UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÍA

FUNDAMENTOS DE LA INTEGRACIÓN DE BASES DE DATOS HETEROGÉNEAS

TESIS

PRESENTADA A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

LIGIA MARÍA PIMENTEL CASTAÑEDA

AL CONFERÍRSELE EL TÍTULO DE
INGENIERA EN CIENCIAS Y SISTEMAS

GUATEMALA, OCTUBRE DE 1,997

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÍA

HONORABLE TRIBUNAL EXAMINADOR

Cumpliendo con los preceptos que establece la Ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de tesis titulado:

FUNDAMENTOS DE LA INTEGRACIÓN
DE BASES DE DATOS HETEROGÉNEAS

tema que me fuera asignado por la Coordinación de la carrera de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería, con fecha 5 de septiembre de 1,996.

Ligia María Pimentel Castañeda

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÍA

MIEMBROS DE LA JUNTA DIRECTIVA

DECANO	Ing. Herbert René Miranda Barrios
VOCAL 1ro.	Ing. Miguel Ángel Sánchez Guerra
VOCAL 2do.	Ing. Jack Douglas Ibarra Solórzano
VOCAL 3ro.	Ing. Juan Adolfo Echeverría Méndez
VOCAL 4to.	Br. Víctor Rafael Lobos Aldana
VOCAL 5to.	Br. Wagner Gustavo López Cáceres
SECRETARIA	Inga. Gilda Marina Castellanos de Illescas

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Julio Ismael González Podszueck
EXAMINADOR	Ing. Julio Roberto Marroquín Duarte
EXAMINADOR	Ing. Francisco Javier Guevara Castillo
EXAMINADOR	Dr. Raúl González de Paz
SECRETARIO	Ing. Francisco Javier González López

Guatemala, 5 de septiembre de 1,997.

Ing. Jorge Luis Álvarez Mejía
Coordinador de la carrera de Ingeniería
en Ciencias y Sistemas

Ingeniero Álvarez:

Por medio de la presente me permito hacer de su conocimiento que he procedido a revisar el trabajo de Tesis titulado **FUNDAMENTOS DE LA INTEGRACIÓN DE BASES DE DATOS HETEROGÉNEAS**, elaborado por la estudiante **Ligia María Pimentel Castañeda.**

En mi calidad de asesor, he analizado el contenido, así como las conclusiones y recomendaciones expuestas y, a mi juicio, el mismo cumple con los objetivos propuestos para su desarrollo.

Sin otro particular, me suscribo de usted.

Atentamente,

Ing. Francisco Javier Guevara Castillo
No. Colegiado 3211

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

Escuelas de Ingeniería Civil, Ingeniería Mecánica Industrial, Ingeniería Química, Ingeniería Mecánica Eléctrica, Escuela Técnica, Ingeniería en Sistemas Ingeniería Electrónica y Escuela Regional de Ingeniería Sanitaria y Recursos Hidráulicos.
Apartado Postal 217-1-01-907, Guatemala
Ciudad Universitaria, Zona 12
Guatemala, Centroamérica

Guatemala,
8 de septiembre de 1,997.

Ingeniero
Jorge Luis Álvarez Mejía
Coordinador, Carrera de
Ingeniería en Ciencias y Sistemas
Universidad de San Carlos de Guatemala

Ing. Jorge Luis Álvarez Mejía

Por este medio hago del conocimiento que he procedido a revisar el trabajo de Tesis titulado **FUNDAMENTOS DE LA INTEGRACIÓN DE BASES DE DATOS HETEROGÉNEAS**, elaborado por la estudiante **Ligia María Pimentel Castañeda**.

En mi calidad de revisor, he analizado el contenido así como las conclusiones y recomendaciones expuestas. Después de haber hecho las modificaciones pertinentes, dejo constancia de mi aprobación del mismo.

Sin otro particular, me suscribo,

Atentamente,

Ing. Jorge Luis Álvarez Mejía
COORDINADOR CIENCIAS Y SISTEMAS

UNIVERSIDAD DE SAN CARLOS
CIENCIAS Y SISTEMAS
FACULTAD DE INGENIERIA

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

Escuelas de Ingeniería Civil, Ingeniería Mecánica Industrial, Ingeniería Química, Ingeniería Mecánica Eléctrica, Técnica y Regional de Post-grado de Ingeniería Sanitaria.

Ciudad Universitaria, zona 12
Guatemala, Centroamérica

Guatemala,
17 de septiembre de 1,997
REF.: CS.075.97

Ingeniero
Herbert René Miranda Barrios
Decano, Facultad de Ingeniería

Señor Decano:

Atentamente me dirijo a usted, para informarle que después de conocer el dictamen del Asesor del trabajo de tesis del estudiante LIGIA MARIA PIMENTEL CASTAÑEDA, titulado FUNDAMENTOS DE LA INTEGRACION DE BASES DE DATOS HETEROGENEAS, procedo a la autorización del mismo.

