional *result_type* en *pg_fetch_array()* es una constante y puede tomar cualquiera de los siguientes valores: *PGSQL_ASSOC, PGSQL_NUM, y PGSQL_BOTH*.

- **pg_Fetch_Object:**

```
object pg_fetch_object (int result, int row [, int result_type]);
```

Obtiene una fila en forma de objeto. Devuelve un objeto cuyas propiedades se corresponden con los campos de la fila obtenida, o *false* si no hay más filas.

pg_fetch_object() es parecida a *pg_fetch_array()*, con una diferencia se devuelve un objeto, en vez de un arreglo. Indirectamente, eso significa que solo se puede acceder a los datos por medio de su nombre de campo, y no a través de sus posiciones (los números son nombres de propiedad inválidos).

El tercer parámetro opcional *result_type,* (*result_type* se añadió en PHP 4.0) en *pg_fetch_object()* es una constante y puede tomar cualquiera de los siguientes valores: *PGSQL_ASSOC, PGSQL_NUM, y PGSQL_BOTH*.

Referente a la velocidad, la función es idéntica a *pg_fetch_array()*, y prácticamente tan rápida como *pg_fetch_row()*, la diferencia es insignificante.

- **pg_Fetch_Row:** `array pg_fetch_row (int result, int row);`

Devuelve un arreglo que se corresponde con la fila obtenida, o *false* en el caso de que no haya más filas. *pg_fetch_row()* obtiene una fila de datos a partir del resultado asociado con el identificador de resultado especificado.

La fila se devuelve en forma de arreglo. Cada columna del resultado se almacena en una posición del arreglo, empezando a partir de la posición cero. Las siguientes llamadas a *pg_fetch_row()* devolverán la fila siguiente en el conjunto resultado, o falso en el caso de que no haya más filas que devolver.

- **pg_FieldIsNull:**

```
int pg_fieldisnull (int result_id, int row, mixed field);
```

Comprueba si un campo vale *NULL* o no. Devuelve cero si el campo en la fila dada no es nulo y uno en el caso de que lo sea. El campo se puede especificar mediante un número o un nombre de campo. La numeración de filas empieza en cero.

- **pg_FieldName:**

```
string pg_fieldname (int result_id, int field_number);
```

Devuelve el nombre de un campo. *pg_FieldName()* devolverá el nombre del campo que ocupa el número de columna dado en el identificador de resultado de PostgreSQL. La numeración de los campos empieza con cero.

- **pg_FieldNum:** `int pg_fieldnum (int result_id,string field_name);`

Devuelve el número de una columna. *pg_FieldNum()* devolverá el número de la columna que corresponde al campo cuyo nombre se le da, dentro del identificador de resultado de PostgreSQL. La numeración de campos comienza en cero. Esta función devolverá -1 en caso de error.

- **pg_FieldPrtLen:**

```
int pg_fieldprtlen (int result_id, int row_number, string
field_name);
```

Devuelve la longitud impresa. *pg_FieldPrtLen()* devolverá la longitud impresa real (número de caracteres) de un valor específico dentro del identificador de resultado PostgreSQL. La numeración de filas comienza en cero. Esta función devolverá -1 en caso de error.

- **pg_FieldSize:** `int pg_fieldsize (int result_id, int field_number);`

Devuelve el tamaño de almacenamiento interno de un campo en concreto. *pg_FieldSize()* devolverá el tamaño de almacenamiento interno (en *bytes*) de uno de los campos del resultado PostgreSQL que se le ha pasado. La numeración de campos empieza en cero. Un tamaño de campo de -1 indica que se trata de un campo de longitud variable. La función devolverá *false* en caso de error.

- **pg_FieldType:** `int` **`pg_fieldtype`** `(int result_id,int field_number);`

Devuelve el nombre del tipo de dato correspondiente al campo cuyo número se pasa como parámetro. *pg_FieldType()* devolverá una cadena con el nombre del tipo de datos de un campo dado dentro del identificador de resultado PostgreSQL *result_id*. La numeración de campos empieza en cero.

- **pg_FreeResult:** `int` **`pg_freeresult`** `(int result_id);`

Libera memoria. *pg_FreeResult()* solo necesita ser llamada si se esta preocupado por usar demasiada memoria mientras el *script* se está ejecutando. La memoria correspondiente a todos los resultados de consulta se libera automáticamente cuando termina el *script*.

Pero, si sé esta seguro de que no se va a necesitar más los datos del resultado en el *script*, se puede llamar a *pg_FreeResult()* con el identificador del resultado como parámetro y la memoria asociada al resultado será liberada.

- **pg_GetLastOid:** `int` **`pg_getlastoid`** `(int result_id);`

Devuelve el identificador del último objeto insertado. *pg_GetLastOid()* se puede usar para conseguir el *Oid* (identificador de objeto) asignado a una tupla insertada si el identificador de resultado proviene de una llamada a *pg_Exec()* que fuese un

INSERT SQL.

Esta función devuelve un entero positivo si hay un *Oid* válido y -1 en caso de que ocurriese un error durante el último comando enviado a través de la función *pg_Exec()* o si esta no fuese un *INSERT*.

- **pg_Host:** `string` **`pg_host`** `(int connection_id);`

Devuelve el nombre del *host. pg_Host()* devuelve el nombre del *host* al que el identificador conexión PostgreSQL pasado, está conectado.

- **pg_loclose:** `void` **`pg_loclose`** `(int fd);`

Cierra un objeto grande *(large object). pg_loclose()* cierra un *Large Object*. fd es el descriptor de fichero del fichero grande obtenido a través de *pg_loopen()*.

- **pg_locreate:** `int` **`pg_locreate`** `(int conn);`

Crea un objeto grande. *pg_locreate()* Crea un *Large Object* y devuelve su *oid*. *conn* determina una conexión de base de datos válida. Los modos de acceso I*NV_READ, INV_WRITE*, y *INV_ARCHIVE* de PostgreSQL no están soportados, el objeto se crea siempre con acceso tanto de lectura como de escritura.

- **pg_loopen:** `int` **`pg_loopen`** `(int conn, int objoid, string mode);`

Abre un objeto grande. *pg_loopen()* abre un *Large Object* (objeto grande) y devuelve un descriptor de fichero para el objeto grande. El descriptor de fichero encapsula información acerca de la conexión. No se debe cerrar la conexión antes de cerrar el descriptor de fichero al objeto grande. *objoid* especifica un *oid* válido para un objeto grande y *mode* puede ser "r", "w", o "rw".

- **pg_loread:** `string` **`pg_loread`** `(int fd, int len);`

Lee un *large object* (objeto grande). *pg_loread()* lee como mucho cien *bytes* a partir de un objeto grande y lo devuelve como una cadena. *fd* especifica un descriptor de fichero de objeto grande válido y *len* especifica máximo número de *bytes* que se deben leer del objeto grande.

- **pg_loreadall:** `void` **`pg_loreadall`** `(int fd);`

Lee un objeto grande entero. *Pg_loreadall()* lee un objeto grande y lo pasa tal cual al navegador, después de enviar todas las cabeceras pendientes. Principalmente dirigido a mandar datos binarios como imágenes o sonido.

- **pg_lounlink:** `void` **`pg_lounlink`** `(int conn, int lobjid);`

Borra un *large object*. *pg_lounlink()* borra el objeto grande con identificador *lobjid*.

- **pg_lowrite:** `int` **`pg_lowrite`** `(int fd, string buf);`

Escribe en un objeto grande. *pg_lowrite()* escribe todo lo que puede en un objeto grande a partir de la variable *buf* y devuelve el número de *bytes* realmente escritos, o falso si ocurre algún error *fd* es un descriptor de fichero para el objeto grande obtenido a través de *pg_loopen().*

- **pg_NumFields:** `int` **`pg_numfields`** `(int result_id);`

Devuelve el número de campos. *pg_NumFields()* devuelve el número de campos (columnas) en un resultado PostgreSQL. El parámetro es un identificador de resultado válido devuelto por *pg_Exec()*. La función devuelve -1 en caso de error.

- **pg_NumRows:** `int` **`pg_numrows`** `(int result_id);`

Devuelve el número de filas. *pg_NumRows()* devuelve el número de filas en un resultado PostgreSQL. El parámetro es un identificador de resultado PostgreSQL válido devuelto por *pg_Exec().* En caso de error se devuelve -1.

- **pg_Options:** `string` **`pg_options`** `(int connection_id);`

Devuelve opciones. *pg_Options()* devuelve una cadena que contiene las opciones especificadas en el identificador de conexión con PostgreSQL dado.

- **pg_pConnect**

`int` **`pg_pconnect`** `(string host, string port, string options, string tty, string dbname);`

Crea una conexión persistente con una base de datos. Devuelve un índice de conexión en caso de éxito, o *false* si no es posible realizar la conexión. Abre una conexión persistente hacia una base de datos de PostgreSQL.

Cada uno de los parámetros puede ser una cadena entrecomillada *(quoted)*, incluyendo el número de puerto. Los parámetros *options* y *tty* son opcionales y pueden omitirse. Esta función devuelve un índice de conexión que luego será empleado al llamar a otras funciones PostgreSQL. Se pueden tener múltiples conexiones persistentes abiertas al mismo tiempo.

Una conexión también se puede establecer con el comando siguiente: *$conn = pg_pconnect("dbname=marliese port=5432").* Otros parámetros además de *dbname* y *port* son *host, tty, options, user* y *password*.

- **pg_Port:** `int` **`pg_port`** `(int connection_id);`

Devuelve el número de puerto. *pg_Port()* devuelve el número del puerto al que el identificador de conexión *connection,* está conectado.

- **pg_Result:**

`mixed` **`pg_result`** `(int result_id, int row_number, mixed fieldname);`

Devuelve valores a partir de un identificador de resultado. *pg_Result()* devuelve valores a partir de un identificador de resultado generado en la función *pg_Exec()*. Los parámetros *row_number* y *fieldname* especifican que celda en la tabla se quiere obtener.

La numeración de filas comienza en cero. En vez de usar el nombre del campo también se puede usar el índice del campo como un número sin entrecomillar. Los índices de campo comienzan también en cero. PostgreSQL tiene muchos tipos y solo los básicos están soportados directamente aquí. Todas las formas de enteros, booleanos y *oids* se devuelven como valores enteros. Todas las formas de los tipos *float* y *real* se devuelven como valores *double*. Todos los demás tipos, incluyendo los *array* se devuelven como cadenas formateadas de la misma manera en que PostgreSQL usa por defecto.

- **pg_tty:** `string` **`pg_tty`** `(int connection_id);`

Devuelve el nombre del *tty* (terminal). *pg_tty()* devuelve el nombre del *tty* hacia el que se dirige la salida de depuración del lado del servidor en el identificador de conexión de PostgreSQL dado.

8.3 EVALUACIÓN DE LA METODOLOGÍA

Para llevar a cabo la evaluación, se utilizo la plataforma computacional Linux SuSE 7.0 y el servidor Apache 1.3.12, el lenguaje de programación PHP, el motor de base de datos PostgreSQL versión 7.0.

Se implementaron los *scripts* que permitieron acceder a la base de datos.

El siguiente formulario se le envía al usuario en respuesta a la solicitud de modificar los datos de un registro.

```
<html><body>
 <?php
    if (!($cedula))
    { print
      " Debe digitar los datos
         </body>
         </html>"; exit;
    }
    else
```

```
   {
    $indice = pg_connect("host=localhost port=5432  user=root
    password=12345678  dbname=prueba");
    $sql = "SELECT * FROM cliente WHERE cedula = '$cedula'";
    $res = pg_exec($indice,$sql);
    $numfil = pg_numrows($res);
    if ($numfil = = 0)
    {
     print"
     No existe el registro con cedula $cedula</h2>
     </body>
     </html>";exit;
    }else{
     $arr = pg_fetch_array($res,0);
    }
   }
?>
<form action='actualizarphp.php3' method='post'>
  Clave del registro a modificar
 <input type='text' name='cedula' value = <?php print
 "$Qacedula "; ?>>
  Nombre <input type='text' name='nombre' value=<?php print
  "$arr[nombre]";?> >
  Apellidos<input type='text' name='apellidos'  value=<?php print
  "$arr[apellidos]";?> >
  Telefono<input type='text' name='telefono' value=<?php print
  "$arr[telefono]";?> >
  Email<input type='text' name='email' value=<?php print
  "$arr[email]";?> >
  <input type='submit' value='Modificar Datos'>
</form>
<?php
  pg_close($indice);
?></body></html>
```

El siguiente *script* realiza la actualización de la tabla en la base de datos, cuando recibe los datos del formulario:

```
<!--actualizarphp.php3-->
<html>
<body>
<?php
#si alguno de las variables de entrada no es recibida
if (!($cedula) or !($nombre) or !($apellidos) or !($telefono) or
!($email))
{print"
 <br><h2>Debe introducir la clave (Cedula)
 <p><a href=modificarphp.php3?cedula = $ cedula>
     Regresar</a></h2>
 </body> </html>";
}
else
{ #se obtiene el objeto de coneccion
  $indice = pg_connect("host=localhost port=5432 user=root
                        password=123456789 dbname=prueba");
  if (!$indice) #si no se obtuvo respuesta
  {
   Error_handler("Error al conectar  a base de datos" , $indice );
  }
  #de lo contrario, se crea la cadena con la sentencia sql

  $sql= "UPDATE cliente SET nombre='$nombre',
  apellidos='$apellidos',telefono='$telefono', email='$email' ";
  $sql .= "WHERE cedula = '$cedula'";
  $res = pg_exec($indice,$sql); #se ejecuta la sentencia
  print"

    <h2>Transaccion realizada, actualizado el registro.</h2>
```

```
      <a href = modificarphp.php3?cedula=$cedula>
      Regresar</a><h2>
      pg_close($indice);
 }
?></body></html>
```

8.4 RESULTADO DE LA EVALUACIÓN

8.4.1 Lenguaje. Conocer y aprender la sintaxis de PHP para la programación de aplicaciones CGI resulta sencilla, es un lenguaje fácil y practico. Esta hecho para escribir aplicaciones, es robusto y flexible.

Vale la pena destacar que el paquete de este lenguaje trae consigo otras herramientas o funciones para otras aplicaciones tales como generación de documentos XML, soporte Java, funciones de calendario, funciones para la generación de documentos *PDF*, funciones para pago electrónico, funciones criptográficas entre otros.

8.4.2 Soporte a plataformas. Posee un buen soporte para diferentes plataformas computacionales: *Microsoft (Windows 9x, 2000, NT)*, todas las variaciones de *Unix, Macintosh* y demás plataformas.

8.4.3 Soporte a bases de datos. Posee un excelente soporte nativo para bases de datos, además de poder trabajar con las interfaces JDBC y ODBC. Para crear interfaces Web-bases de datos no existe un API o interfaz común, las sintaxis de

los comandos para acceder a los gestores de bases de datos es diferente para cada uno, lo que no permite en un momento dado, compartir el código como sucede con otras interfaces, aunque el código puede ser desarrollado en una plataforma y migrado a otra sin ningún tipo de inconvenientes.

8.4.4 Flexibilidad. No tiene las herramientas suficientes para un manejo formal de los errores provenientes de la base de datos y el poco manejo que posee no es suficiente para atrapar errores y mostrarlos tal como son. Dependiendo de la configuración que tenga el motor PHP, del lado del servidor Web, las operaciones que se realicen sobre al gestor de base de datos, poseerán unos tiempos de respuesta muy rápidos, comparado con otras técnicas.

8.4.5 Soporte. Es una herramienta que hace parte del conglomerado *open source* o código abierto por lo que es gratuita, posee un buen soporte y excelente documentación en línea.

9. ASP Y ADO

9.1 ASP

Active Server Pages (ASP) es un entorno de *scripting* que *Internet Information Server (IIS),* el servidor Web de *Windows NT*, proporciona para la creación de aplicaciones Web. ASP en la practica, se constituye como un filtro *ISAPI* ubicado en el servidor para completar y extender la funcionalidad de *IIS*.

La extensión de estos archivos es *asp,* y son procesados en el servidor. Estan escritos en HTML y lenguajes de *script* tales como *JavaScript* o *VBScript*.

Cuando un usuario solicita, en su navegador, la presentación de una pagina ASP el servidor llama una extensión para procesar la pagina. Esta extensión ejecuta los *scripts* de servidor, se comunica con los componentes, construye el documento HTML puro sin que quede constancia alguna de *script* de servidor que lo ha generado, y lo envía al navegador el documento HTML.

9.2 ADO

ADO (ActiveX Data Objects), objetos de datos activos*,* es el componente central de la estrategia de *UDA (Universal Data Access)* Acceso a Datos Universal y de alto nivel de *Microsoft*. El *acceso a datos universal* proporciona un acceso de alto rendimiento a diversas fuentes de información, tanto relacionales como no relacionales, y una interfaz de programación independiente de la herramienta y del lenguaje.

ADO es un entorno de programación orientado a objetos que facilita el acceso y manipulación de fuentes de datos. Una aplicación que utilice ADO puede obtener acceso a servidor de base de datos a través de un proveedor de base de datos OLE DB.

Existen varios modelos de acceso a datos ADO, dentro de estos se encuentran las bibliotecas ADODB, ADOX y JRO, todas estas son tecnologías *Microsoft*. Cada una de estas bibliotecas tiene su particularidad:

- **La biblioteca ADODB:** es una biblioteca pequeña que contiene objetos fundamentales y ofrece las bases para la realización de conexión, la emisión de ordenes, la recuperación de conjuntos registros y la exploración de estos. Se puede utilizar para realizar las tareas básicas de mantenimiento, como modificar, añadir y borrar registros.

- **La biblioteca ADOX:** soporta el lenguaje de definición de datos *(DDL)* y los aspectos de seguridad. Ofrece objetos que ponen en contacto con el esquema completo de una base de datos, por ejemplo permite crear tablas y relaciones. El modelo incluye soporte para la integridad referencial, actualizaciones y borrados en cascada, ofrece procedimientos y vistas, así como colecciones *Users* y *Groups* para la seguridad de la base de datos por usuarios.
- **La biblioteca JRO:** permite las replicas de las bases de datos *Jet. Microsoft Jet* y los objetos de replicación (JRO) permiten agregar características a la aplicación que son específicas del motor de bases de datos *Microsoft Jet. Jet* y los objetos de duplicación están basados fundamentalmente en *Microsoft ActiveX Data Objects* (ADO), porque están conectados a un objeto *Connection* de ADODB. Sin embargo, *Jet* y los objetos de duplicación funcionan sólo con bases de datos de Microsoft Jet. Con los objetos *Jet* y *Replication* se puede:

    - Crear y sincronizar conjuntos de réplicas de bases de datos.
    - Compactar una base de datos y especificar opciones para ella, como contraseñas y el cifrado.
    - Escribir los cambios pendientes en los datos de la base de datos y leer a la memoria desde la base de datos los datos más recuentes para así actualizar la caché de memoria.

9.3 LA BIBLIOTECA ADODB

La biblioteca ADODB define un conjunto de solo siete objetos: tres principales y cuatro secundarios llamados colecciones. (Véase la Figura 23). El objeto *Connection* aparece en la parte más alta de la jerarquía. Pero se pueden crear conexiones de forma implícita utilizando otros objetos. Los objetos *Connection, Command, Recordset* y *Field* tienen colecciones *Properties.*

ADODB tiene un modelo de objetos extremadamente sencillo que sin embargo, posee una gran potencia. Cualquier operación que desee realizarse con OLE DB, esta soportada por estos siete objetos.

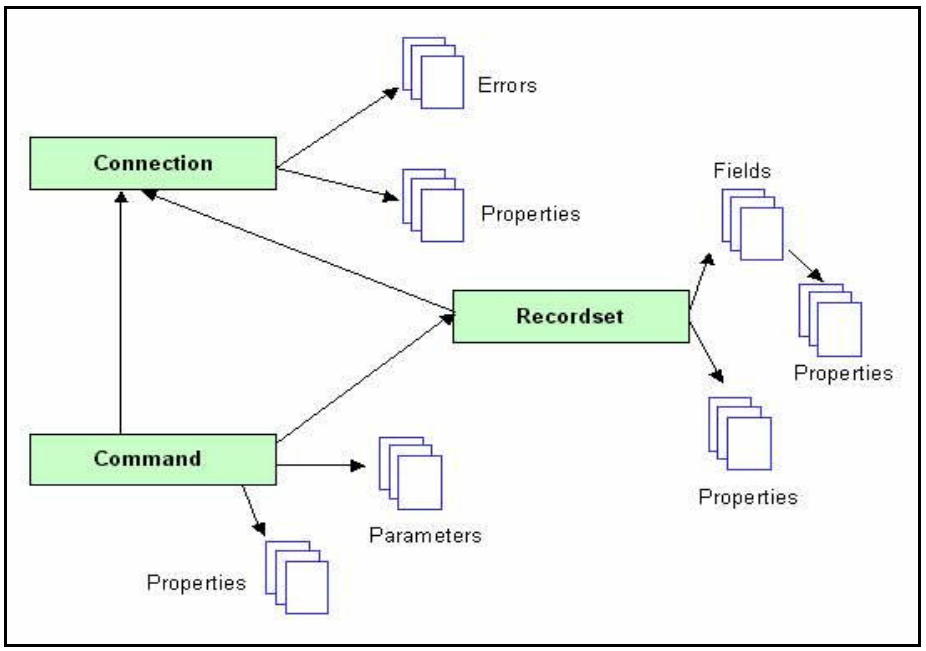

Figura 23. Jerarquía de los objetos ADO

9.3.1 El objeto Connection. El objeto *Connection* representa una conexión con una fuente de datos OLE DB o ODBC. Sobre esta conexión se podrá acceder a los datos de la mencionada fuente de datos. Se utiliza tanto *implícitamente* como *explícitamente* cuando se trabaja con una base de datos. Cuando se crea de forma explicita, se puede gestionar de forma eficiente una o más conexiones y reasignar los papeles que tienen en una aplicación. Al crear implícitamente un objeto *Connection* se puede hacer que la cantidad de código sea menor. Cada objeto nuevo que se crea con una conexión implícita consume mas recursos.

Los siguientes son algunos de sus métodos:

- ***Open:*** establece la conexión de acceso compartido.
- ***Close:*** termina la conexión.
- ***Execute:*** ejecuta un comando y devuelve un *recordset* (conjunto de registros) si es el caso.
- ***BeginTrans:*** inicia una transacción.
- ***CommitTrans:*** finaliza una transacción.
- ***RollbackTrans:*** finaliza una transacción, descartando las modificaciones.
- ***OpenSchema:*** obtiene información del esquema de la base de datos objeto de la conexión. Permite a una aplicación explorar los objetos disponibles en una colección a través de una conexión sin enumerar los elementos en una lista. La salida de este método puede contener información sobre tablas, vistas, procedimientos, índices y más.

Las siguientes son algunas de sus propiedades:

- ***Provider:*** elección del proveedor OLE DB.
- ***ConnectionString, ConnectionTimeout, Default Database:*** configuración de la conexión.
- ***Mode:*** determina el tipo de conexión para garantizar diversos tipos de acceso restringido a una base de datos. Estos modos de configuración son propios de todos los conjuntos de registros y ordenes que asignan una conexión a su propiedad *ActiveConnection*. La Tabla 27, describe ocho constantes que pueden utilizarse para configurar la propiedad *Mode* de una conexión. Se pueden utilizar estas constantes para controlar el tipo de edición que uno o más usuarios pueden realizar durante una conexión a una base de datos.
- ***CursorLocation:*** Establece o devuelve la posición de un servicio de cursores.
- ***IsolationLevel:*** nivel del aislamiento de las transacciones.
- ***CommandTimeout:*** tiempo máximo de validez del comando.
- ***Errors:*** errores devuelto por la fuente de datos.

Tabla 27. Constantes utilizadas para configurar la propiedad *Mode*

Constante	valor	Comportamiento
adModeUnknown	0	Permisos no configurados o indeterminados.
adModeRead	1	Permiso de solo lectura.
adModeWrite	2	Permiso de solo escritura.
adModeReadWrite	3	Permiso de lectura/escritura.
adModeShareDennyRead	4	Evita que otros abran el origen de registros con permisos de lectura.
adModeShareDennyWrite	8	Evita que otros abran el origen de registros con permisos de escritura.
adModeShareExclusive	12	Evita que otros abran la conexión.
adModeShareDennyNone	16	Acceso compartido (predeterminado).

Ejemplo:

```
‘se declara y abre una referencia a miconexion como un objeto
‘connection, se declara explícitamente la conexión.
Set miconexion = Server.CreateObject("ADODB.Connection")

‘Permiso de solo lectura
Miconexion.Mode = adModeRead
miconexion.ConnectionTimeout = 10
miconexion.CommandTimeout = 20
‘Declaracion del modo de conexión y de la fuente de datos DSN.
Miconexion.Mode = adModeRead
miconexion.ConnectionString = "DSN=midsn; DATABASE = prueba”
```

```
‘Se abre la conexion a traves del metodo Open sobre el objeto
‘miconexion, esto hace que la base de datos este disponible.
miconexion.Open
‘Se ejecuta un comando a través del objeto miconexion mediante el
‘metodo Execute y se obtiene un recordset como resultado
Set rs = miconexion.Execute(“select * from alumnos”)
‘ ..... Se efectúan las operaciones con los datos obtenidos en
‘el recordset
‘Importate: se cierra la conexion
miconexion.Close
```

9.3.2 El objeto Recordset. Un *recordset* (conjunto de registros) es una constructora de programación para trabajar con registros. Los registros se pueden basar en una tabla o en una vista en el proyecto actual, en otro archivo, en una sentencia SQL o una orden que devuelva filas. Este conjunto de registro será utilizado para obtener, mostrar y modificar los datos. Lo que se pueda hacer con un conjunto de registros depende del proveedor OLE DB y de los atributos originarios del origen de datos.

Al igual que se pueden extraer conjuntos de registros utilizando otros objetos como conexiones y órdenes, la buena mezcla de propiedades y métodos del objeto *Recordset* lo hacen una elección natural para el realizar gran parte del procesamiento del conjunto de filas.

Las operaciones que se pueden realizar con un conjunto de filas incluyen: explorar las filas; visualizar todos o algunos de sus contenidos; añadir, revisar y borrar registros; encontrar registros y filtrar registros para seleccionar uno o cualquier subconjunto de un conjunto de registros completo.

Los conjuntos de registros han sido históricamente no persistentes –normalmente existían solo el tiempo que estaba abiertos en un programa-. Desde la versión 2.1 de ADO, se ofrece conjunto de registros persistentes, que pueden ser guardados en discos y abiertos posteriormente.

Este objeto representa una tabla con todos sus registros, aunque uno solo de ellos es el activo. En el registro activo (también llamado *cursor*) es en el que se puede leer o modificar los valores de los campos. Los siguientes son algunos de los métodos y propiedades de este objeto:

- ***Open*:** abre la conexión con la base de datos (implícita), define y abre *recordsets*, ejecuta comandos SQL. Es una forma común de hacer que un conjunto de registros este disponible en un procedimiento.

Exploración del conjunto de registros. Los primeros cuatro métodos permiten la exploración del conjunto de registros mediante el cambio de la posición del registro actual:

- ***MoveFirst:*** mueve el cursor al primer registro del conjunto de registros. Este método funciona con todo los tipos de cursores.
- ***MoveLast:*** establece el ultimo registro en un conjunto de registros como la posición actual. Requiere de un tipo de cursor que permita el movimiento hacia atrás o, al menos, el movimiento basado en marcadores.
- ***MoveNext:*** mueve el cursor al siguiente registro. Sitúa la posición del registro actual en un registro posterior (en la dirección del registro final del conjunto de registros). Si la posición del registro actual es el ultimo registro, la propiedad *EOF* del conjunto de registros se establece a *True*. Si se llama este método cuando la propiedad *EOF* esta a *True* se genera un error en tiempo de ejecución.
- ***MovePrevious:*** mueve el cursor al registro anterior. Envía la posición actual del registro al registro al registro anterior. Si la posición del registro anterior es el primer registro la propiedad *BOF* del conjunto de registros se establece a *True.* Si se llama este método cuando la propiedad *BOF* del conjunto de registros esta ya en *True,* se provoca un error en tiempo de ejecución.
- ***Move:*** mueve el cursor a la fila activa del *recordset*. Trabaja en forma diferente a los otros cuatro métodos de exploración del conjunto de registros porque puede mover la posición del registro actual un numero variable de registros en cualquier dirección. Se usa un argumento positivo para indicar que se mueve hacia el último registro y un argumento negativo para identificar un movimiento hacia el primer registro. Si el movimiento se

extiende mas allá del primer o ultimo registros el método *Move* configura las propiedades *BOF* y *EOF* a *True.* Si esa propiedad esta ya configurada a *True*, el método provoca un error en tiempo de ejecución.

- ***Find:*** el método *Find* del objeto *Recordset* busca el primer registro que se ajuste a un criterio especifico de selección.

Métodos para agregar o eliminar registros de una tabla:

- ***AddNew:*** crea un nuevo registro en blanco. Añade nuevos registros a un conjunto de registros. Después de llamar el método, se definen los valores de los campos en la nueva fila que se desea añadir. Entonces se puede mover a fuera del registro mediante el método *Move* o llamar al método *Update* mientras se esta en la fila.
- ***Update:*** actualiza el nuevo registro. Envía los cambios almacenados en memoria y actualiza la tabla real.
- ***UpdateBatch:*** realiza una actualización por lotes.
- ***Delete:*** borra el registro actual.

Propiedades que hacen referencia al numero de registros:

- ***RecordCount:*** numero de registros de la tabla a la que representa el objeto *Recordrset*.

- ***EOF:*** *(End Of File).* Vale *true* si el cursor se encuentra en el ultimo registro y *false* si no.
- ***BOF:*** *(Begín Of File).* Vale *true* si esta en el primer registro y *false* si no.

Propiedades que hacen referencia al origen de los datos:

- ***ActiveConnection:*** indica al *Recordset* la base de datos en la que buscar la tabla. Permite a las aplicaciones aprovechar una conexión abierta, creada por el objeto *Connection* previamente. Su utilización simplifica el método *open,* eliminando la necesidad de incluir información de la conexión.
- ***Source:*** indica al objeto *Recordset* la tabla a la que representara. A la propiedad *Source* se le asigna normalmente una cadena de caracteres con el nombre de la tabla, también es posible asignarle una sentencia SQL y el objeto *Recordset* referenciará al resultado de aplicar dicha sentencia.
- ***CursorType:*** el tipo de cursor que se utiliza para recorrer los registros de un conjunto de registros. Por defecto toma el valor *adOpenForwardOnly* que como su nombre indica solo sirve permite desplazarse hacia delante. Si se quiere hacer uso de todos los métodos de movimiento de un conjunto de registros es necesario cambiar el cursor por uno más potente.

  Tipos de cursor: cada conjunto de registros contiene un *cursor,* es decir, un puntero en el *recordset* que en cada momento apunta al registro actual.

Navegar en el conjunto de registros supone, cambiar la ubicación del cursor. Existen diversos tipos de cursores, cada uno de ellos con una funcionalidad diferente.

- *Forward-Only*: es el tipo de cursor predeterminado, es de solo lectura y sin posibilidad de navegar hacia atrás. Actúa en una sola dirección y puede acelerar la acción del cursor
- *Keyset:* permite modificar registros y navegar en ambas direcciones con buenos rendimientos, pues no maneja los datos sino una clave que los representa. Las modificaciones hechas por un usuario no se reflejan en los de otro hasta que no vuelve a generarse el conjunto de registros.
- *Dynamic:* se ven las modificaciones de otros usuarios en el origen de datos. tiene activadas las funciones de mantenimiento, como añadir, cambiar y borrar registros, y permite la exploración bidireccional en una base de datos sin necesidad de marcas. Su rendimiento es menor.
- *Static:* permite navegar en ambos sentidos pero es de solo lectura. Este cursor es una instantánea de un conjunto de registros en un tiempo determinado. Los cambios en la base de datos realizados por otros usuarios no son visibles.

- ***LockType:*** indica el tipo de bloqueo que se realiza sobre la base de datos. Esta propiedad parcialmente interactúa con el tipo de cursor porque controla como los usuarios pueden manipular un conjunto de registros. Por

defecto toma el valor *adLockReadOnly* que indica que solo sirve para leer datos este se ajusta específicamente con el tipo de cursor *Forward-Only*. Si se quiere añadir o editar registros es necesario cambiar esta propiedad por otro valor. La Tabla 28, describe los cuatro tipos de configuraciones posibles de esta propiedad.

Tabla 28. Constantes utilizadas para utilizar la propiedad LockType

Constante	valor	Comportamiento
adLockReadOnly	1	Acceso de solo lectura.
adLockPessimistic	2	Bloquea un registro tan pronto un usuario comienza a editarlo.
adLockOptimistic	3	Bloquea un registro solo cuando un usuario elige comprometer las ediciones al volver a la base de datos.
adLockBatchOptimistic	4	Permite la edición de un lote de registros antes de intentar actualizar una base de datos remota desde el lote local de registros.

Otras propiedades son:

- ***Sort:*** esta propiedad puede afectar a los resultados de los métodos *Find* y *Move.* Esta propiedad designa uno o más campos que pueden determinar el orden en el cual se visualizan las filas. La configuración de la propiedad *sort* permite la designación de un orden ascendente o descendente para cualquier campo. El orden predeterminado es el ascendente.

La configuración de la propiedad *sort* no ordena físicamente las filas, simplemente se determina el orden en el cual el conjunto de registros pone disponible sus filas.

- ***Filtered:*** esta propiedad define para un conjunto de registros un nuevo conjunto de registros que es una versión filtrada del conjunto de registros original. Aunque esta propiedad posee aplicaciones especiales para la sincronización de bases de datos y actualizaciones por lotes de un origen de datos remotos, también puede ser una alternativa sencilla para definir un nuevo conjunto de registros basado en una instrucción SQL.

Ejemplo de *recordset*:

```
‘se declara y abre una referencia a miconexion como un objeto
‘Connection, se declara explícitamente la conexión.
Set miconexion = Server.CreateObject("ADODB.Connection")
miconexion.ConnectionString = "DSN=midsn"

‘se abre la conexion y queda disponible para el resto de la
‘aplicacion.
miconexion.Open

‘se declara y abre una referencia a mirecordset como un objeto
‘Recordset
Set mirecordset = Server.CreateObject("ADODB.Recordset")
‘se utiliza el objeto miconexion como una referencia (conexion
```

```
‘explicita), para aprovechar la conexión abierta
mirecordset.ActiveConnection = miconexion

‘se declara de que tabla se obtendrán los resultados
mirecordset.Source = "Clientes"

‘selecciona todos los registros de la tabla
mirecordset.Open
    ‘........  Operaciones con los datos
    ‘........ de la tabla Clientes.
‘se cierran los objetos
mirecordset.Close
miconexion.Close
```

9.3.3 El objeto Command. Este objeto es la representación de un comando que se le envía a la base de datos. Es altamente dependiente del controlador ODBC o del OLE DB *provider,* y a que describe un comando a enviar a la base de datos, cada gestor de base de datos tendrá su propio conjunto de comandos admisible, con lo que el objeto *Command* lleva aparejada una inherente heterogeneidad.

El objeto *Command* contiene la definición de una sentencia SQL cualquiera, que puede, por lo tanto realizar tareas como realizar una consulta, actualizar o insertar datos o ejecutar procedimientos almacenados en el servidor. Sus propiedades y métodos dependerán de la naturaleza de la fuente de datos.

Dentro de la biblioteca ADODB, los objetos *Command* presentan tres ventajas principales:

- Pueden realizar una consulta de selección para devolver un conjunto de filas de un origen de datos.
- Ejecutan una consulta de parámetros de forma que se puede introducir en tiempo de ejecución un criterio de búsqueda.
- Permiten consultas de acción en un origen de datos para realizar operaciones como actualizar, borrar y añadir registros.

Algunas propiedades y métodos son:

- ***CommandType*:** hay realmente varios tipos de objeto *Command.* Esta propiedad configura el tipo de objeto *Command.* La instrucción se puede basar en una sentencia SQL, una tabla o un procedimiento almacenado. La Tabla 29, muestra las constantes para esta propiedad.
- ***ActiveConnection*:** esta propiedad es una referencia al objeto *Connection* que enlaza con la base de datos. Es imprescindible asignar esta propiedad antes de invocar el método *Command.Execute.*
- ***CommandText:*** este es el texto del comando. Si se trata de una consulta SQL (lo habitual), esta propiedad es simplemente una cadena con el texto instrucción SQL.
- ***CommandTimeout:*** determina cuanto espera ADO para que concluya la

ejecución de una instrucción. Esta propiedad toma un valor *long* que especifica el máximo tiempo de espera en segundos. Su valor predeterminado es 30. Si el tiempo de espera transcurre antes de que el objeto *Command* complete su ejecución, ADO cancela la instrucción y devuelve un error.

- ***CreateParameter:*** este método crea un nuevo parámetro para la instrucción. Después de crear el parámetro, se puede utilizar el método ***Append*** para añadir el parámetro a la colección *Parameters* para una instrucción. Después de ejecutar una consulta de parámetros, también se tiene que asignar el valor al parámetro.
- ***Execute:*** el método que ejecuta la consulta. El resultado de la ejecución del comando normalmente será un conjunto de registros. Se pueden especificar hasta tres argumentos para este método. El primer argumento permite al objeto *Command* informar al procedimiento que lo llama de cuantos registros han sido afectados. El segundo argumento puede ser un *array Variant* con los parámetros para manejar la instrucción. El tercer argumento dice a ADO como evaluar el origen. Puede ser cualquiera de los nombres de constante listada en la Tabla 29.

Tabla 29. Constantes para CommandType

Constante	Valor	Comportamiento
AdCmdText	1	Permite ejecutar una instrucción en términos de una instrucción SQL, un procedimiento almacenado o incluso una tabla. Se reserva esta instrucción para una instrucción SQL.
AdCmdTable	2	Determina el conjunto de resultado de una tabla designada previamente. Devuelve todas las columnas de una tabla basada en una instrucción SQL generada de forma interna.
adCmdStoredProc	4	Ejecuta una instrucción en términos de un texto para un procedimiento almacenado.
AdCmdUnknow	8	Predeterminado. No hay especificación del tipo del texto de la instrucción.
AdCmdFile	256	Evalúa una instrucción basándose en el nombre del archivo para un conjunto de registros persistente.
adCmdTable Direct	512	Evalúa una instrucción como un nombre de tabla. Devuelve todas las columnas en la tabla sin ningún tipo de código intermedio.

9.3.4 El objeto Error. Este objeto permite analizar los errores que puedan producirse al realizar una conexión con la fuente de datos, es decir con el *data provider,* aunque no todos. Cada vez que se produce un error y para cada uno de ellos, se crea uno de estos objetos que se almacena en la *colección Errors* del objeto *Connection.*

9.3.5 El objeto Field. Cada objeto *Recordset* contiene una *colección Fields* que representan las columnas del conjunto de registros resultado que integran el citado *Recordset.* Cada uno de los elementos de la colección es un objeto *Field,* las siguientes son algunas de las propiedades y métodos de este objeto:

- ***Name*:** nombre del campo.
- ***Value:*** valor del campo al que representa el objeto en el conjunto de registros, para el actual registro.
- ***Type, Precision, NumericScale:*** característica de tipo del campo.
- ***DefinedSize:*** tamaño del campo.
- ***Attributes, Properties:*** funcionalidad del objeto *Field*.
- ***OriginalValue, UnderlyingValue:*** gestión y resolución de conflictos en actualizaciones de datos con cursor en el cliente *(Batch updates).*
- ***AppendChunk, GetChunk:*** gestión de campos que contienen datos binarios *(Blobs)*.

9.3.6 El objeto Parameter. Representa a un parámetro para un objeto *Command,* sea para una consulta parametrizada o para la ejecución de un procedimiento almacenado que los incluya, por ejemplo: SELECT * FROM ?, donde el signo de interrogación representa un parámetro que podrá ser sustituido por un parámetro en tiempo de ejecución. Cada uno de los parámetros presentes en un comando, es representado por un objeto *Parameter,* que debe definirse y añadirse a la *colección Parameters* al objeto *Command* mediante sus métodos

CreateParameter y *Append.* Estas son algunas de sus propiedades y métodos:

- ***Name*:** nombre del parámetro.
- ***Value*:** valor del parámetro tanto para lectura como escritura.
- ***Type, Precision, NumericScale:*** características de tipo del parámetro.
- ***Size:*** tamaño del parámetro.
- ***Direction:*** indica si el parámetro es de entrada o de retorno.
- ***Atributes:*** funcionalidad del objeto *Field.*
- ***AppendChunk:*** paso de datos binarios.

9.3.7 El objeto Property. Representa una propiedad de un objeto ADO definida por el proveedor de datos. Son lo que se denominan propiedades dinámicas. ADO define dos tipos de propiedades: *incorporadas y dinámicas.* Las primeras son las que se definen con el modelo de objetos, y son accesibles inmediatamente y en todo momento a través del propio objeto, sin que aparezcan como objetos *Properties* del objeto. Las propiedades dinámicas son aquellas que define el proveedor de datos y por tanto no son las mismas para todas las fuentes de datos, ni existen en el modelo propiamente dicho. Estas propiedades extras ofrecen una funcionalidad adicional y se almacenan en la *colección Properties,* en la forma de objetos *Property*. Se puede acceder a las propiedades incorporadas de un *recordset* con la siguiente sintaxis:

```
Mirecordset.NombrePropiedad
```

Para acceder a las propiedades dinámicas se utilizaría la siguiente sintaxis:

```
Mirecordset.Properties(NombrePropiedad)
o
Mirecordset.Properties(IndicePropiedad)
```

Las siguientes son las propiedades de este objeto:

- ***Name:*** nombre de la propiedad.
- ***Value:*** valor de la propiedad.
- ***Type:*** características de tipo del parámetro.
- ***Attributes:*** características de la propiedad.

9.4 LA BIBLIOTECA ADOX

Las *Extensiones ActiveX Data Objects* para objetos de datos,para el Lenguaje de Definición de Datos *(DDL)* y la Seguridad, ADOX, es una extensión de los objetos y del modelo de programación ADO. Incluye objetos que permiten crear y modificar esquemas, así como seguridad. Como se trata de un acercamiento basado en objetos a la manipulación de esquemas, se puede escribir el código que funcione correctamente con diferentes orígenes de datos, independientemente de las diferencias que puedan existir en las sintaxis nativas. ADOX es una biblioteca que acompaña a los objetos ADO del núcleo, es una extensión de la biblioteca ADODB.

Expone objetos adicionales para crear, modificar y eliminar objetos del esquema, como tablas y procedimientos. También incluye objetos de seguridad que permiten mantener usuarios y grupos y conceder y quitar permisos en los objetos.

9.4.1 Modelo de objetos ADOX. La siguiente figura contiene el diagrama que muestra cómo se representan y relacionan estos objetos en ADOX:

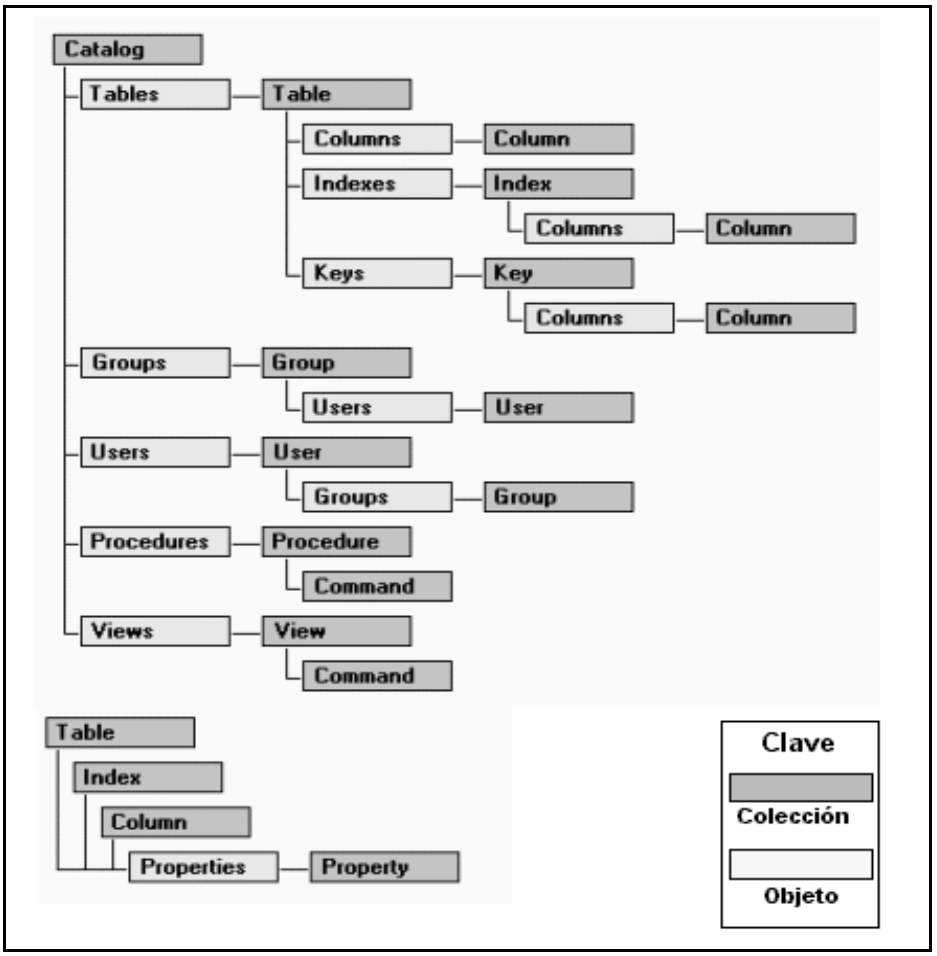

Figura 24. Arquitectura de la biblioteca ADOX

9.4.1.1 Objetos de ADOX. La siguiente tabla contiene el conjunto de objetos de ADOX

Tabla 30. Objetos de la biblioteca ADOX

Objeto	Descripción
Catalog	Contiene colecciones que describen el catálogo de esquema de un origen de datos.
Column	Representa una columna de una tabla, índice o clave.
Group	Representa una cuenta de grupo que tiene permisos de acceso en una base de datos segura.
Index	Representa un índice de una tabla de base de datos.
Key	Representa un campo clave principal, externa o única de una tabla de base de datos.
Procedure	Representa un procedimiento almacenado.
Table	Representa una tabla de base de datos; incluyendo columnas, índices y claves.
User	Representa una cuenta de usuario que tiene permisos de acceso en una base de datos segura.
View	Representa un conjunto de registros filtrados o una tabla virtual.

Cada uno de los objetos *Table, Index* y *Column* dispone además de una colección *Properties* de ADO estándar.

9.4.1.2 Colecciones ADOX. La siguiente tabla contiene el resumen de la colección ADOX.

Tabla 31. Resumen de la colección ADOX

Colección	Descripción
Columns	Contiene todos los objetos Column (Columna).
Groups	Contiene todos los objetos Group (Grupo).
Indexes	Contiene todos los objetos Index (Índice) de una tabla.
Keys	Contiene todos los objetos Key (Clave) de una tabla.
Procedures	Contiene todos los objetos Procedure (Procedimiento) de un catálogo.
Tables	Contiene todos los objetos Table (Tabla).
Users	Contiene todos los objetos User (Usuario) de un catálogo o grupo.
Views	Contiene todos los objetos View (Vista) de un catálogo.

9.5 EVALUACIÓN DE LA METODOLOGÍA

En la evaluación de la metodología se utilizo la plataforma *Windows NT Server 4.0* con el servidor Web *IIS* 4.0, el motor de base de datos *Microsoft SQL Server 7,* ASP, el modelo de programación ADO versión 2.2, con la biblioteca de acceso a datos ADODB y como interfaz cliente de la fuente de datos se uso ODBC.

El siguiente constituye el código del formulario para la inserción de un registro en la base de datos, que el cliente envía al servidor:

```
<html><body>
<form action="insertar.asp" method="post">
  Idcliente <input type="text" name="idcliente">
  Nombre <input type="text"  name="nombre">
  Apellidos <input type="text" name="apellidos">
  Telefono <input type="text" name="telefono">
  Email<input type="text" name="email" size="20">
  <input type="submit" value="enviar">
            <input type="reset" value="Restablecer">
</form>
</body></html>
```

El código del *script* ASP que recibe los datos del formulario y permite la inserción de registros en la base de datos, es el siguiente:

```
<html>
<head><title>Insertar.asp</title></head>
<body>
<% DSN = "aspclientes" ‘La fuente de datos ODBC
tabla = "cliente"   ‘La tabla
‘Se crea el objeto
Set rs=Server.CreateObject("ADODB.Recordset")
Cursor=0  ‘El tipo de cursor
```

```
‘Se abre la conexión mediante el método open
rs.open "SELECT * from "&tabla,DSN,Cursor,2,1

‘Se crea un registo en blanco con el metodo Addnew
rs.AddNew
  rs("idcliente") = Request.Form("idcliente")
  rs("nombrecli") = Request.Form("nombre")
  rs("apellidoscli") = Request.Form("apellidos")
  rs("telefonocli") = Request.Form("telefono")
  rs("emailcli") = Request.Form("email")
rs.Update ‘Se actualiza el registro con los valores pasados

response.write "<body><html>"
response.write "Se ha insertado el registro<br>"
response.write "</body></html>"
rs.close  ‘se cierra el objeto
%>
</body>
</html>
```

9.6 RESULTADO DE LA EVALUACIÓN

9.6.1 Lenguaje. ASP es un marco de trabajo independiente del lenguaje, por lo tanto, se pueden desarrollar las aplicaciones en los lenguajes de *script* soportados (VBScript, JavaScript).

Aprender a utilizar los objetos y los componentes del servidor, así como sus métodos y eventos es muy fácil y no requiere de muchos conocimientos previos, es muy potente y flexible.

Implementar una aplicación de servidor que manipule una fuente de datos con ADO, puede realizarse en *VBScript, Visual C++, Visual J++, Visual InterDev*, o incluso *Perl*. El modelo de objetos de ADODB es relativamente sencillo y con este se pueden implementar de una manera practica y rápida, aplicaciones Web que manipulen fuentes de datos.

Aunque esta capacidad multilenguaje pueda ser bien vista por los desarrolladores, ya que cada cual implementaría el código en el lenguaje que mejor maneje; pero para el sistema donde resida, puede ser una sobrecarga el hecho de tener que llegar a interpretar o compilar, por ejemplo dos lenguajes en una sola pagina.

9.6.2 Soporte a plataformas. Esta metodología solamente puede implementarse en las plataformas *Windows* (9x, NT, 2000) de *Microsoft*, ya que son tecnologías propietarias de esta corporación.

9.6.3 Soporte a bases de datos. Están soportados todos los controladores ODBC y cualquier fuente de datos compatible con OLE DB, lo que le confiere a esta tecnología una gran flexibilidad.

9.6.4 Flexibilidad. ADO es un modelo de objetos que permite una gran flexibilidad al programador, por consiguiente, hay distintas posibilidades para lograr la misma tarea. Por ejemplo, para ejecutar una consulta, es posible usar el método *Execute* del objeto *Command* o bien el del objeto *Connection.*

ADO proporciona un acceso a los datos constante y de alto rendimiento para crear un cliente de base de datos para el usuario o un objeto empresarial del nivel medio con una aplicación, una herramienta, un lenguaje o un navegador Web. Esta tecnología permite a las empresas integrar orígenes de datos distintos, crear soluciones de fácil mantenimiento y utilizar las herramientas, las aplicaciones y los servicios de la plataforma que los desarrolladores prefieran.

10. SERVLETS

10.1 QUE ES UN SERVLET

Un *servlet* es un componente Web basado en java, manejado por un *Container* (Contenedor), para producir contenido dinámico. Como otro componente basado en java, los *servlets* son clases de java independiente de la plataforma, compiladas en el idioma de la maquinaria de java para una plataforma neutral, que pueden ser cargados dinámicamente y ejecutados dentro de un servidor Web que soporte java.

Los contenedores, algunas veces llamados generadores de *servlets*, son extensiones del servidor Web para proveer funcionalidad *servlet*. Los *servlets* interactúan con los clientes a través del paradigma *solicitud / respuesta* implementado por el contenedor de *servlets*.

La tecnología *Servlet* provee a los desarrolladores de un mecanismo consistente y simple para extender la funcionalidad de un servidor Web y el acceso a los actuales sistemas de negocio. Un *servlet* se puede asimilar como un *applet* que corre del lado del servidor; pero, al contrario de los *applets*, los *servlets* no tienen interfaz gráfica de usuario.

Los s*ervlets* es la tecnología elegida, de la plataforma Java, para expandir e incrementar los servidores Web, estos proporcionan una metodología basada en componentes, independiente de la plataforma para crear aplicaciones Web sin las limitaciones de desempeño del modelo CGI y distinto de los mecanismos de extensiones propietarias de servidor (léase ISAPI, NSAPI).

Poseen completo acceso al conjunto de API de Java incluyendo JDBC para el acceso a las bases de datos corporativas. Proporcionan una forma de generar documentos dinámicos que son fáciles de escribir y rápidos en ejecutarse. Los *servlets* también solucionan el problema de hacer la programación del lado del servidor con los API específicos de la plataforma: están desarrollados con el API *Java Servlet*, una extensión estándar de Java.

10.1.1 Que es un contenedor de servlets. El contenedor de *servlet* es una parte de un servidor Web que suministra los servicios de red sobre el cual las solicitudes y las respuestas se establecen, decodifica las solicitudes y formatos de respuestas basadas en MIME. Un contenedor también contiene y maneja los *servlets* a lo largo de su ciclo de vida. Todo contenedor de *servlets* debe soportar las peticiones y las respuestas a través del protocolo HTTP; pero peticiones y/o respuestas basadas a través de protocolos tales como HTTPS (HTTP sobre *SSL*), pueden también tener soporte.

10.2 CARACTERÍSTICAS BÁSICAS

Los *servlets* surgen debido a la limitación existente que existían en los *applets* desarrollados para el Web en dos vertientes distintas: la imposibilidad de acceder a otro servidor que no sea el mismo en el que el *applet* se este ejecutando (generalmente la máquina cliente) y la limitación en el acceso a los servicios de dicha máquina. Por otro lado un *applet* es un programa que se descarga desde el Web y se ejecuta en el navegador y puede resultar relativamente sencillo llegar al código fuente del mismo el cual puede verse comprometido seriamente dejando al descubierto información que puede que no queramos que se vea.

Qué son por tanto los *servlets*, un *servlet* no es más que una aplicación que se ejecuta en un servidor Web quedando a la espera para resolver peticiones efectuadas por los clientes. Mediante los *servlets* se puede tener acceso a otros servidores y acceder a la información que ellos contienen, por ejemplo una base de datos.

Visto lo anterior, los *servlets* se pueden ver como la evolución lógica de los CGI, ya que estos últimos no son más que aplicaciones cuyo cometido es resolver peticiones hechas por los clientes que adolecían de un número de inconvenientes considerables como era el hecho del lenguaje de programación utilizado (generalmente lenguajes interpretados), un rendimiento muy bajo, baja portabilidad, dificultad de comunicación entre CGI, entre otros, que se han visto superados por la aparición de los *servlets.*

Las siguientes son las características básicas de todo *servlet*:

- Son independientes de la plataforma en donde se ejecutan.
- Son muy rápidos (en comparación con los CGI).
- Se pueden comunicar distintos *servlets* entre si.
- Son muy seguros (hacen uso del *Security Manager* de Java)
- Son más eficientes dado que múltiples llamadas sobre el mismo *servlet* no origina la ejecución de distintos procesos sino que crean *threads* de ejecución una vez que ya existe el proceso del *servlet* ejecutándose en la máquina.

10.3 CICLO DE VIDA DE UN SERVLET

El ciclo de vida de un *servlet e*s un proceso que consta de cuatro pasos básicos que se repiten siempre. (Véase la Figura 25), como se indica a continuación:

- El cliente solicita una petición a un servidor mediante una URL.
- El servidor recibe la petición y la procesa, en este caso el servidor detecta que es un *servlet* e inicia la ejecución del mismo sino se encuentra ya en ejecución alguna instancia del mismo.
- El *servlet* procesa la petición y la retorna al cliente.
- El *servlet* creado solo se destruye cuando el contenedor de *servlets* se cierra o bien cuando hay un fallo en el sistema.

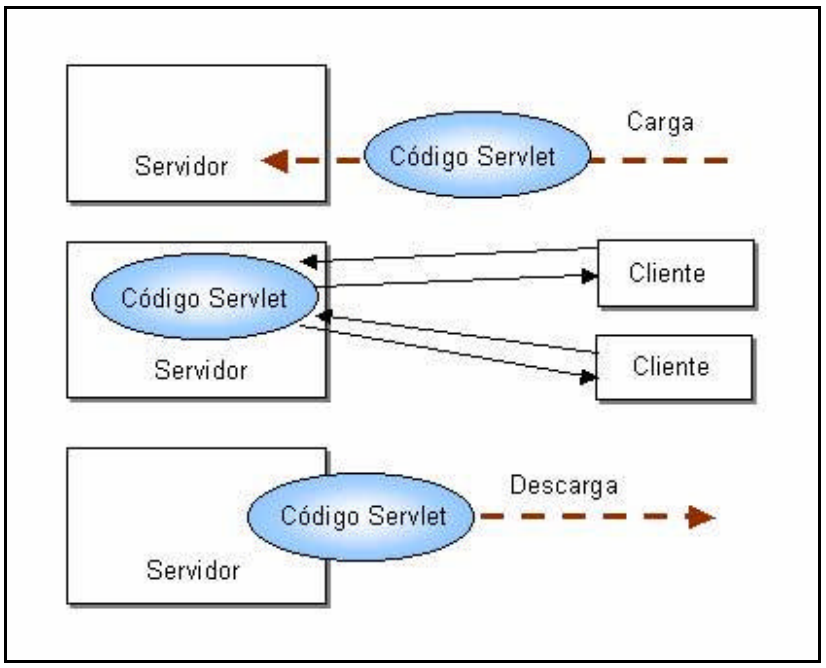

Figura 25. Ciclo de vida de un Servlet

10.4 ARQUITECTURA DE LOS SERVLETS

El API que se utiliza para el desarrollo de los *servlets*, contiene dos *packages* (paquetes) distintos: *javax.servlet* y *javax.servlet.http* (en la Figura 26, está representada la jerarquía de herencia. En dicha figura se representan las ***clases*** con letra normal y las ***interfaces*** con *cursiva*). Todas las clases e interfaces que hay que utilizar en la programación de *servlets* están en estos dos *packages*.

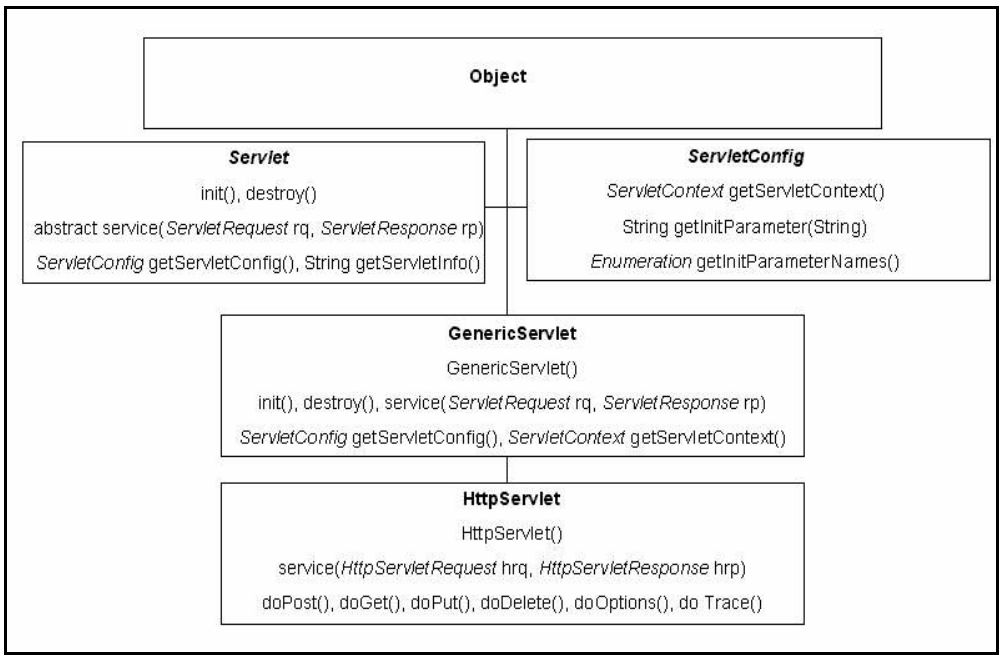

Figura 26. Jerarquía de herencia de las principales clases para crear servlets

Se puede observar que la clase ***GenericServlet*** es una clase abstracta que implementa dos interfaces, ***Servlet*** y ***ServletConfig***, siendo fundamental la interfaz *Servlet* ya que en ella se declaran los métodos más importantes para la existencia de un *servlet* (ciclo de vida) como son:

- init() que se ejecuta sólo al arrancar el *servlet.*
- destroy() que se ejecuta cuando va a ser destruido, y
- service() que se ejecutará cada vez que el *servlet* deba atender una solicitud de servicio.

Todos los *servlets* implementan esta interfaz directamente, por medio de la extensión de la clase que la implementa, ***HttpServlet***. Esta interfaz está provista de métodos que manipulan a los *servlets* y su comunicación con los clientes.

Cuando un *servlet* es llamado desde un cliente, recibe dos objetos: uno es un *ServletRequest* y el otro es un *ServletResponse*. La clase *ServletRequest* encapsula la comunicación desde el cliente al servidor, mientras que la clase *ServletResponse* encapsula la comunicación desde *servlet* hacia el cliente, a estas dos clases se les conoce también como interfaz.

La interfaz *ServletRequest* permite al *servlet* acceder a información como, los nombres de parámetros pasados por el cliente, el protocolo usado por el cliente, y los nombres de los *host* remotos que hacen la solicitud y el servidor que la recibe. Esto es proporcionado al *servlet* por medio del acceso al flujo de entrada, *ServletInputStream,* mediante el cual, el *servlet* obtiene la información de los clientes que están usando los protocolos de aplicación de HTTP como son *POST, PUT*; etc. Las subclases que extienden la interfaz *ServletRequest* permiten al *servlet* recibir más datos específicos del protocolo.

La interfaz *ServletResponse* proporciona al *servlet* los métodos para contestarle al cliente. Permite al *servlet* configurar la forma de salida de los datos para el cliente (tipo MIME de respuesta) y provee un flujo de salida, *ServletOutputStream,* que proporciona al *servlet* un medio por el cual responder a las solicitudes de los clientes. Las subclases de *ServletResponse* le dan mas capacidad al *servlet* para

responder.

Las clases e interfaces descritas conforman a un *servlet* básico. Pero existen métodos adicionales que provee el API con la capacidad para controlar sesiones o múltiples conexiones, entre muchas más aplicaciones. Algunas otras interfaces, cuyo papel se resume a continuación son:

1. La interfaz *ServletContext* permite a los *servlets* acceder a información sobre el entorno en que se están ejecutando.
2. La interfaz *ServletConfig* define métodos que permiten pasar al *servlet* información sobre sus parámetros de iniciación.
3. La interfaz *ServletRequest* permite al método *service()* de *GenericServlet* obtener información sobre una petición de servicio recibida de un cliente. Algunos de los datos proporcionados por *GenericServlet* son los nombres y valores de los parámetros enviados por el formulario HTML y un *inputstream*.
4. La interfaz *ServletResponse* permite al método *service()* de *GenericServlet* enviar su respuesta al cliente que ha solicitado el servicio. Esta interfaz dispone de métodos para obtener un *output stream* o un *writer* con los que enviar al cliente datos binarios o caracteres, respectivamente.
5. La interfaz *HttpServletRequest* deriva de *ServletRequest*. Esta interfaz permite a los métodos *service()*, *doPost()*, *doGet()*, *doPut()*, *doDelete()*, *doOptions()*, *doTrace()* de la clase *HttpServlet* recibir una petición de servicio HTTP. Esta interfaz permite obtener información de la cabecera de la petición de servicio HTTP.

6. La interfaz *HttpServletResponse* extiende *ServletResponse*. A través de esta interfaz los métodos de *HttpServlet* envían información a los clientes que les han pedido algún servicio.

10.4.1 La clase HttpServlet. Los *servlets* que utilizan el protocolo HTTP son los más comunes. Utilizando el protocolo HTTP los clientes (navegadores) y los servidores puedan comunicarse entre sí, mediante la utilización de una serie de métodos como son *GET*, *HEAD*, *POST*, *PUT*, *DELETE*, *TRACE*, *CONNECT y OPTIONS*. Estos métodos son los que van dentro de la clase *HttpServlet.*

La clase abstracta *javax.servlet.http.HttpServlet* implementa la interfase *javax.servlet.Servlet* e incluye un numero importante de funciones adicionales. La forma más sencilla de escribir un *servlet* HTTP es heredando de *HttpServlet* que es también una clase *abstract*, de modo que es necesario definir una clase que derive de ella y redefinir en la clase derivada al menos uno de sus métodos, tales como doGet() o doPost().

La clase *HttpServlet* proporciona una implementación del método service() en la que distingue qué método se ha utilizado en la petición, llamando seguidamente al método *adecuado (doGet(), doHead(), doDelete(), doOptions(), doPost() y doTrace())*. Estos métodos e corresponden con los métodos HTTP anteriormente citados.

Así pues, la clase *HttpServlet* no define el método service() como *abstract*, sino como *protected*, al igual que los métodos *init(), destroy(), doGet(), doPost()*; etc.,

de forma que ya no es necesario escribir una implementación de *service()* en un *servlet* que herede de dicha clase.

La clase *HttpServlet* sabe qué métodos han sido redefinidos en una sub-clase, de forma que puede comunicar al cliente qué tipos de métodos soporta el *servlet* en cuestión. Así, si en una clase *servlet* sólo se ha redefinido el método *doPost()* y el cliente realiza una petición de tipo HTTP GET el servidor lanzará automáticamente un mensaje de error similar al siguiente:

```
501 Method GET Not Supported
```

Donde el número que aparece antes del mensaje es un código empleado por los servidores HTTP para indicar su estado actual.

El siguiente ejemplo ilustra la estructura de un *servlet* básico:

```
Import javax.servlet.*;
import javax.servlet.http.*;

public class EjemploServlet extends HttpServlet

// Este método se ejecuta una única vez (al ser inicializado el
//servlet). Se suelen inicializar variables y realizar
//operaciones costosas en tiempo de ejecución (abrir ficheros,
//bases de datos, etc)
{ public void init(ServletConfig config) throws ServletException
   // Llamada al método init() de la superclase (GenericServlet)
   //  Así se asegura una correcta inicialización del servlet
```

```
{ super.init (config);
       Resto del código de init
} // fin del método init()
//El método destroy() es llamado por el servidor Web al
//"apagarse" (o el contenedor ). Sirve para proporcionar una
//correcta esconexión de una base de datos, cerrar ficheros
//abiertos; etc.
public void destroy()
{
  Código del destroy
} // fin del método destroy()

//El método dopost() es llamado mediante un HTTP POST. Este
//método se llama automáticamente al ejecutar un formulario HTML
public void doPost (HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException
{
  Código del doPost generalmente:
   - Obtención de variables
   - Tratamiento de datos
   - Devolver Pagina HTMl
} // fin del método doPost()

// Función que permite al servidor Web obtener una pequeña
// descripción del servlet, qué cometido tiene, nombre del autor,
// comentarios adicionales; etc.
public String getServletInfo()
{
    Método adicional de información
} // fin del método getServletInfo()
}
```

Descripción:

- La clase ***EjemploServlet*** hereda de la clase ***HttpServlet***, que a su vez hereda de ***GenericServlet*** por esta razón el método ***service()*** no se redefine. La forma más sencilla de crear un *servlet*, es heredar de la clase ***HttpServlet***, de esta forma se está identificando la clase como un *servlet* que se conectará con un servidor HTTP.
- El método ***init()*** es el primero en ser ejecutado. Sólo es ejecutado *la primera vez* que el *servlet* es llamado. Se llama al método ***init()*** de la superclase ***GenericServlet*** a fin de que la inicialización sea completa y correcta. La interfaz ***ServletConfig*** proporciona la información que necesita el *servlet* para inicializarse.
- El método ***destroy()*** proporciona una descarga correcta del *servlet* de la memoria, de forma que no queden recursos ocupados indebidamente, o haya conflictos entre recursos en uso. Tareas propias de este método son por ejemplo el cierre de las conexiones con otros computadores o con bases de datos.
- Si el formulario HTML utiliza el método *HTTP POST* para la transmisión de sus datos, habrá que redefinir el método ***doPost(),*** que se encarga de procesar la respuesta y que tiene como argumentos el objeto que contiene la petición y el que contiene la respuesta (pertenecientes a las clases ***HttpServletRequest*** y ***HttpServletResponse***, respectivamente).

Este método será llamado tras la inicialización del ***servlet*** (en caso de que no haya sido previamente inicializado), y contendrá el núcleo del código del ***servlet*** (llamadas a otros ***servlets***, llamadas a otros métodos, entre otros).

- El método ***getServletInfo()*** proporciona datos acerca del ***servlet*** (autor, fecha de creación o funcionamiento) al servidor Web. No es en ningún caso obligatoria su utilización aunque puede ser interesante cuando se tienen muchos ***servlets*** funcionando en un mismo servidor y puede resultar compleja la identificación de los mismos.

10.5 APLICACIÓN DE LOS SERVLETS

Algunas de las aplicaciones de *servlets* incluyen:

- Procesamiento de datos enviados con HTTPS a partir de una forma HTML, incluyendo la orden de compra o datos de una tarjeta de crédito. Un *servlet* como este podría ser parte de una orden de entrada al sistema de procesamiento de datos, haciendo la respectiva actualización en la base de datos, y quizá hasta un sistema de pago en línea.
- Interacción múltiple entre personas. Un *servlet* puede manipular múltiples peticiones al mismo tiempo; podrían sincronizar dichas solicitudes de sistemas que dan soporte, por ejemplo conferencias en línea.
- Redireccionamiento de peticiones. Los *servlets* pueden reenviar las peticiones a otros servidores y *servlets,* esto permite que al ser usados se

balancee la carga entre varios servidores que atiendan la misma tarea, según el tipo de proceso que deban atender.

Algunas de las aplicaciones más específicas implementadas con *servlets* son:

- Típicos sistemas *middleware*. Para consultar las bases de batos, los s*ervlets* pueden utilizar JDBC, lo que les permite extraer información de cualquier sistema de base de datos.
- Automatización de un sistema de recepción y publicación de información. Se pudiera implementar una estación meteorológica simple, que permitiese acceso a su información mediante una página Web. Por un lado se tendría un *servlet* que recolectaría la información de los diversos tipos de sensores y la almacenaría en bases de datos, y por otro lado, un *servlet* que se encargaría de presentar esta información en función de las peticiones del cliente basándose en estas mismas bases de datos.
- Control de la recepción de correo electrónico, y de sistemas de noticias, *chats*; etc. Conviene recordar que Java está especialmente indicado para la programación utilizando los protocolos TCP/IP.
- Dado que puede manejar múltiples peticiones en forma concurrente, es posible implementar aplicaciones de colaboración, como por ejemplo una aplicación de videoconferencia.

10.6 ACCESO A BASES DE DATOS A TRAVÉS DE JDBC

Una de las tareas más importantes y más frecuentemente realizadas por los *servlets* es la conexión a bases de datos mediante JDBC. Esto es debido a que los *servlets* son un componente ideal para hacer las funciones de *middleware* en un sistema con una arquitectura de tres capas, como se muestra en la siguiente figura.

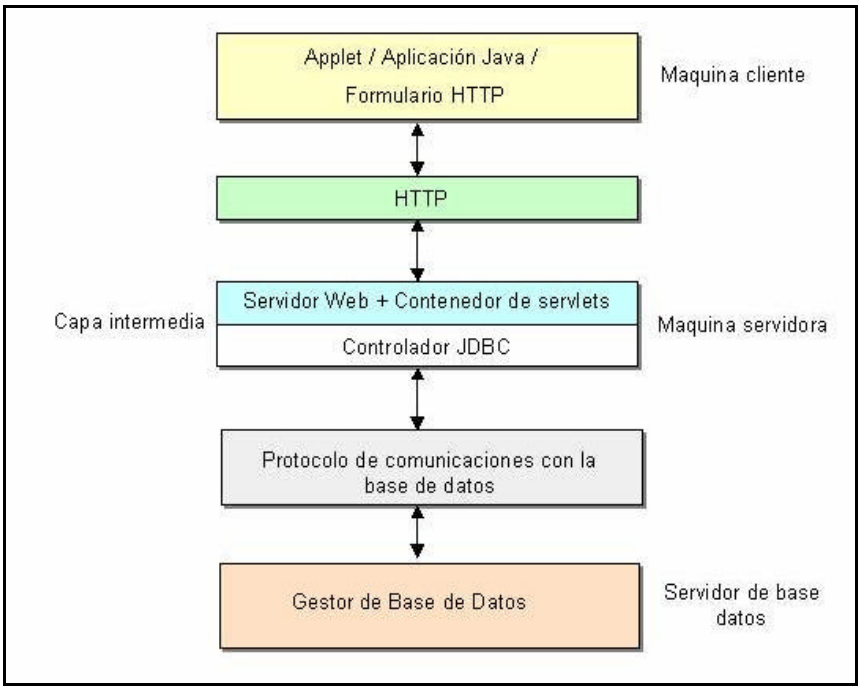

Figura 27. Arquitectura Cliente-Servidor de tres capas

Este modelo presenta la ventaja de que el nivel intermedio mantiene en todo momento el control del tipo de operaciones que se realizan contra la base de datos, y además, está la ventaja adicional de que los controladores JDBC no tienen que residir en la máquina cliente, lo cual libera al usuario de la instalación

de cualquier tipo de controlador.

10.7 EVALUACIÓN DE LA METODOLOGÍA

Para la evaluación de esta metodología se utilizo la plataforma Windows 98, el servidor de aplicaciones Java, *Allaire JRun* versión 3.0 la base de datos MySQL versión 3.23.33 para Windows, como *driver* se uso el puente JDBC-ODBC.

El siguiente código HTML pertenece al formulario que solicita la ejecución del servlet para realizar inserciones:

```
<html>
<body>
<!--se solicita la ejecución de la clase insertar.class, que recibirá los
datos que se le pasan para su procesamiento-->
<form name="form1" method="post"
action="http://192.168.0.2:8100/servlet/insertar">
Insertar Campos
Clave: <input type="text" name="clave">
Nombre: <input type="text" name="nombre">
Apellidos <input type="text" name="apellidos">
Profesion:
<select name="profesion">
    <option>Ing. Sistemas</option>
    <option>Ing. Mecanica</option>
    <option>Ing. Electronica</option>
    <option>Ing. Industrial</option>
    <option>Ing. Electrica</option>
</select>
Direccion: <input type="text" name="direccion">
Comentarios:<textarea name="comentarios" cols="40" rows="5">
```

```
</textarea>
<input type="submit" >
<input type="reset" >
</form>
</body>
```

El siguiente código pertenece al archivo Java *insertar.java* que debe ser compilado para generar la clase *insertar.class*, esta clase es ejecutada en el contenedor de *servlets* en respuesta a la solicitud que el cliente hace a través del formulario:

```
// fichero insertar.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class insertar extends HttpServlet {
// Declaración de variables miembro
private String clave = null;
private String nombre = null;
private String apellidos = null;
private String profesion = null;
private String direccion = null;
private String comentarios = null;
// Referencia a un objeto de la interface java.sql.Connection
Connection conn = null;
public void init (ServletConfig config) throws ServletException
{
  super.init(config);
  String dsn = new String("jdbc:odbc:graduados");
  // Carga del Driver del puente JDBC-ODBC
  try
  {
     Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  }
  catch(ClassNotFoundException ex)
```

```
  { System.out.println("Error al cargar el driver");
    System.out.println(ex.getMessage());
  }
   // Establecimiento de la conexión con la base de datos
  try
  { conn = DriverManager.getConnection(dsn, "", "");
  }
  catch (SQLException sqlEx)
  { System.out.println("Se ha producido un error al" +
     " establecer la conexión con: " + dsn);
     System.out.println(sqlEx.getMessage());
  }
     System.out.println("Iniciando ServletOpinion (version BD)...");
} // fin del método init()

public void destroy ()
{
   super.destroy();
   System.out.println("Cerrando conexion...");
   try
   { conn.close();
   }
   catch(SQLException ex)
   { System.out.println("No se pudo cerrar la conexion");
     System.out.println(ex.getMessage());
   }
}
// fin del método destroy()
public void doPost (HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException
{  boolean hayError = false;
     // adquisición de los valores del formulario
     if(req.getParameter("clave")!=null)
        clave = req.getParameter("clave");
     else
        hayError=true;
     if(req.getParameter("nombre")!=null)
        nombre = req.getParameter("nombre");
```

```
      else
       hayError=true;
     if(req.getParameter("apellidos")!=null)
         apellidos = req.getParameter("apellidos");
     else
        hayError = true;
     if(req.getParameter("profesion")!=null)
         profesion = req.getParameter("profesion");
     else
        hayError = true;
     if(req.getParameter("direccion")!=null)
        direccion = req.getParameter("direccion");
     else
        hayError = true;
     if(req.getParameter("comentarios")!=null)
        comentarios = req.getParameter("comentarios");
     else
        hayError = true;
     // Mandar al usuario los valores adquiridos
     // (Si no se ha producido error)
     if(!hayError)
     {
       if (actualizarBaseDeDatos() == 0)
          devolverPaginaHTML(resp);
       else
         resp.sendError(500, "Se ha producido un error"+
         " al actualizar la base de datos");
     }
     else
      resp.sendError(500, "Se ha producido un error"+
      " en la adquisición de parámetros");
} // fin doPost()
public int actualizarBaseDeDatos()
{ Statement stmt=null;
  int numeroFilasActualizadas=0;
  // Ejecución del query de actualización de la base de datos
  try
  {stmt = conn.createStatement();
```

```
    numeroFilasActualizadas = stmt.executeUpdate("INSERT INTO"+
    " graduados VALUES"+"('"+clave+"' , '"+nombre+"',
    '"+apellidos+"','"+profesion+"','"+direccion+"',
    '"+comentarios+"')");
    if(numeroFilasActualizadas!=1) return -1;
  }
   catch (SQLException sql)
  {System.out.println("Se produjo un error creando Statement");
    System.out.println(sql.getMessage());
    return -2;
  }
  finally
  {
    // Se cierra el Statement
    if(stmt!=null)
    {
      try
      {stmt.close();
      }
      catch(SQLException e)
      { System.out.println("Error cerrando Statement");
        System.out.println(e.getMessage());
        return -3;
      }
    }
     return 0;
  } // fin finally
} // fin método actualizarBaseDeDatos()
public void devolverPaginaHTML(HttpServletResponse resp)
{
 /PrintWriter para escribir (sólo para mandar texto)
  PrintWriter out=null;
  try
  { out=resp.getWriter();
  }
  catch (IOException io)
  {System.out.println("Se ha producido una excepcion");
  }
```

```
  // Se establece el tipo de contenido MIME de la respuesta
  resp.setContentType("text/html");
  // Se mandan los valores
  out.println("<html>");
  out.println("<head>");
  out.println("<title>Valores recogidos en el formulario
  </title>");
  out.println("</head>");
  out.println("<body>");
  out.println("<b><h2><center>Valores recogidos del");
  out.println("formulario:</center> </h2><br>");
  out.println("<h3><font face=\"Arial, Helvetica, sans-serif\">Clave:
"+clave+"<br>");
  out.println("Nombre:  "+nombre+"<br>");
  out.println("Apellido:  "+apellidos+"<br>");
  out.println("Profesion:  "+profesion+"<br>");
  out.println("Direccion:  "+direccion+"<br>");
  out.println("Comentarios:  "+comentarios+"<br></font><h3>");
  out.println("</b><a href=\"http://192.168.0.2/servlet-
jdbc/insertar.htm\">Regresar</a>");
  out.println("</body>");
out.println("</html>");
// Se fuerza la descarga del buffer y se cierra el PrintWriter
out.flush();
out.close();
} // fin de devolverPaginaHTML()
// Función que permite al servidor Web obtener una
// pequeña descripción del servlet, qué
// cometido tiene, nombre del autor, comentarios adicionales; etc.
public String getServletInfo() {
return "Este servlet lee los datos de un formulario "+
"y los introduce en una base da datos";
} // fin de getServletInfo()
} // fin de la clase servletOpinion2
```

10.8 RESULTADOS DE LA EVALUACIÓN

10.8.1 Lenguaje. Para manejar la clase *Servlet* en toda su extensión se necesita tener un buen conocimiento previo del lenguaje. Por ser creados en Java, hereda las bondades y desventajas de este poderoso lenguaje. Sin embargo se pueden enunciar algunas características que se divisaron al momento de digitar el código:

- Comparado con los otros lenguajes de programación evaluados, conocer y aprender java, necesita mas tiempo.
- El código de estas aplicaciones, resulta más extenso y tedioso codificar, por ejemplo el contenido para producir las respuestas HTML que debe ser generada, para ser enviada al cliente. Hay que llevar un control estricto de los errores en tiempo de ejecución.
- El código debe compilarse cada vez que se necesite modificar la aplicación.

Es recomendable utilizar JSP para generar las aplicaciones, ya que no hay que compilar las aplicaciones con cada cambio en el código, además de poder embeber la lógica de aplicación en las paginas HTML.

Por esta razón durante la creación de las aplicaciones, se utilizo la herramienta *Codecharge,* esta herramienta no implementa clases o archivos Java, sino que crea paginas JSP codificadas de una manera óptima.

10.8.2 Soporte a plataformas. Se hereda la característica multiplataforma del

lenguaje Java. Las clases pueden ser ejecutadas por cualquier servidor Web que soporte la ejecución de *servlets,* o cualquier contenedor, en cualquier plataforma que soporte Java.

10.8.3 Soporte a bases de datos. Están soportadas las fuentes de datos ODBC, y todos los gestores de bases de datos que posean controladores JDBC.

10.8.4 Eficiencia. Con CGI tradicional, se arranca un nuevo proceso para cada solicitud HTTP. Si el programa CGI hace una operación relativamente rápida, la sobrecarga del proceso de arrancada puede dominar el tiempo de ejecución. Con los *servlets*, la *Máquina Virtual Java* permanece ejecutándose, y cada petición es manejada por un hilo Java de peso ligero, no un pesado proceso del sistema operativo.

De forma similar, en CGI tradicional, si hay *n* peticiones simultáneas para el mismo programa CGI, el código de este problema se cargará *n* veces en memoria. Con los *servlets*, hay *n threads* pero sólo una copia de la clase *Servlet* ejecutándose.

11. METODOLOGÍA IDC (Internet Database Conector)

Literalmente IDC significa: Conector de bases de datos de Internet. Es una técnica desarrollada por *Microsoft*, y usada por los servidores Web que produce *(IIS y PWS)*. Esta permite enviar consultas a cualquier base de datos cuya interfaz nativa sea reconocida en una fuente de datos ODBC, y por consiguiente recibir respuestas como resultado de tal consulta.

Sus ambientes de trabajo son las plataformas de sistemas operativos *Microsoft Windows* (NT y 9x), sin que esto le imponga ninguna limitación al verse involucrada en la participación de cualquier consulta utilizando enlaces con otras plataformas de sistema operativos. Con el IDC se puede acceder a bases de datos que utilicen SQL como lenguaje de interrogación, como SQL Server, Access 97/2000, Oracle entre otros, y siempre utilizando la interfaz nativa con que vienen provistos los motores de bases de datos.

Es una alternativa para tener en cuenta ya que es muy practica y versátil. En el momento de su aparición vino a competir / sustituir a la técnica de acceso a datos a través de la tecnología CGI.

11.1 CARACTERÍSTICAS DE LA METODOLOGÍA

Dentro de las características relevantes se puede decir que se destaca su naturaleza. Esta técnica utiliza scripts para definir las consultas, los cuales se alojan del lado del servidor y son interpretados en tiempo de ejecución. Es una metodología totalmente transparente, tanto para el programador como para el usuario final. Es decir, no es necesario conocer de antemano la forma como dicha metodología manipula los datos, como se complementa con la parte de comunicaciones, o de cómo se conecta o desconecta de una fuente de datos ODBC. Conceptualmente IDC, funciona de la siguiente manera:

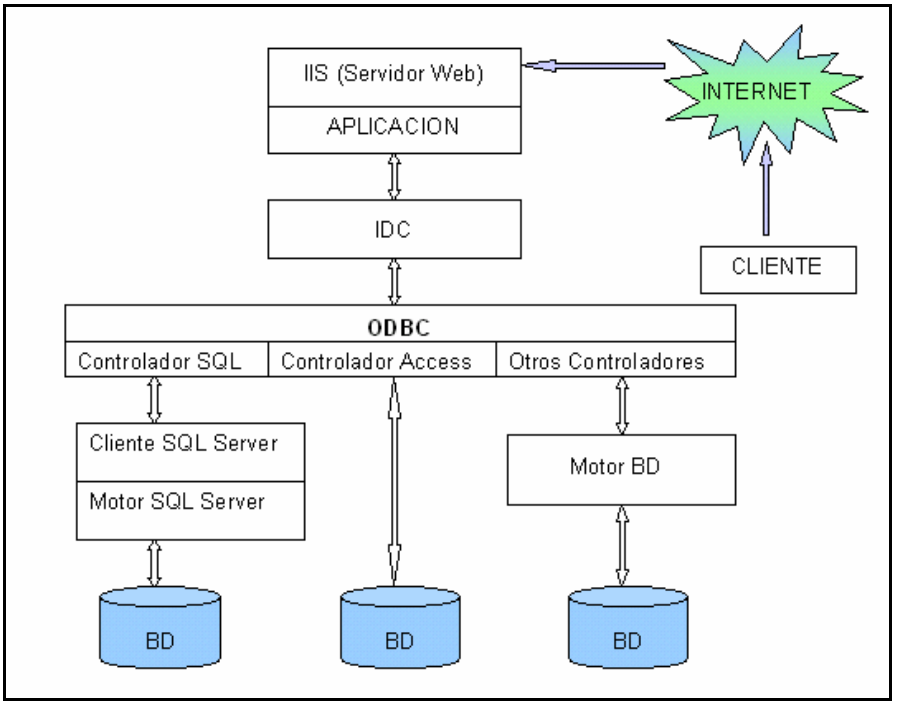

Figura 28. Modelo conceptual de IDC

11.2 LÓGICA DE PROGRAMACIÓN PARA METODOLOGÍA IDC

Para utilizar el Conector de bases de datos de Internet se deben instalar las interfaces nativas de la o las bases de datos a utilizar, lo cual permita que sean reconocidas en una fuente de datos ODBC. Después tiene que agregar instrucciones de consulta a bases de datos en un archivo de Conector de bases de datos de Internet IDC. Finalmente, tiene que crear un archivo de extensión de HTML *(.htx)* que contenga las etiquetas HTML que den formato a los datos devueltos.

El archivo HTX es el que se devuelve al navegador. La consulta que se va a ejecutar, debe ser una sentencia o conjunto de comandos SQL.

11.3 FUNCIONAMIENTO DE IDC

El proceso que sigue una aplicación Web (esto porque en realidad es el fin de cualquier metodología para acceder bases de datos a través de sitios Web), basada en DC desde el momento en que un cliente hace una solicitud de datos hasta que le es enviada una respuesta de parte del servidor Web se realiza en 6 pasos básicos. (Véase la Figura 29).

Paso 1. El cliente reclama los datos, normalmente esta operación se hace a través de un formulario contenido en cualquier pagina html de una sede Web, y mediante una dirección URL.

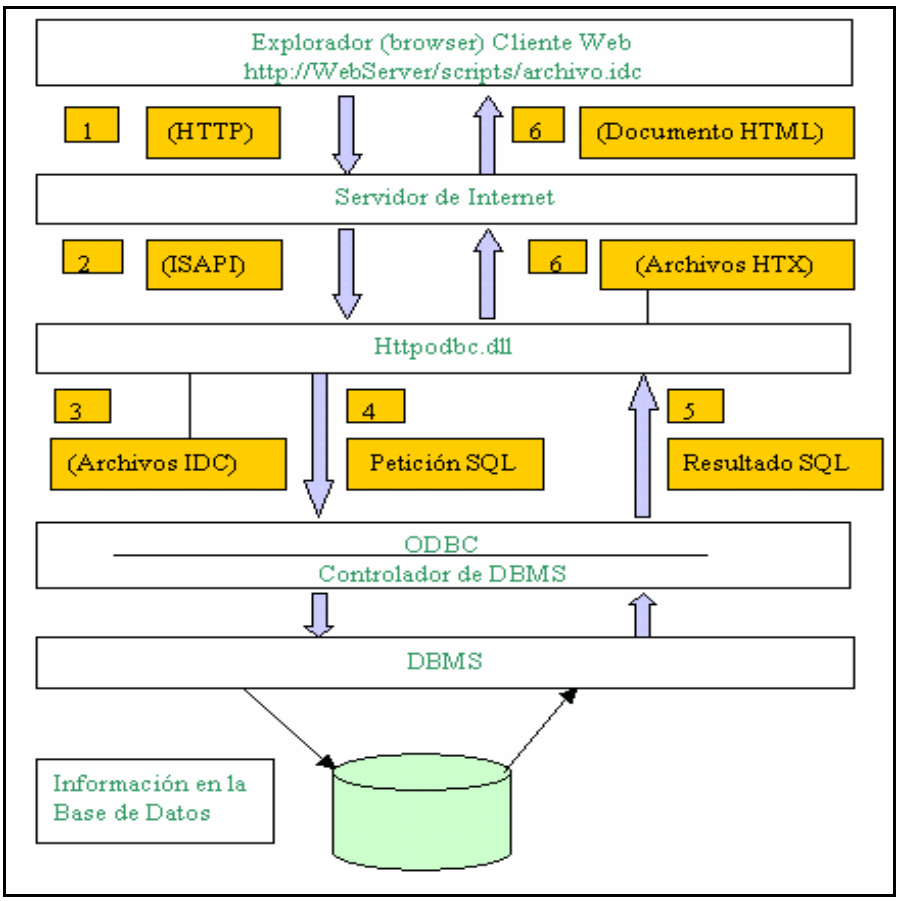

Figura 29. Pasos de funcionamiento IDC

Paso 2. La petición HTTP llega en este caso a IIS, que mediante ISAPI se conecta a la extensión del servidor para procesar consultas ODBC, residente en la librería HTTPODBC.DLL. El servidor Web carga HTTPODBC.DLL y le suministra la información restante de la dirección URL.

Paso 3. El archivo HTTPODBC.DLL se conecta con el controlador ODBC especificado en el archivo IDC, es en este instante cuando se pasan los parámetros contenidos en el archivo IDC para que sean procesados.

Paso 4. El controlador ODBC envía la consulta al gestor de base de datos, que la procesa y devuelve los resultados al controlador ODBC que a su vez vuelve a transmitirlo al conector de bases de datos de Internet HTTPODBC.DLL.

Paso 5. El gestor mediante su modulo IDC incorporado, fusiona los datos obtenidos con el archivo de extensiones HTML y genera un documento HTML puro. Los archivos de extensiones HTML contienen etiquetas especiales de HTML utilizadas por IDC para controlar dónde y cómo se combinan los datos devueltos por la consulta SQL.

Paso 6. El documento generado en HTML puro, es devuelto al cliente.

11.4 ARCHIVOS CONSTITUYENTES DE UNA APLICACIÓN IDC

Todo el código necesario en la metodología IDC se halla almacenado en dos archivos de texto, que se almacenan en el directorio para *scripts* de la sede Web, en la que se ubica la aplicación. IDC utiliza dos tipos de archivos fundamentales para controlar cómo se tiene acceso a las bases de datos y cómo se generan las páginas Web de salida o resultados, en la implementación de soluciones utilizando esta metodología. Estos archivos son:

a) El archivo de Conector de bases de datos de Internet (.idc)

b) El archivo de extensión HTML (.htx)

Los archivos IDC y HTX se complementan entre sí y siempre se deben manejar como un par, mientras que el archivo IDC ejecuta una consulta sobre una base de datos en el servidor Web, el archivo HTX utiliza extensiones HTML para dar formato al conjunto de resultados en HTML arrojados en la consulta. A continuación se describirán estos archivos y su filosofía de trabajo.

11.5 ARCHIVO INTERNET DATABASE CONECTOR (IDC)

El archivo IDC contiene la información necesaria para enviar consultas a una base de datos soportada o configurada en una conexión ODBC. Toda la consulta estará elaborada en sentencias de comandos SQL las cuales manipularan los datos contenidos en las tablas señaladas en dicha consulta. Los archivos de consulta IDC se pueden generar con cualquier editor de texto y deben grabarse con la extensión **idc**, esta es la extensión por defecto para esta clase de archivos.

La siguiente codificación es la vista de la estructura básica de una consulta IDC:

```
Datasource: DB_NEPTUNO
Username:
Password:
Template: inserta.htx
SQLStatement: INSERT INTO CLIENTES
+(idcliente,nombrecompañía,dirección,ciudad) VALUES
+ ('%T1%','%T2%','%T3%','%T4%');
```

Donde las palabras datasource, username, password, template y sqlstament son campos utilizados por IDC, para lograr distintas acciones como se observara en las secciones siguientes.

11.5.1 Contenido del archivo internet database conector (idc).

Esta metodología posee una serie de tres clases de comandos o campos de encabezado que son de distinta naturaleza entre si, a saber:

- **Campos de carácter requerido (obligatorio).**
- **Campos de carácter opcional.**
- **Campos opcionales avanzados de ODBC.**

Las siguientes tablas relacionan los campos que se pueden especificar en los archivos IDC, así como sus respectivas definiciones. Los parámetros o las variables del servidor pueden encontrarse en cualquier parte de los archivos IDC.

Tabla 32. Campos requeridos

Campo	Descripción
Datasource	Nombre que se corresponde con el Nombre de origen de datos (DSN) del sistema ODBC que haya creado con el Administrador de ODBC.
Template	Nombre del archivo de extensión de HTML que da formato a los datos devueltos por esta consulta. Por convención, estos archivos utilizan la extensión htx.

Campo	Descripción
SQLStatement	Instrucción SQL que se va a ejecutar. La instrucción SQL puede contener valores de parámetros, que deben aparecer entre caracteres de porcentaje (%) en el cliente. La instrucción SQL puede ocupar varias líneas en el archivo IDC. A continuación del campo SQLStatement, las líneas que comiencen con un signo más (+) se consideran parte del campo SQLStatement. En un mismo archivo puede haber varias instrucciones SQL.

Tabla 33. Campos opcionales

Campo	Descripción
DefaultParameters = parámetro=valor [, parámetro=valor] […]	Valores de los parámetros, si hay alguno, que se vaya a utilizar en el archivo Conector de bases de datos de Internet si no los especifica el cliente.
Expires	Número de segundos que hay que esperar antes de actualizar una página de resultados almacenados en memoria caché. Si una petición posterior es idéntica, se devolverá la página de la memoria caché sin tener acceso a la base de datos. El campo Expires es útil cuando desee forzar una nueva consulta en la base de datos después de un cierto periodo de tiempo. De forma predeterminada, IDC no almacena en memoria caché las páginas de resultados; Sólo las almacena en memoria caché cuando se utiliza el campo Expires.
MaxFieldSize	Tamaño máximo de búfer asignado al campo por IDC. Se truncarán los caracteres siguientes. Este parámetro sólo se aplica a los campos devueltos por la base de datos que sobrepasen los 8192 bytes. El valor predeterminado es 8192 bytes.

Campo	Descripción
MaxRecords	Número máximo de registros que IDC devolverá desde cualquier consulta. MaxRecords no tiene un valor predeterminado, lo que significa que una consulta puede devolver hasta 4000 millones de registros. Establezca este valor para limitar el número de registros devueltos.
RequiredParameters	Nombres de los parámetros, si hay alguno, que HTTPodbc.dll comprueba que se reciben del cliente; de lo contrario, se devuelve un error. Los nombres de los parámetros están separados por comas.
Translationfile	Ruta del archivo que asigna caracteres no ingleses (como à, ô o é) de forma que los navegadores puedan presentarlos correctamente en formato HTML. Si el archivo de traducción no está en el mismo directorio que el archivo .idc, debe escribir la ruta completa al archivo de traducción. Sintaxis: Translationfile: C:\directorio\archivo. Utilice el campo Translationfile si va a publicar una base de datos en un idioma diferente del inglés. El archivo de traducción es un archivo de texto que asigna cada carácter especial con el formato siguiente: valor=cadena<CR>, donde valor es un carácter internacional y cadena es el correspondiente código HTML.
Username	Nombre de usuario válido en el origen de datos suministrado en el campo Datasource. Nota. Si utiliza Microsoft SQL Server con la opción de seguridad integrada, se pasan por alto los campos de nombre de usuario y contraseña del archivo IDC. El inicio de sesión en SQL Server se realiza con las credenciales del usuario Web. Si la petición proviene de un usuario anónimo, el nombre del usuario y la contraseña están determinados por la configuración del usuario anónimo en el Administrador de servicios Internet (de forma predeterminada, IUSR_NombreDeEquipo). Si la petición del cliente contuviera credenciales de inicio de sesión, se utilizarían el nombre de usuario y la contraseña suministrados por el usuario final para iniciar la sesión en SQL Server.

Campo	Descripción
Password	Contraseña correspondiente al nombre de usuario. Si la contraseña es nula, se puede omitir este campo.
ODBCConnection	Inserte este campo con el valor Pool para agregar la conexión al conjunto de conexiones, que mantiene abierta la conexión con la base de datos para posibles futuras consultas. IDC envía los datos a través de una conexión libre del conjunto en una ejecución posterior de un archivo IDC que contenga los mismos valores de Datasource, Username y Password. Establezca esta opción para mejorar el rendimiento al utilizar Conector de bases de datos de Internet. Además, existe la opción nonpool, que especifica que la conexión del archivo IDC en el que se ha establecido esta opción no se debe tomar del conjunto de conexiones. Asigne a este campo el valor NoPool para administrar la memoria caché de conexiones con mayor precisión. Además, si hay algún límite en cuanto al número de conexiones actuales, no le interesa que el conjunto de conexiones monopolice todas las conexiones; de lo contrario, nadie más podría conectar con SQL Server.Nota. Para definir el valor predeterminado de la agrupación de conexiones, debe definir como 1 la entrada del Registro PoolIDCConnections.
Content-Type	Tipo MIME válido que describe lo que se va a devolver al cliente. Casi siempre será “text/html” si el archivo HTX contiene HTML.

Las opciones avanzadas de ODBC, que se muestran en la Tabla 34, permiten depurar y ajustar el controlador ODBC utilizado por el Conector de bases de datos de Internet. Para obtener más detalles acerca de estas opciones, se debe consultar la documentación del controlador ODBC adecuado para el gestor de base de datos. La sintaxis genera para establecer estas opciones es:

```
ODBCOptions: Opción=Valor[,Opción=Valor…]
```

Por ejemplo, para detener la instrucción SQL si su ejecución dura más de 10 segundos y activar el seguimiento de las llamadas a las funciones de ODBC, tendría que especificar lo siguiente en el archivo IDC:

```
ODBCOptions: SQL_QUERY_TIMEOUT=10, SQL_OPT_TRACE=1,
SQL_OPT_TRACEFILE=C:\Sql.log
```

Tabla 34. Campos opcionales avanzados de ODBC

Opción	Valor	Propósito
SQL_ACCESS_MODE	0 = Lectura y escritura 1 = Sólo lectura.	Indicador para el controlador ODBC o el origen de datos de que la conexión no necesita admitir instrucciones SQL que produzcan una actualización. Este modo puede utilizarse para optimizar las estrategias de bloqueo, administración de transacciones u otras áreas del controlador o el origen de datos. No es necesario que el controlador impida la emisión de tales instrucciones hacia el origen de datos. El comportamiento del controlador y del origen de datos cuando se les pide que procesen instrucciones SQL que no sean de sólo lectura durante una conexión de sólo lectura está definido por cada implementación. SQL_ACCESS_MODE tiene el valor 0 de forma predeterminada, lo que permite lectura y escritura.
SQL_LOGIN_TIMEOUT	Entero	Número de segundos de espera para que se complete una petición de inicio de sesión antes de desconectar. El valor predeterminado depende del controlador y no puede ser cero. Si su valor es 0, el tiempo de espera queda desactivado y el intento de conexión puede durar indefinidamente. Si el tiempo de espera especificado sobrepasa el tiempo de espera máximo de inicio de sesión del origen de datos, el controlador sustituye el valor.
SQL_OPT_TRACE	0 = Seguimiento desactivado	Cuando el seguimiento está activado, todas las llamadas a funciones de ODBC invocadas por HTTPodbc.dll se escriben en el archivo de seguimiento. Puede especificar un archivo de

Opción	Valor	Propósito
	1 = Seguimiento activado	seguimiento con la opción SQL_OPT_TRACEFILE. Si el archivo ya existe, ODBC anexa entradas al archivo. De lo contrario, crea el archivo. Si se ha activado el seguimiento y no se especifica el archivo de seguimiento, ODBC escribe en el archivo Sql.log.
SQL_OPT_TRACEFILE	Nombre de archivo	Nombre del archivo de seguimiento que se va a utilizar cuando SQL_OPT_TRACE sea 1. El nombre predeterminado es SQL.LOG
SQL_PACKET_SIZE	Entero	Tamaño de paquete de la red, en bytes, que se va a utilizar para intercambiar información entre el sistema de administración de bases de datos (DBMS) y el servidor Web. Nota: Muchos orígenes de datos no aceptan esta opción o sólo devuelven el tamaño de paquete de la red. Si el tamaño especificado sobrepasa el tamaño máximo de paquete o es menor que el tamaño mínimo de paquete, el controlador sustituye dicho valor.
SQL_TRANSLATE_DLL	Nombre de archivo	Nombre de una DLL que contiene las funciones SQLDriverToDataSource y SQLDataSourceToDriver que el controlador carga y utiliza para realizar tareas como la traducción de juegos de caracteres.
SQL_TRANSLATE_OPTION	Entero	Valor que controla la funcionalidad de la traducción, que es específica de la DLL de traducción que se esté utilizando. Para obtener más detalles, consulte la documentación del controlador y de la DLL de traducción.

Opción	Valor	Propósito
SQL_TXN_ISOLATION	Entero 1= Lectura sin confirmar 2= Lectura confirmada 4= Lectura repetible 8= Serializable 16= Versiones	Establece el nivel de aislamiento de la transacción. El Conector de bases de datos de Internet no acepta transacciones que abarquen más que la petición del archivo IDC. Sin embargo, en algunos DBMS, si se asigna a la opción SQL_TXN_ISOLATION el valor 1 (Lectura sin confirmar) se tendrá más concurrencia y, por tanto, mayor rendimiento. Sin embargo, con este valor, se pueden recuperar datos que no se hayan confirmado en la base de datos mediante otras transacciones.
SQL_MAX_LENGTH	Entero	Máxima cantidad de datos de una columna de caracteres o binaria que el controlador va a devolver. Esta opción está destinada a reducir el tráfico de la red y sólo se debe utilizar cuando el origen de datos (no el controlador) de un controlador multicapa pueda implementarla.
SQL_MAX_ROWS	Entero	Máximo número de filas que se van a devolver en una instrucción SELECT. Si el valor es 0 (el predeterminado), el controlador devuelve todas las filas. Esta opción está destinada a reducir el tráfico de la red cuando el propio origen de datos puede limitar las filas devueltas, al contrario que la variable integrada MaxRecords de Conector de bases de datos de Internet, que limita las filas recuperadas.
SQL_NOSCAN	0= Busca y convierte las cláusulas de escape 1= No busca y convierte las cláusulas de escape	Especifica si el controlador no examina las cadenas SQL en busca de cláusulas de escape. Si es 0 (el valor predeterminado), el controlador busca cláusulas de escape en las cadenas SQL. Si es 1, el controlador no examina las cadenas SQL en busca de cláusulas de escape; en su lugar, el controlador envía la instrucción directamente al origen de datos. Si su instrucción SQL no contiene cláusulas de escape de ODBC, una sintaxis especial entre llaves ({ }), al asignar a esta opción el valor 1 se consigue una pequeña mejora en el rendimiento, al hacer que el controlador no examine la cadena SQL.

Opción	Valor	Propósito
SQL_QUERY_TIMEOUT	Entero 0= Sin tiempo de espera	Número de segundos de espera en la ejecución de una instrucción SQL antes de cancelar la consulta. Cuando su valor es 0 (el predeterminado) no hay tiempo de espera. Si el tiempo de espera especificado sobrepasa el tiempo de espera máximo del origen de datos, o si es menor que el tiempo mínimo de espera, el controlador sustituye dicho valor.
Entero	Específico del controlador	Se pueden especificar valores específicos del controlador con el formato número = valor. Por ejemplo, 4322=1, 234= Cadena

11.5.2 Como ejecutar una consulta IDC. El archivo de consulta IDC se invoca como cualquier otro con extensión *.htm* o *.html*, mediante un enlace, un botón de un formulario o escribiéndolo directamente en la línea de URL. Por ejemplo sea el archivo **articulos.idc**, con un enlace:

```
<A HREF="/aplicaciones/articulos.idc"> Ver artículos </A>
```

Desde un formulario:

```
<FORM METHOD="POST" NAME="ARTICULOS"
ACTION="/aplicaciones/articulos.idc">
<INPUT TYPE="SUBMIT" VALUE="Ver articulos">
</FORM>
```

Directamente: `HTTP://[servidorWeb]/aplicaciones/articulos.idc`

11.5.3 Paso de parámetros en IDC. Se puede tener acceso a los parámetros de los archivos conector de bases de datos de Internet desde el archivo con extensión HTML si antepone el prefijo “idc” y un punto al nombre del parámetro.

Sintaxis general: *texto de comentario <%idc.nombrecampo%>*

Ejemplo: `Autores cuyas ventas son mayores de <%idc.ventas%>`

Se pueden generar consultas más eficaces con bases de datos si se pasan parámetros al archivo IDC.

Los parámetros son los nombres y los valores de los controles en formato HTML, como el botón *Enviar,* y los nombres especificados directamente en las direcciones URL. Los navegadores de Web envían estos nombres y estos valores, que pueden utilizarse en las instrucciones SQL del servidor.

Nota: Cuándo pase parámetros, los espacios o los caracteres de escape como ?, &, y % pueden ser problemáticos. Cuando especifique un nombre de campo que vaya a utilizar como parámetro en otra consulta IDC, utilice la secuencia de escape *"%z"* para especificar un nombre de campo como: *<%"%z", NombreCampo %>* . Para efectos de ilustrar el paso de parámetro en esta metodología, se explicaran ciertas porciones de código como ejemplo. La siguiente instrucción SQL sólo devuelve los autores cuyas ventas anuales superen las 5000 unidades:

```
+SELECT au_lname, ytd_sales
+ from pubs.dbo.titleview
+ where ytd_sales>5000
```

Como se puede observar esto representa una limitante ya que se condiciona la consulta a un determinado valor, en el caso de que se desee adicionar consultas dinámicas al sitio en construcción. Mediante el uso de un parámetro, se podría diseñar una página Web que preguntara al usuario el valor en lugar de una cantidad predeterminada. La página Web tiene que pedir al usuario la cifra de ventas anuales y después asignar la cantidad a la variable asociada. A continuación se presenta la nueva solución. He aquí el formulario HTML con un campo de entrada para obtener el número:

```
<FORM METHOD="POST" ACTION="/scripts/archivo.idc">
Escriba las ventas anuales: <INPUT NAME="ventas" VALUE="5000">
<INPUT TYPE="SUBMIT" VALUE="Ejecutar consulta">
</FORM>
```

En el archivo IDC, se utilizara el valor del atributo *NAME* de la etiqueta *FORM*:

```
SQLStatement:
+SELECT au_lname, ytd_sales
+ from pubs.dbo.titleview
+ where ytd_sales > %ventas%
```

Los parámetros tienen que ir entre caracteres de porcentaje (%) para distinguirlos de los identificadores normales de SQL. Cuando el Conector de bases de datos de Internet encuentra el parámetro en el archivo IDC, sustituye el valor enviado por el navegador y después envía la instrucción SQL al controlador ODBC.

El carácter de porcentaje (%) también es un carácter comodín en SQL. Los comodines se utilizan en las consultas SQL para buscar elementos de una tabla que contengan determinados caracteres, para insertar un único carácter de porcentaje como comodín de SQL, utilice %%, esto impide que IDC utilice el carácter % como marcador de parámetro. Por ejemplo:

```
SQLStatement:
+SELECT au_lname, ytd_sales, title from pubs.dbo.titleview
+ where title like '%%%título%%%'
```

Para que el signo % se reconozca como comodín de SQL tiene que escribirlo dos veces y después agregar los signos % que marcan el parámetro para distinguir la cadena del parámetro. En el ejemplo, la consulta busca todas las entradas de la columna "title" que contengan la palabra título. Esta consulta devuelve lo siguiente:

```
Título
título y escritura
página de título principal
autor y título
```

Para devolver todas las entradas que contengan la palabra título como sus seis primeras letras, tendría que dar formato a la consulta como sigue:

```
SQLStatement:
+SELECT au_lname, ytd_sales, title
+ from pubs.dbo.titleview
+ where title like '%título%%%'
```

En este ejemplo, se obtienen los resultados siguientes:

```
título
título y escritura
```

Para devolver todas las entradas que contengan la palabra título como sus seis últimas letras, tendría que dar formato a la consulta como sigue:

```
SQLStatement:
+SELECT au_lname, ytd_sales, title
+ from pubs.dbo.titleview
+ where title like '%%%título%'
```

En este ejemplo se obtienen los resultados siguientes:

```
título
autor y título
```

11.6 ARCHIVO DE EXTENSIÓN HTML (HTX)

Los archivos de extensión son plantillas que representan y especifican el formato de salida de la pagina que contendrá los datos obtenidos con la consulta definida en el archivo IDC. Cuando la consulta se ejecuta, el conjunto de registros obtenidos se fusiona con el archivo de extensión (HTX) y los datos de la fuente ODBC. Estos datos combinados se adjuntan a los encabezados estándar de HTTP *(200 OK status, Content-Type; etc.*), que se pasan al servicio WWW y se devuelven al cliente, en la forma de un archivo HTML. El hecho de que el resultado es llevado a HTML es una de las grandes ventajas que ofrece esta metodología, por que esto implica que cualquier tipo de navegador usado pueda leer dichos resultados.

Nota: Para mayor claridad se recomienda dar el mismo nombre base a los dos archivos, por ejemplo: **articulos.idc** y **articulos.htx**, lo anterior se hace con el fin de tener un control más efectivo sobre el desarrollo que sé este realizando, aunque pueden utilizarse diferentes nombres de archivos para cada par IDC-HTX.

11.6.1 Palabras claves de los archivos HTX. Así como los archivos IDC poseen una serie de campos claves, los archivos de extensión HTML contienen seis palabras claves *(begindetail, enddetail, if, else, endif y "%z"),* que controlan forma en que se combinan los datos de la base de datos y el formato HTML en el archivo HTX, para generar el documento HTML de resultado.

Las cuales se encierran o enmarcan por <%%> o <!--%%--> que el archivo IDC utiliza para agregar datos dinámicos al documento, estas palabras claves rodean una sección del documento HTX en la cual el despliegue de datos de la base de datos serán fusionados. Dentro de esta sección, los nombres de los campos de la base de datos también se delimitan por <%%> o <!–%%-->, y son utilizados para marcar la posición de los datos resultantes de la consulta. La referencia a los nombres de los campos devueltos en la consulta se puede hacer en cualquier parte del archivo de extensión.

Los nombres de las columnas de la base de datos especifican qué datos se van a devolver en el documento HTML. Por ejemplo, la siguiente línea de un archivo HTX combina los datos de la columna *NombreCorreoElectrónico* para todos los registros procesados:

```
<%begindetail%><%NombreCorreoElectrónico%><%enddetail%>
```

Estas palabras cla ve se explican en los siguientes numerales.

11.6.1.1 <%begindetail%>, <%enddetail%>. Estas palabras clave enmarcan una sección del archivo con extensión HTML en la que se van a combinar datos procedentes de una base de datos. Dentro de esta sección, bs nombres de columna delimitadas con <% y %> o <!–% %--> se utilizan para marcar la posición de los datos devueltos por la consulta. La siguiente es la sintaxis general:

```
<%begindetail%>
<%nombre de columna(1)%>:<%nombre de columna(2)%>:...
<%nombre de columna(n)%>
<%enddetail%>
```

Por ejemplo:

```
<%begindetail%>
<%au_lname%>: <%ytd_sales%>
<%enddetail%>
```

Presentará las columnas *au_lname* y *ytd_sales*. De esta forma se puede hacer referencia a cualquier columna. También se puede hacer referencia a nombres de columna en cualquier parte del archivo con extensión HTML.

Nota: Si la consulta no devuelve registros, se saltará la sección <%begindetail%>. Por cada instrucción SQL que genere un conjunto de resultados (por ejemplo, SELECT), debe haber una sección *<%begindetail%> <%enddetail%>* correspondiente en el archivo HTX.

11.6.1.2 <%if%>, <%else%>, <%endif%>. Los archivos con extensión HTML pueden contener lógica condicional con una instrucción *if-then-else* para controlar cómo se genera la página Web. Por ejemplo, un uso común es insertar una condición para presentar los resultados de la consulta en la primera fila de las secciones *<%begindetail%>*; pero si la consulta no devuelve ningún registro, lo correcto seria colocar el mensaje adecuado para tal situación unido con la

sentencia *%idc.parametro%.* Si utiliza la instrucción <*%if%*> y una variable integrada llamada *CurrentRecord* puede personalizar el resultado de forma que se presente un mensaje de error cuando no se devuelvan registros. La sintaxis general es la siguiente:

```
<%if condición %>
texto HTML
[<%else%>
texto HTML]
<%endif%>
```

Donde ***condición*** tiene el formato:

`valor1` **`operador`** `valor2.` Y *operador* puede ser uno de los siguientes:

EQ: si valor1 es igual a valor2

LT: si valor1 es menor que valor2

GT: si valor1 es mayor que valor2

CONTAINS: si cualquier parte de valor1 contiene la cadena valor2

Los operandos *valor1* y *valor2* pueden ser nombres de campos de una tabla, una de las variables integradas *(CurrentRecord o MaxRecords),* el nombre de una variable HTTP, o una constante definida por el usuario.

Cuando se utilizan nombre de variables en una instrucción *<%if%>,* no se requiere delimitarlos con <% y %>. Por ejemplo, para hacer un proceso especial sobre el nombre del autor “Fuentes”, utilice la condición:

```
<%begindetail%>
<%if au_lname EQ "Fuentes"%>
es Fuentes
<%endif%>
<%enddetail%>
```

También se puede utilizar la instrucción *<%if%>* para hacer algún proceso especial basándose en la información de las variables HTTP. Por ejemplo, para dar formato a una página de manera diferente según el tipo de navegador Web del cliente, podría incluir lo siguiente en el archivo con extensión HTML.

```
<%if HTTP_USER_AGENT contains "Mozilla"%>
el cliente acepta características avanzadas de HTML
<%else%>
el cliente es <%HTTP_USER_AGENT%>
<%endif%>
```

En el siguiente ejemplo se muestra una sección del archivo HTX donde se aprecia el uso de la instrucción *<%if%>*, se desea conocer que autores tienen ventas anuales superiores a cierta cantidad de dinero.

```
<%begindetail%><%if CurrentRecord EQ 0%>
<caption>Resultados de la consulta:</caption>
<tr> <th><b>Autor</b></th>
      <th><b>Ventas<br>en dólares)</b></th>
</tr>
<%endif%>
   <tr> <td><%au_lname%></td> <%ytd_sales%></td> </tr>
<%enddetail%>
<p><%if CurrentRecord EQ 0%>
     <b><i>Lamentablemente, no hay autores que hayan vendido más
de </i><%idc.sales%>.</b><%endif%></p>
```

Las secciones *<%begindetail%>* y *<%enddetail%>* delimitan el lugar del documento donde aparecerán las filas devueltas por la base de datos. Las columnas devueltas por la consulta aparecerán rodeadas por <%%>; en este ejemplo son *<%au_lname%>* y *<%ytd_sales%>.*

11.6.1.3 Variables integradas CurrentRecord, MaxRecords. La variable integrada *CurrentRecord* contiene el número de veces que se ha procesado la sección *<%begindetail%>.* La primera vez que se pasa por la sección *<%begindetail%>,* el valor es cero, después, el valor de *CurrentRecord* se incrementa por cada registro devuelto por la base de datos.

La variable integrada *MaxRecords* contiene el valor del campo *MaxRecords* del archivo Conector de bases de datos de Internet. *MaxRecords* y *CurrentRecord* sólo se pueden utilizar en instrucciones *<%if%>.*

11.6.1.4 Secuencia de escape "%z". Cuando especifique un nombre de campo que vaya a utilizar como parámetro en otra consulta IDC, utilice la secuencia de escape "%z" para especificar un nombre de campo como: *<% "%z", NombreCampo %>.*

11.7 UTILIZAR CUADROS DE LISTA DE SELECCIÓN MÚLTIPLE DE FORMULARIOS HTML

Cuando se utiliza un formulario HTML que contiene la etiqueta *<SELECT MULTIPLE…>*, el conector de bases de datos de Internet convierte los elementos seleccionados en una lista separada por comas, la lista puede utilizarse en el archivo IDC como otros parámetros. Sin embargo, como el parámetro es en realidad una lista, normalmente se utilizará sólo en instrucciones SQL Select que tengan una cláusula *IN*, como en los ejemplos siguientes.

Si el nombre del parámetro en el archivo IDC se encuentra entre comillas simples, todos los elementos de la lista también irán entre comillas simples. El nombre del parámetro tiene que ir entre comillas simples siempre que la columna de la cláusula *IN* sea una columna de tipo carácter o de otro tipo que acepte literales entre comillas (por ejemplo, fechas y horas). Si no hay comillas simples alrededor del nombre del parámetro, tampoco se pondrán comillas simples alrededor de los elementos de la lista.

No hay que poner el nombre del parámetro entre comillas simples cuando la columna de la cláusula *IN* sea de tipo numérico o de cualquier otro tipo en el que los literales no vayan entre comillas simples. Por ejemplo, si un formulario HTML contuviera el cuadro de lista de selección múltiple que se muestra a continuación:

```
<SELECT MULTIPLE NAME="región">
<OPTION VALUE="Oeste">
<OPTION VALUE="Este">
<OPTION VALUE="Norte">
<OPTION VALUE="Sur">
</SELECT>
```

Puede generar un archivo IDC con una sentencia SQL:

SQLStatement:SELECT name, región FROM customer WHERE región IN '%región%')

Si el usuario selecciona “Norte”, “Oeste” y “Este” en el formulario HTML, la instrucción SQL quedaría convertida en:

```
SELECT name, región FROM customer WHERE región IN ('Norte',
'Oeste', 'Este')
```

Otro ejemplo de formulario HTML es el mostrado a continuación; pero esta vez utiliza datos numéricos y por tanto, el parámetro no va entre comillas simples en el archivo IDC.

```
<SELECT MULTIPLE NAME="año">
<OPTION VALUE="1994">
<OPTION VALUE="1995">
<OPTION VALUE="1996">
</SELECT>
```

Puede generar un archivo IDC con la instrucción SQL:

```
SQLStatement : SELECT product, sales_year FROM sales WHERE
sales_year IN(%año%)
```

Si el usuario selecciona “1994” y “1995” en el formulario HTML, la instrucción SQL quedaría convertida en:

```
SELECT product, sales_year FROM sales WHERE sales_year IN
(1994, 1995)
```

11.8 USO DE CONSULTAS POR LOTES Y CONSULTAS MÚLTIPLES

En un archivo IDC puede agrupar consultas SQL de dos maneras, como consultas por lotes o como consultas múltiples.

11.8.1 Consultas por lotes. Si va a consultar bases de datos que puedan procesar varias œonsultas Simultáneamente en una misma instrucción SQL, debe dar formato a sus instrucciones con la sintaxis de consultas por lotes para optimizar el rendimiento. Por ejemplo:

```
SQLSTatement:
+insert into perf(testtime, tag) values (getdate(), '%tag%')
+SELECT au_lname, ytd_sales from pubs.dbo.titleview where
ytd_sales>5000
+SELECT count(*) as nrecs from pubs.dbo.titleview where
ytd_sales>5000
```

11.8.2 Consultas múltiples. Para las bases de datos que no soporten procesar una serie de consultas SQL simultáneamente, se pueden formular las consultas como consultas múltiples. Por ejemplo:

```
SQLStatement:
+insert into perf(testtime, tag) values (getdate(), '%tag%')
SQLStatement:
+SELECT au_lname, ytd_sales from pubs.dbo.titleview where
ytd_sales>5000
SQLStatement:
+SELECT count(*) as nrecs from pubs.dbo.titleview where
ytd_sales>5000
```

Las consultas por lotes se procesan a la vez, mientras que las consultas múltiples se procesan de una en una. Por tanto, se obtiene un mayor rendimiento si se le da formato a las consultas por lotes y si la base de datos puede procesar dichas consultas.

11.9 VARIABLES DE HTTP

Hay diversas variables en los archivos con extensión HTML que pueden ofrecer mucha información acerca del entorno y del cliente Web conectado con el servidor, son las variables de entorno CGI y variables HTTP para aplicaciones IDC. Además, están disponibles todos los encabezados enviados por el cliente.

Para tener acceso a ellos desde el Conector de bases de datos de Internet, deben ser convertidos. Los siguientes son los pasos de conversión:

- Agregar HTTP_ al principio.
- Convertir todos los caracteres de guiones en caracteres de subrayado.
- Convertir todas las letras a mayúsculas.

11.10 EJEMPLOS

Ejemplo 1. Inserción de registros en tablas: a continuación se presenta el código de una consulta IDC que realiza la inserción de registros sobre una tabla.

El siguiente formulario HTML para capturar datos:

```
<html>
<body>
  INSERTA CLIENTE
<form method="POST" action="../../SCRIPTS/TESIS/INSERTA.IDC">
  IDENTIFICACION DEL CLEINTE<input type="text" name="T1">
  NOMBRE DE LA COMPAÑÍA <input type="text" name="T2">
  DIRECCION <input type="text" name="T3>
  CIUDAD <input type="text" name="T4">
<input type="submit" value="Enviar" >
<input type="reset" value="Restablecer">
</form>
</body>
</html>
```

Consulta IDC. El siguiente es el código del archivo IDC que recibe los datos enviados y los manipula para insertarlos en la tabla indicada en el código.

```
Datasource: DB_NEPTUNO
Template: inserta.htx
SQLStatement: INSERT INTO CLIENTES
+(idcliente,nombrecompañía,dirección,ciudad) VALUES
+ ('%T1%','%T2%','%T3%','%T4%') ;
```

Donde los valores T1,T2,T3,T4 representan los campos de captura que contienen la información del registro a adicionar, y serán remplazados a igual correspondencia con los campos escritos en la consulta. Es decir idcliente con T1, nombrecompañía con T2 y así sucesivamente. Estos campos pueden ser definidos en la captura con el mismo nombre de los campos (columnas) de la tabla a afectar.

Plantilla HTX.

```
<html>
<body>
<p align="center"><big>REGISTRO INSERTADO...OK.</big></p>
href="HTTP://servidor/tesis/presentacion.htm">REGRESAR</a></p>
</body>
</html>
```

Cabe recordar que no todos los archivos HTX, suelen contener plantillas para visualizar resultados como tal, de un archivo IDC, en el ejemplo anterior se usa para colocar un mensaje y un vinculo al marco de una pagina principal.

Ejemplo 2. Listado de registros desde una tabla: a continuación se exponen los archivos IDC y HTX de una consulta IDC que lista el contenido de una tabla.

Consulta IDC.

```
Datasource: DB_NEPTUNO
SQLStatement: SELECT
idcliente,nombrecompañía,dirección,ciudad from clientes
Template: LISTARNEP.htx
```

Plantilla HTX.

```
<html>
<head>
<title>LISTADOS DE REGISTROS EN TABLA CLIENTES</title>
</head>
<body>
<p align="center">LISTA REGISTROS</p>
<table border="1" width="100%" height="63">
  <tr>
    <td width="25%" height="18"><p align="center">ID CLIENTE</td>
    <td width="25%" height="18"><p align="center">COMPAÑIA</td>
    <td width="25%" height="18"><p align="center">DIRECCION</td>
    <td width="25%" height="18"><p align="center">CIUDAD</td>
  </tr>
<%begindetail%>  <tr>
    <td width="25%" height="33"><p
align="center"><%idcliente%></td>
    <td width="25%" height="33"><p
align="center"><%nombrecompañía%></td>
    <td width="25%" height="33"><p
align="center"><%dirección%></td>
    <td width="25%" height="33"><p align="center"><%ciudad%></td>
  </tr>
<%enddetail%></table>
</body>
</html>
```

11.11 EVALUACIÓN DE LA METODOLOGÍA

El ambiente donde se evaluó la metodología fue una intranet basada en plataforma computacional Windows. La metodología en estudio se evaluó con los gestores de bases de datos *MS SQL Server 7.0, ACCESS 2000*.

Realizando operaciones clásicas sobre las tablas de las bases de datos seleccionadas para tal efecto, a saber: inserción, eliminación, modificación, listado de registros.

11.12 RESULTADO DE LA EVALUACIÓN

Los objetivos de la evaluación se dieron de forma satisfactoria, las consultas diseñadas para cada operación considerada sobre registros (inserción, eliminación, listado, modificación) contenidos en las tablas tuvieron el éxito esperado. En términos generales IDC, representa una metodología de nivel intermedio la cual posee puntos a favor y puntos en contra.

En la actualidad esta metodología ha sido relegada por otras más potentes, las cuales están mas a tono con los requerimientos del momento. Su desempeño es aceptable, la estructura de SQL, permite que IDC logre tiempos de respuestas acorde a las circunstancias en donde sé este utilizando.

11.12.1 Lenguaje. Se hace necesario que el desarrollador tenga cocimientos básicos del lenguaje SQL, si se utilizara esta metodología en una aplicación robusta se hace obligatorio que el conocimiento de dicho lenguaje sea proporcional claro esta, que para desarrollar aplicaciones, se debería utilizar metodologías mas adecuadas en donde IDC se use como una técnica intermedia.

SQL representa un gran aliado en esta técnica las construcciones de códigos que se pueden dar son realmente ilimitadas. Acerca de HTML, se debe tener un buen conocimiento de este debido a que muchos comandos o campos usados por esta metodología están estrechamente ligados con el comportamiento de ciertas cláusulas de HTML.

11.12.2 Soporte a plataformas. Como es una metodología Microsoft, es obvio que posee una asistencia total a los sistemas operativos de esta compañía, incluyendo motores de bases de datos, interfaces nativas de motores de bases de datos, servidores de aplicación; etc. Respecto a la transportabilidad de código entre plataformas, se puede decir que esta técnica solo se aplica a plataformas *Microsoft*, por lo que su rango de operación hacia otras plataformas computacionales diferentes a esta no es viable.

Dado lo anterior la técnica IDC esta ligada a las plataformas Microsoft, lo cual la hace dependiente de dichas plataformas computacionales. La administración de esta técnica mas exactamente las consultas generadas usando sus parámetros, suelen estar supeditadas por el servidor Web que las manipula.

11.12.3 Soporte a bases de datos. La fuente de datos ODBC, hace que cualquier metodología que la soporte tenga un acceso casi ilimitado a los gestores de base datos mas usados en la actualidad, dentro de los que se pueden destacar: *Oracle, MS SQLServer, Access, Visual FoxPro* y *Paradox*.

11.12.4 Flexibilidad. Es una técnica fácil de aprender, que se brinda una opción viable para la creación de soluciones interesantes aplicadas en la Web. Al ser una técnica que descarga todo el trabajo sobre el servidor, parecería que esto le restara puntos a su desempeño; pero no y esto debido a que es una de las metodologías interpretada como se anoto en secciones anteriores. En pocas palabras IDC es fácil de usar y programar. La generación de las consultas requiere un mínimo esfuerzo de programación por lo cual esto le asigna cierta ventaja respecto a otras técnicas.

11.12.5 Soporte. La documentación existente acerca de esta metodología es abundante en el Web, además su filosofía se encuentra documentada en una proporción aceptable en la documentación de instalación de la plataforma Windows NT.

12. COLDFUSION

ColdFusion de *Allaire Corp.,* es un sistema para la rápida implementación de aplicaciones Web dinámicas y sitios Web interactivos, dirigido a desarrolladores profesionales. Este provee la manera más rápida para integrar navegador, servidor y tecnologías de bases de datos dentro de aplicaciones Web robustas.

Esta es una herramienta que hace parte de un conjunto de mayor que reúne varios elementos de *software*, cuyo objetivo es el de albergar la lógica comercial, es decir, el código que permite el acceso, la manipulación y validación de los datos de múltiples servidores corporativos, para luego enviar a los los clientes la información que necesitan. (Véase la Figura 30). A este tipo de macro herramientas se le conoce con el nombre de *Servidores de Aplicaciones.*

El desarrollo de aplicaciones con *ColdFusion* no requiere la codificación en un lenguaje de programación tradicional; en lugar de eso, las aplicaciones se construyen mediante la combinación de HTML estándar con un sencillo lenguaje de marcación de servidor, el *ColdFusion Markup Languaje (CFML).*

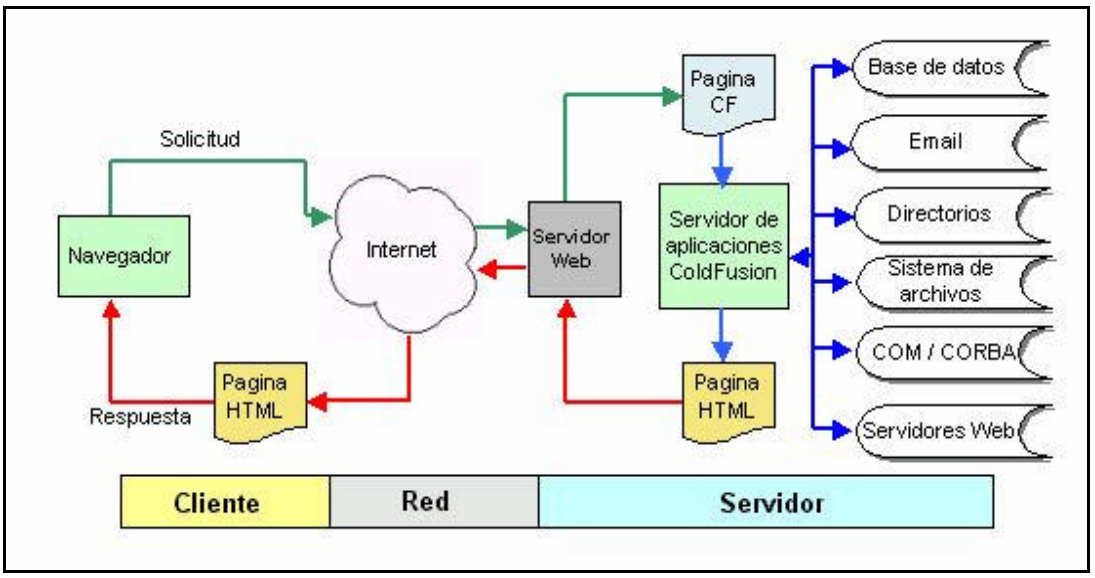

Figura 30. Arquitectura de la herramienta ColdFusion

12.1 CFML

Este lenguaje basado en etiquetas puede ser usado para crear *scripts* del lado del servidor. Para una integración dinámica de los datos, la lógica de aplicación, y la generación de la interfaz de usuario. Las aplicaciones Web pueden contener XML, HTML, y otras tecnologías clientes tales como CSS o JavaScript.

Como cualquier lenguaje de programación, CFML posee la estructura procedural que se requiere para escribir la funcionalidad necesaria de cualquier programa. Todas las estructuras de control y decisión de cualquier lenguaje de programación están contempladas en *ColdFusion*. Las paginas de aplicación de *ColdFusion* se diferencian de las paginas estáticas HTML en los siguientes puntos:

- Son guardadas y referenciadas por una extensión especifica.
- La extensión por defecto es CFM.
- Contienen *ColdFusion Markup Languaje.*

ColdFusion proporciona un vasto conjunto de características que permite:

- Desarrollo rápido.
- Desarrollo escalable.
- Integración abierta.
- Seguridad completa.

12.2 INTEGRACIÓN CON LAS BASES DE DATOS

Debido a que *ColdFusion* es un lenguaje basado en etiquetas, estas están clasificadas según su funcionalidad. Son muchas y de una u otra manera para realizar alguna operación existe una relación de *enlace* entre ellas, un análisis detallado de las etiquetas para el acceso a fuentes de bases de datos de este lenguaje esta fuera del alcance de este documento. Para el acceso y manipulación de datos se cuenta con un conjunto definido de etiquetas.

El siguiente conjunto de etiquetas son las utilizadas regularmente para realizar las operaciones con la base de datos, para obtener una visión mas detallada de estas se recomienda remitirse a la documentación oficial del lenguaje.

- **CFINSERT:** inserta un nuevo registro en la fuente de datos.

- **CFQUERYPARAM:** verifica el tipo de dato de un parámetro de consulta. La etiqueta CFQUERYPARAM se enlaza con una etiqueta CFQUERY. Específicamente es embebida dentro de la sentencia SQL. Si se especifican parámetros opcionales, CFQUERYPARAM también puede ejecutar la validación de datos.

- **CFPROCPARAM:** la etiqueta CFPROCPARAM se relaciona con una etiqueta CFSTOREDPROC. Se utiliza en la información especifica del parámetro, incluyendo tipo, nombre, valor, y longitud.

- **CFSOREDPROC:** La etiqueta CFSOREDPROC es la principal etiqueta usada para ejecutar *stored procedures* (procedimientos almacenados) vía ODBC o usando la conexión nativa de una fuente de datos.

- **CFPROCRESULT:** la etiqueta CFPROCRESULT es anidada dentro de una etiqueta CFSTOREDPROC. El parámetro *NAME* de esta etiqueta especifica el nombre de un conjunto de resultado al igual que otras etiquetas *ColdFusion*, como son CFOUTPUT y CFTABLE, usadas para tener acceso al conjunto de resultado.

- **CFTRANSACTION:** se utiliza CFTRANSACTION para agrupar consultas en una sola parte. CFTRANSACTION también provee el tratamiento *commit* y *rollback.*

- **CFQUERY:** la etiqueta CFQUERY pasa las sentencias SQL a la fuente de datos para cualquier tratamiento.

- **CFUPDATE:** la etiqueta CFUPDATE actualiza los registros existentes en la base de datos.

El acceso a las fuentes de datos puede hacerse a través de ODBC (bajo *Windows NT*) o los soportes nativos. *ColdFusion* posee soporte nativo para *Oracle(7, 8.0)*, *Informix 7.3, DB2, Sybase* y fuentes de datos OLE DB.

El siguiente es un ejemplo de un formulario CFM :

```
<!--Fomulario.cfm-->
<html>
<!--Conexión a la fuente de datos ODBC a traves del DSN y consulta
SQL-->
<CFQUERY NAME="buscaclientes" DATASOURCE="aspclientes" >
  SELECT idcliente
  FROM cliente
  ORDER BY idcliente
</CFQUERY>
<!--formulario que el cliente envia al servidor Web-->
<CFFORM name="form1" action="http://192.168.0.2/resultados.cfm"
```

```
method="post" >
  <p> Cliente
    <select name="select" >
    <!--se utiliza el nombre la consulta anteriomente realizada
    como una referencia al conjunto de resultados-->
    <CFOUTPUT QUERY = "buscaclientes">
    <option value ="#idcliente#" >#idcliente# </option>
    </CFOUTPUT>
   </select>
  </p>
  <input type="submit" name="Enviar">
</CFFORM>
</body>
</html>
```

Cuándo se envía el anterior formulario, ejecuta en el servidor la pagina CFM *resultados.cfm,* esta pagina es procesada por el *servidor ColdFusion* y *envia* la respuesta al servidor Web, enviando al cliente la información solicitada en formato HTML o XML.

```
<!--resultados.cfm-->
<html>
<body>
<!--Conexión a la fuente de datos ODBC a traves del DSN y consulta
SQL-->
<CFQUERY NAME="clientes_actuales" DATASOURCE="aspclientes"
dbtype="ODBC">
     SELECT *
     FROM cliente
     WHERE idcliente = #form.select#
      <!como parametro de consulta se utiliza el valor enviado por
```

```
      el cliente en el formulario-->
</CFQUERY>
<Table border="1">
  <Tr>
     <Td>Idcliente</Td> <Td>Nombre</Td> <Td>Apellidos</Td>
     <Td>Telefono</Td> <Td>Email</Td>
  </Tr>

<!--se imprime el resultado de la consulta usando el nombre de la
consulta como referencia-->
<CFOUTPUT QUERY="clientes_actuales">
     <Tr>
          <Td>#idcliente#</Td> <Td>#nombrecli#</Td>
          <Td>#apellidoscli#</Td> <Td>#telefonocli#</Td>
         <Td>#emailcli#</Td>
     </Tr>
</Cfoutput>
</Table>
</body>
</html>
```

12.3 EVALUACIÓN DE LA METODOLOGÍA

Para la evaluación de esta interfaz se utilizo la plataforma Windows NT 4.0 y el servidor Web *IIS 4*, la herramienta *ColdFusion Server* versión 4.5, con una fuente de datos ODBC. Para generar el código de las interfaces de usuario (los archivos *.cfm) se utilizo la herramienta *CodeCharge* versión 1.1, la cual genera un código 100% CFML muy depurado.

12.4 RESULTADO DE LA EVALUACIÓN

ColdFusion es el servidor de aplicaciones producido por la empresa *Allaire Corporation*, es uno de los productos lideres en el campo de los *servidores de aplicaciones* Web totalmente equipados. Su mercado es la pequeña y mediana empresa.

12.4.1 Lenguaje. *ColdFusion* es un lenguaje de *scripting* para desplegar información, no se puede crear verdaderas aplicaciones, es primitivo ya que no es posible definir funciones y no soporta la sintaxis de operadores estándar, aunque es muy bueno para el despliegue de paginas y la interacción con la bases de datos. Para un desarrollador acostumbrado a la codificación en lenguajes verdaderos (C, pascal, java, etc.), implementar aplicaciones con el conjunto de estas etiquetas, de buenas a primera no le resultaría muy cómodo. Conocer y aprender el lenguaje es tedioso y realmente lento.

El servidor *ColdFusion* libera al servidor Web de las tareas de procesamiento de los archivos CFML, permitiendo que exista rapidez en el procesamiento de la respuesta. La arquitectura es sólida, aunque el paradigma basado en etiquetas y páginas podría resultar torpe en grandes proyectos de desarrollo.

12.4.2 Soporte a plataformas. *ColdFusion* esta limitado a un seleccionado conjunto de plataformas: Windows (NT, 9x, 2000), Solaris, Linux y HP/UX.

12.4.3 Soporte a bases de datos. Las conexiones a las bases de datos se hacen de una manera abstracta, haciéndolo fácil de usar. Permite el cambio de plataforma de fuente de datos, sin cambios en el código. Se puede desarrollar el código con una fuente de datos e implementarlo en otra.

Posee soporte nativo para varios motores de bases de datos (este soporte nativo solo esta disponible en la versión *Enterprise* de la herramienta), y esta limitado a pocos productos. También se obtiene acceso a fuentes de datos ODBC en ambientes *Windows*.

12.4.4 Flexibilidad. No es una herramienta muy flexible debido a las características propias de un lenguaje de etiquetas y a la poca o nula libertad que tiene el desarrollador de implementar sus propias funciones.

Posee un buen manejo de los errores, tanto en tiempo de programación como el de ejecución. Para aplicaciones que lo requieran, permite trabajar con objetos CORBA, COM, *servlets* de Java, JavaScript, generar contenido basado en XML y más.

12.4.5 Soporte. Este es un producto comercial, es uno de los servidores de aplicación más baratos del mercado comparados con otros de su categoría, aunque no deja de ser costoso. El producto esta excelentemente documentado. Cuenta además con una buena comunidad de desarrolladores que ofrecen sus conocimientos a través de sitios Web.

13. RESULTADOS GENERALES DE LAS EVALUACIONES

Durante las etapas de investigación y evaluación de las diferentes metodologías, se pudo observar que existen dos maneras que se distinguen para crear las aplicaciones que permiten acceder a los gestores de bases de datos a través del Web.

La primera alternativa la denominamos *básica,* esta consiste en la construcción de la aplicación digitando código, esto es, el desarrollador escribe el código utilizando un lenguaje de programación conocido y un API que acceda a la base de datos, para implementar la aplicación. Ejemplo de este tipo son:

- El lenguaje Perl y el modulo DBI.
- El lenguaje Java (Servlets y API JDBC).
- El ambiente de desarrollo ASP y ADO.
- PHP y las interfaces de bases de datos.
- C++ y ODBC.
- TCL/Tk y sus interfases de bases de datos.

La segunda alternativa consiste en utilizar una utilidad *herramienta–interfaz server side.* Una utilidad de este tipo puede suministrar:

- Un lenguaje propio.
- Soporte nativo para varios gestores de bases de datos.
- Soporte a interfaces cliente de bases de datos y tecnologías diversas.
- Asistentes para el desarrollo de las aplicaciones y mas aditamentos.
- O bien, pueden ofrecer una solución integral de una o varias de las soluciones 'básicas'.

En esta segunda alternativa se identificaron dos tipos de productos:

Las interfaces de servidor: herramientas pequeñas y poco conocidas que generalmente actúan como programas CGI. Son herramientas recomendadas para labores en las que no se demanden grandes desempeños, como una pequeña *Intranet* o donde el volumen de transacciones sea muy bajo. La Tabla 35, contiene una muestra representativa de este tipo de herramientas. Estas interfaces poseen características que las hace muy útiles al momento de desarrollar una aplicación básica especifica, que las convierten en herramientas eficientes. Muchas poseen su propio lenguaje de programación estructurado, creado claro esta, para la tarea que les es encomendada.

El lenguaje puede ser, un lenguaje basado en etiquetas, como lo poseen las interfaces *ODBiC* y *WhizBase,* también suele utilizar o un lenguaje propietarios como *VBScript o JScript* en el caso de la interfaz *OpenX*.

Tabla 35. Interfaces de servidor

Herramienta	Plataforma				Interfaces cliente soportadas	Soporte nativo a gestores
	Linux	Win NT	Win 98	Otras		
OpenX		•			ODBC, ADO	MS SQL Server, Oracle, Sybase, IBM DB2, Informix, InterBase y Centura SQLBase.
Baserunner	•	•	•			FoxPro, dBase IV
ODBiC		•	•		ODBC	
WhizBase		•	•		ODBC	MSAccess,dBase, Paradox, FoxPro y archivos ASCII
WebData	•	•				MySQL
XWorks		•	•			Visual FoxPro
FoxWeb		•	•			Visual FoxPro
WebBase		•	•		ODBC	
Corel Web. data		•	•		ODBC	MS Access, MS Excel, MS Foxpro, dBASE, Lotus 123 , Oracle.

Las interfaces *robustas:* son herramientas que están dentro de los denominados *servidores de aplicaciones.* Estas herramientas tienen dos propósito fundamentales: primero convertir la lógica comercial en código de

software; segundo llevar ese código a la capa media de tal forma que pueda soportar varios servidores y clientes livianos - navegadores Web, dispositivos compatibles con el *WAP (Wireless Access Protocol: Protocolo de Aplicaciones Inalámbricas),* buscapersonas y demás -. La siguiente tabla contiene una lista de algunos de los servidores de aplicación más comunes:

Tabla 36. Servidores de aplicación

Herramienta	Plataforma				Interfaz cliente soportadas	Soporte Nativo
	Linux	Win. NT	Win. 98	Otras		
WebObjects		•		•	ODBC, JDBC	Oracle, Sybase y Informix
IE Integrator	•	•		•	ODBC	DB2, Oracle, Sybase y MS SQL Server
Sapphire/Web		•		•	ODBC, JDBC, ADO, ORB, DSO	DB2, Informix, MS SQL Server, Oracle y Sybase
ColdFusion	•	•	•	•	ODBC, OLE, ORB	Oracle, Sybase
Application Server		•		•		DB2, Oracle, MS SQL Server, Informix, SQL Anywhere, Sybase
Jade /Business Logic Server	•	•	•	•		Oracle, Informix, Sybase, MS SQL Server, Access y DB2
Tango		•		•	ODBC	Oracle, FileMaker Pro
Enhydra	•	•	•	•		

Este tipo de herramientas, posee ambientes de desarrollo integrado (IDE), herramientas para crear sitios Web *y* funciones de conectividad de fuentes de datos, ofrecen además compatibilidad con diversos lenguajes de programación, escalabilidad, seguridad, protección a fallos, estas son recomendadas para macro operaciones de alto desempeño ya que son diseñadas para el ámbito empresarial o corporativo.

Cabe destacar además, que algunos fabricantes de gestores de bases de datos proveen sus propias soluciones o interfases. Estas interfases o herramientas propietarias, pertenecen al segundo tipo anteriormente mencionado. Están diseñadas para ofrecer la posibilidad de publicar contenido en cualquier sitio Web, junto con capacidades de trabajo en grupo, comunicación y negocios en Internet.

Algunos de estos productos son conocidos como *portales empresariales*, prácticamente caen dentro de la categoría de los *servidores de aplicaciones* ya que soportan muchas tecnologías y especificaciones de otros fabricantes; pero están generalmente, ligados al gestor de su casa matriz.

Un portal empresarial es una aplicación que le permite a una corporación o empresa dar a sus clientes, socios y empleados un acceso único y personalizado a la información del negocio, de tal manera que puedan tomar decisiones acertadas y relacionarse con otros a través de la red.

Algunas de estas herramientas son:

- Oracle WebDB.
- Oracle Application Server.
- Sybase Enterprise Portal.
- Sybase Enterprise Application Server.
- Progress Aptivity, Webspeed.

Los siguientes cuadros resumen los resultados obtenidos de cada metodología evaluada:

Cuadro 1. Soporte a plataformas de las diferentes metodologías

	Plataforma			
Metodología	**Linux/Unlx**	**Windows NT**	**Windows 98**	**Otras**
ADO-ASP		•	•	
ODBiC		•	•	
IDC-HTX		•	•	
Interfaz PHP	•	•	•	•
Servlet-JDBC	•	•	•	•
Modulo Perl DBI	•	•	•	•
ColdFusion	•	•	•	•

Cuadro 2. Tecnologías de servidor soportada o implementada

	Tecnología		
Metodología	**CGI**	**ISAPI**	**Servlets**
ADO-ASP		•	
ODBiC	•		
IDC-HTX		•	
Interfaz PHP	•		
Servlet-JDBC			•
Modulo Perl DBI	•		
ColdFusion	•	•	•

Cuadro 3. Interfaces clientes soportadas y capacidad de soporte nativo

	Interfaz cliente					
Metodología	**ODBC**	**OLE DB**	**JDBC**	**ADO**	**DBI**	**Soporte Nativo**
ADO-ASP	•	•		•		
ODBiC	•					
IDC-HTX	•	•		•		
Interfaz PHP	•		•	•		•
Servlet-JDBC	•		•			
Modulo Perl DBI	•		•	•	•	•
ColdFusion	•	•				•

Cuadro 4. Facilidad de aprendizaje del lenguaje y/o metodología

	Aprendizaje			
Metodología	**Accesible**	**Rápido**	**Complejo**	**Lento**
ADO-ASP	•	•		
ODBiC	•			•
IDC-HTX	•	•		
Interfaz PHP	•	•		
Servlet-JDBC			•	•
Modulo Perl DBI	•	•		
ColdFusion	•			•

14. CONCLUSIONES

Este trabajo nació con el objetivo de evaluar las tecnologías y metodologías que permiten: accesar los gestores de datos relacionales, modificar y publicar la información contenida en ellos de una manera interactiva, a través de Internet como medio de comunicación y el navegador como interfaz cliente con el usuario final. Teniendo en cuenta este propósito se pueden enunciar las siguientes conclusiones:

A lo largo del proceso, se adquirió amplio conocimiento sobre las diferentes técnicas y tecnologías existentes, de sus sistemas, arquitecturas, características de actuación y enfoques. Logrando plantear una metodología coherente que permitiera evaluar bajo un mismo patrón las diferentes técnicas. Se pudo conocer que el conjunto de estas técnicas es extenso y cambiante, ya que existen innumerables propuestas tecnológicas y metodológicas, que abarcan muchas filosofías y múltiples plataformas computacionales.

Se pudo ademas determinar que:

Realmente son las tecnologías clientes de bases de datos las que realizan todo el trabajo. Trabajo que es transparente al la aplicación de servidor y por lo tanto al

resto del sistema informático (incluido el usuario). Una buena parte de las tecnologías clientes de bases de datos son tecnologías *Microsoft*. Mientras que otras son excelentes interfases con arquitecturas diversas de otros notables productores de software.

Las tecnologías propuestas por los lenguajes de programación nacidos en Unix*: PHP, Perl, Java, Tcl*, a manera individual ofrecen contextos de desarrollo homogéneos y robustos, que ofrecen grandes posibilidades de:

- Escalabilidad.
- Flexibilidad.
- Soporte multiplataforma.
- Economía. Además de realizar su trabajo de manera eficiente.

La eficiencia de en tiempo de ejecución de las aplicaciones dependerá de múltiples factores, tales como:

- La plataforma computacional.
- La tecnología y filosofía del software de servidor Web.
- El diseño y eficiencia del código de la aplicación.
- Los tiempos de respuesta del gestor de base de datos a las consultas.
- Los controles de seguridad de la implementación.
- La arquitectura de la propia metodología.

Al culminar la elaboración del presente trabajo, se puede concluir que todos los objetivos tanto el objetivo general y los objetivos específicos fueron trabajados y se pudieron alcanzar los planteamientos de cada uno.

A continuación se redacta en forma pormenorizada, los aspectos más relevantes en el desarrollo de cada objetivo.

- *Elaboración de una propuesta metodológica que facilite la implementación de proyectos de sistemas que permitan accesar bases de datos en plataformas centralizadas a través de sitios Web.*

Este como objetivo principal se logro por completo, se documentaron las técnicas de acceso a bases de datos a través de sitios Web, que en la actualidad gozan con la mayor aceptación en el medio. Dichas técnicas además de ser representativas, gozan de una arquitectura altamente técnica y de punta que hacen de la documentación final un material de consulta.

- *Conducir una investigación objetiva que permita conocer las principales metodologías para accesar bases de datos a través de sitios Web.*

De este objetivo se puede decir que fue el motor que impulso el desarrollo del proyecto, esto debido a su naturaleza (investigativo-descriptivo). Cabe anotar que el material recopilado para su posterior análisis sobre cada metodología, tenia ciertos puntos que trabajar en forma extra debido a factores tales como:

- Estructura de la documentación.
- Filosofía de cada técnica, herramienta, metodología, motor de base de datos, servidor Web, servidor de red, entre otros.

No obstante todos los contratiempos que se presentaron, en donde se vieron involucrados el aspecto técnico *(hardware)* y el aspecto lógico *(software),* enriquecieron nuestros conocimientos y por supuesto el documento final.

- *Clasificación de las metodologías de acuerdo a la arquitectura plataforma de sistema operativo que estas abarquen.*

Se utilizaron dos plataformas de sistemas operativos de servidores para la realización de este proyecto a saber: Windows NT 4.0 y Linux SuSE 7.0. Las distintas técnicas de acceso que fueron seleccionadas para la evaluación se distribuyeron en estas dos plataformas.

- *Evaluación de los motores de bases de datos más comunes.*

Se escogieron los motores de bases de datos que tiene mayor relevancia en la actualidad, a saber: MS SQL SERVER 7.0, ACCESS 2000, InterBase versión 6, PostgreSQL versión 7.0, MySQL versión 3.22.32.

Con todo lo anterior expuesto, podemos concluir que los objetivos y alcances del proyecto se cumplieron a cabalidad.

15. RECOMENDACIONES

Las siguientes son nuestras recomendaciones para los proyectos que se vayan a implementar:

1. No sugerimos que para el desarrollo de un determinado trabajo de acceso a bases de datos a través Internet, se utilice una u otra técnica o metodología esto debido a que las circunstancias o variables involucradas en el esquema de desarrollo puede contener aspectos muy diversos y radicales, lo cual conlleva a discernir acerca de cual o cuales herramientas utilizar.

2. En lo posible se debe conformar un equipo de trabajo interdisciplinario o delegar funciones para las diferentes etapas del proyecto, pues es una aplicación la que se implementará, y su eficiencia y eficacia se verá afectada desde las etapas previas a la implementación.

3. Se debe escoger la plataforma computacional de desarrollo, de acuerdo a:

    - Costos.
    - Servidores Web soportados.
    - Herramientas de programación y desarrollo Web soportadas o facilitadas.

4. Se debe escoger el gestor de base de datos ideal para el tipo de proyecto. Se deben tener en cuenta características tales como:

- Costos.
- Tamaño y aplicación del proyecto.
- Soporte a la arquitectura Cliente / Servidor.
- Manejo de conexiones persistentes.
- Manejo de imágenes o graficas (si se va a manejar un volumen considerado de estas).
- Soporte al SQL estándar, aun si el gestor posea su propio dialecto.
- Soporte e interfaces clientes (controladores) para el API nativo que se puedan necesitar en la aplicación tales como: JDBC, ODBC, OLEDB, ADO y demás.
- Interfaces y/o soporte para lenguajes que se vayan a utilizar tales como: ASP, PHP, Perl, Java, C/C++, TCL/Tk.
- Características de seguridad (SSL u otros).
- Soporte comercial.

1. Se debe escoger la metodología o técnica de acceso, teniendo en cuenta factores tales como:

- Costos.
- Plataformas computacionales soportadas.

- Conocimientos previo de lenguajes para programación de aplicaciones Web.
- Comprensión y manejo de las interfaces API o interfaces clientes del gestor.
- Lógica de aplicación y características del proyecto.
- Tamaño del proyecto, este permite escoger la técnica o metodología a seguir, por ejemplo: creación de las aplicaciones digitando el código, utilizando aplicaciones de servidor o servidores de aplicación.

Para grandes proyectos no es aconsejable, crear las aplicaciones utilizando la programación pura (digitar el código de las soluciones). Es recomendable que para grandes proyectos se empleen las utilidades que permiten de una manera interactiva a través de asistentes implementar el código de las aplicaciones de servidor. Estas utilidades poseen un *IDE* que le facilitan al desarrollador integrar las bases de datos, con los formularios y crear el código fuente de las aplicaciones de una manera eficiente. Claro esta que en ciertas circunstancias se debe programar usando el código puro para lograr los objetivos planteados.

Las técnicas que se evaluaron en este estudio son solo una muestra representativa de los métodos y técnicas existentes en el mercado. Por lo tanto al no poderse abarcar todo el campo, se deja abierta la puerta a que futuras investigaciones afines, lo complementen y amplíen.

Tales técnicas incluyen nuevas tecnologías como **Java**, **CORBA**, **COM+**, **.NET**, que no fueron estudiadas, debido a que se escapan al alcance de este trabajo de grado; pero logran calificar para obtener estudio separado, debido a su macro naturaleza: completas, complejas y robustas.

Nos permitimos sugerir que para futuros proyectos de grado se tengan como posibles desarrollos los siguientes temas:

- Estudio de los métodos para acceso a bases de datos distribuidas a través de Internet.
- Desarrollo e implementación de una herramienta integral para accesar bases de datos utilizando tecnologías multidisciplinarias.
- Aspectos relacionados con la seguridad de las transacciones sobre bases de datos a través del Web.
- Estudio y Aplicación de tecnologías basadas en tecnología Java: *JavaBeans empresariales, Servlets, JDBC, JSP*. Para desarrollo de soluciones de *e-commerce* y *e-bussines.*
- Metodologías de acceso a los sistemas de información mediante la tecnología *WAP.*

BIBLIOGRAFÍA

DELGADO GARRON, Alberto. Descubre Internet Information Server 4. 1 *ed*. Madrid : Prentice Hall, 1998. 597 p.

DOBSON, Rick. Programación avanzada con Microsoft Access 2000. 1 *ed*. Madrid : McGraw-Hill, 2000. 537 p.

INSTITUTO COLOMBIANO DE NORMAS TÉCNICAS Y CERTIFICACIÓN. Tesis y otros trabajos de grado. Bogotá : ICONTEC., 1996. 132 p. NTC. 1486.

Ó FOGHLÚ, Mícheál. Perl 5 : Soluciones instantáneas. México : Prentice Hall, 1997. 341 p.

SUSE. SuSE Linux 7.0 : Instalación, Configuración y primeros pasos. 1 ed. Nürnberg : SuSE GmbH, 2000. 650 p.

TACKETT, Jack y BURNETT, Steven. Edición Especial Linux 4ª Edición. 4 ed. Madrid : Prentice Hall, 2000. 1110 p.

ROWE, Jeff. Creación de servidores de bases de datos para Internet con CGI. Madrid : Prentice Hall, 1995. 395 p.

Información relacionada en Internet:

Lista de etiquetas HTML: Una referencia a los elementos actualmente soportados por HTML, incluidas las extensiones HTML oficiales.
http://utopia.knoware.nl/users/schluter/doc/tags/index.html

Microsoft MSDN Online: La versión en línea del la biblioteca *Microsoft Developer Network(MSDN)*.
http://msdn.microsoft.com/library/default.asp

Interfaces clientes de bases de datos de Microsoft: Este sitio contiene la información relacionada con los ultimas noticias de las tecnologías de acceso a datos de *Microsoft.*
http://www.microsoft.com/data/default.htm

W3C Consortium: Este es el sitio del *World Wide Web Consortium (W3C),* contiene desarrollos de tecnologías relacionadas con el *Web*.
http://www.w3.org

Recursos CGI: innumerables recursos para CGI en todos los campos *http://www.davecentral.com/projects/* , directorio *databases*

Gestores de bases de datos

- Oracle: *http://www.oracle.com*
- MySQL: *http://www.mysql.com*
- Sybase: *http://www.sybase.com*
- Postgres: *http://www.postgresql.org*
- InterBase: *http://www.interbase.com*
- Ingres: *http://www.cai.com/products/ingres.htm*
- FrontBase: *http://www.frontbase.com*
- msql: *http://www.hughes.com.au/software/msql2/*
- Informix: *http://www.informix.com/*
- Access: *http://www.microsoft.ttd.es/spain/support/*

Lenguajes de *scripting,* programación y APIs:

- Java/servlets: *http://java.sun.com/products/servlet/*
- PHP: *http://www.php.net, http:// www.zend.com*
- Perl: *http://www.cpan.org , http:// www.activestate.com.*
- Python: *http://www.python.org*
- TCL/Tk: *http://www.etsimo.uniovi.es/ tcl/*

Interfaces Web-Bases de datos:

- ODBiC*: http:// www.odbic.com*
- OpenX: *http://www.openx.ca*
- Whizbase: *http://www.whizbase.com*
- Foxweb: *http://www.foxweb.com*

Servidores de aplicación:

- WebObject, Apple Computer: *http:// www.apple.com/web objects*
- Coldfusion Studio, Allaire Corporation: *http:// www.allaire.com*
- Sapphire/Web, Bluestone Software: *http://www.bluestone.com*
- Tango, Prevasive Software: *http://www.pervasive.com*
- Application Server, SilverStream Software: *http:// www.silverstream.com*
- Jade/Business Logic Server, Vision Software: *http://www.vision-soft.com*
- Enterprise Application Server (EAS), Sybase: *http:// www.sybase.com*
- Novera, TSI Software: *http:// www.novera.com*
- WebSphere, IBM: *http:// www.ibm.com/software/webservers/appserv/*
- Enhydra java/Extensible Markup Language Application Server, Enhydra: *www.enhydra.org*

ANEXOS

ANEXO A

MANUAL DE USUARIO DEL SITIO METODOLOGÍAS PARA ACCESAR BASES DE DATOS A TRAVÉS DE SITIOS WEB

CONTENIDO

LISTA DE FIGURAS

1. MANUAL DE USUARIO DEL SITIO METODOLOGÍAS PARA ACCESAR BASES DE DATOS A TRAVÉS DE SITIOS WEB

1.1 OBJETIVO

El sitio Web pretende ofrecer a la comunidad universitaria, un medio mediante el cual pueda obtener información acerca de las tecnologías existentes que permiten acceder a los datos contenidos en gestores de bases de datos relacionales a través de contenidos Web.

1.2. CONTENIDO

La pagina principal esta constituida por tres marcos a saber:

- ***Superior***
- ***Derecho***
- ***Izquierdo***

***Superior*:** contiene el titulo del proyecto de grado.

Izquierdo: posee el menú de navegación constituido por siete enlaces o botones:

- *Home:* muestra la pagina principal.
- *Introducción:* contiene el origen, los antecedentes y objetivos del proyecto.
- *Generalidades:* contiene información básica sobre los temas relacionados con el proyecto.
- *Técnicas:* contiene la información técnica de las metodologías evaluadas. Se puede acceder a la información especifica a través de los enlaces.
- *Evaluación:* contiene los resultados individuales de las evaluaciones de las metodologías probadas.
- *Resultados:* contiene las conclusiones y recomendaciones.
- *Recursos:* contiene enlaces a otros sitios relacionados con la temática tratada.

Derecha: *muestra el contenido a usuario.*

La Figura 1, muestra la pantalla de presentación de la pagina, que se puede obtener con pulsar sobre el botón *Home* (inicio).

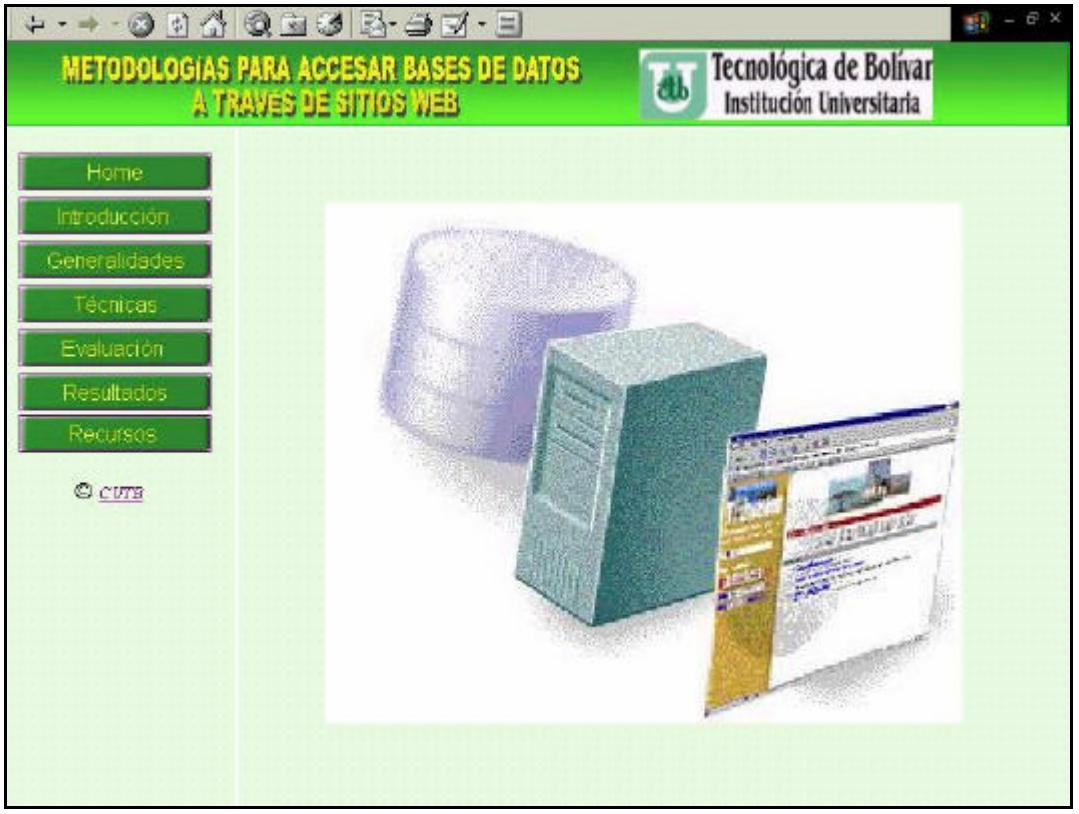

Figura 1. Pagina principal

ANEXO B

RELACIÓN DE TECNOLOGÍAS CLIENTES DE BASES DE DATOS DE MICROSOFT

RELACIÓN DE TECNOLOGÍAS CLIENTES DE BASES DE DATOS DE MICROSOFT

Las tecnologías *Microsoft* clientes de bases de datos y la relacion entre estas, son mostradas en la siguiente figura:

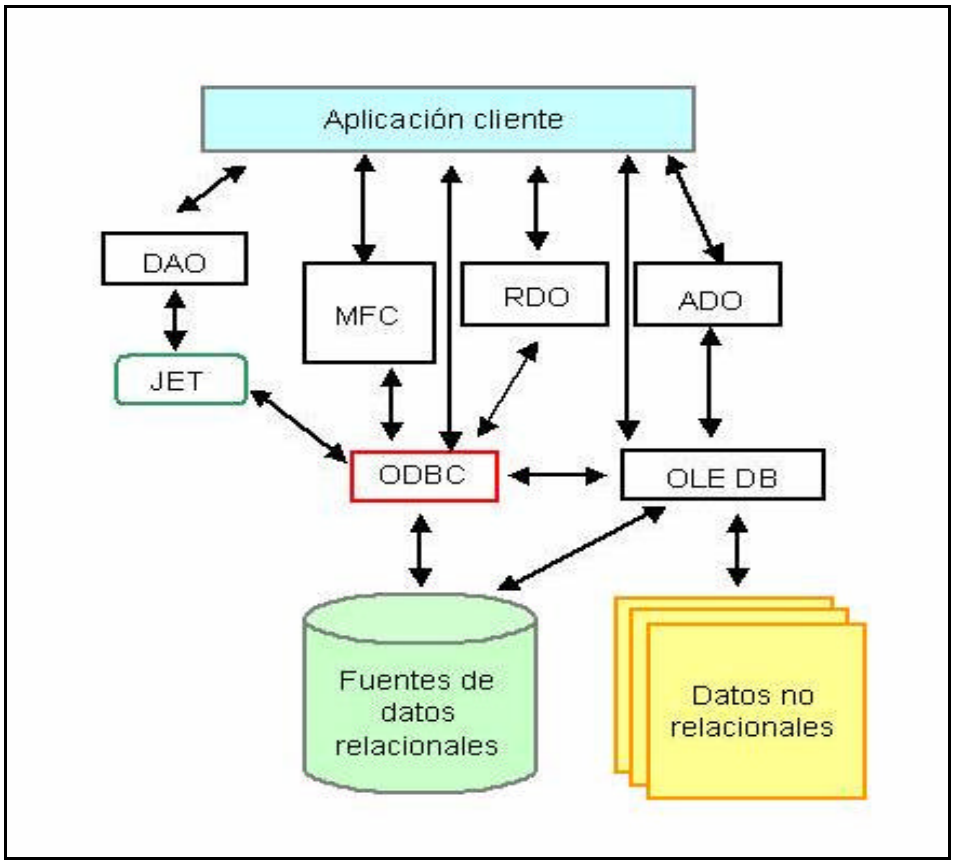

Figura 1. Relación funcional de las interfaces clientes de bases de datos