Atentamente,

"ID Y ENSEÑAD A TODOS"

Ing. Jorge Luis Alvarado Mejía
COORDINADOR
INGENIERIA EN CIENCIAS Y SISTEMAS

UNIVERSIDAD DE SAN CARLOS
CIENCIAS Y SISTEMAS
FACULTAD DE INGENIERIA

JLAM/edj

c.c. Archivo

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

Escuelas de Ingeniería Civil, Ingeniería Mecánica Industrial, Ingeniería Química, Ingeniería Mecánica Eléctrica, Técnica y Regional de Post-grado de Ingeniería Sanitaria.

Ciudad Universitaria, zona 12
Guatemala, Centroamérica

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la autorización por parte del Coordinador de la Carrera de Ingeniería en Ciencias y Sistemas, al trabajo de tesis titulado FUNDAMENTOS DE LA INTEGRACION DE BASES DE DATOS HETEROGENEAS, presentado por el estudiante universitario LIGIA MARIA PIMENTEL CASTAÑEDA, procede a la autorización para la impresión de la misma.

IMPRIMASE

Ing. Herbert René Miranda Barrios
DECANO

UNIVERSIDAD DE SAN CARLOS
DECANO
FACULTAD DE INGENIERIA
DE GUATEMALA

Guatemala, octubre de 1,997

ACTO QUE DEDICO

A Dios:

Mi luz y mi salvación.

A mis padres:

Jorge Mario Pimentel M. y Asunción Castañeda de Pimentel. Gracias por su esfuerzo y sacrificios para forjar mi camino como profesional.

A mi socio de la vida:

Carlos Guillermo Marroquín Morales gracias por tu amor, tu apoyo y por tu motivación para dar siempre lo mejor de mí.

A mis hermanas:

Silvia Regina Pimentel Castañeda y Nancy Marlene Pimentel Castañeda con amor especial.

A mis amigos de Assist:

Maritza, Luis Fernando, Luis Emilio, Francisco, Marlon y Donald. Gracias por darme un lugarcito entre ustedes, espacio de disco para mi tesis y por muchos momentos agradables.

Al Lic. David Álvarez:

Gracias por tu orientación en mi investigación y por ayudarme a dar forma a este documento.

A Álvaro Medrano:

Gracias por ser un buen amigo.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

GLOSARIO

ANSI. Siglas en inglés del Instituto Nacional Americano de Normas

Conectividad. Un sistema posee conectividad si puede comunicarse con otros sistemas en tiempo real.

Datawarehouse. Se refiere a un almacén de datos que usualmente se almacena en una base de datos multidimensional.

DMBS. Siglas en inglés para utilizadas para referirse al sistema manejador de la base de datos.

Hardware. Todos los componentes físicos de un sistema de computadoras (incluye la unidad de proceso -CPU-, dispositivos de almacenamiento, memoria, tarjetas para la interconexión de las computadoras).

Middleware. Conjunto especial de componentes de software dedicados a la integración de sistemas.

Modularidad. Se refiere a la propiedad que un sistema de poder dividirse en módulos.

Optimización. En el lenguaje de bases de datos, es el proceso de analizar un query con el objetivo de obtener un nuevo query que presente los mismos resultados que el query inicial, pero con el consumo mínimo de recursos.

Portabilidad. Una aplicación posee portabilidad si puede ser instalado en diferentes sistemas operativos (y ejecutar sin problemas en los mismos).

Query (queries). En inglés, el término Query significa Consulta. En el lenguaje de bases de datos, es un concepto más amplio, que incluye, además de consultas, la modificación, inserción y eliminación de entradas de una tabla.

Relacional. Se refiere a las bases de datos que son percibidas por los usuarios como tablas y relaciones entre ellas.

Replicación. Un sistema tiene replicación si un conjunto de datos se puede representar en el nivel físico mediante varias copias almacenadas (réplicas) en muchos sitios distintos.

Software. El conjunto de programas que se ejecutan en una computadora (incluye el Sistema Operativo).

INTRODUCCIÓN

En las décadas de los 60's y 70's, los sistemas computacionales y las aplicaciones de bases de datos estaban basados en sistemas centralizados, es decir, existía un sistema central único. En esta época manejar un sistema de base de datos era bastante caro y la mano de obra requería demasiada especialización. La década de los 80's la tecnología de los microprocesadores evolucionó haciéndose más barata y fácil de adquirir y trajo como consecuencia la proliferación de las computadoras personales (PC). Esto a su vez permitió a las organizaciones comenzar a automatizar sus procedimientos y desarrollar un conjunto de pequeños sistemas independientes y no integrados en las corporaciones.

En la actualidad, las grandes empresas necesitan unificar la información que maneja cada departamento o núcleo de la organización (en sistemas diferentes) para consolidar sus datos. Estas necesidades, junto con las exigencias de arquitecturas abiertas, suministradas por múltiples proveedores y de acceso económico, están conduciendo la tecnología hacia una arquitectura de sistemas abiertos, en la cual se establecen estándares y normas de diseño de software, hardware y comunicaciones. Las bases de datos juegan un papel muy importante en esta evolución de la tecnología, en donde el objetivo principal es tener acceso a datos que se encuentran en dos (o más) sistemas de bases de datos independientes, con diferente DBMS, y que la información obtenida sea semánticamente correcta y representativa.

Un sistema de múltiples bases de datos provee acceso integrado a DBMS heterogéneos e independientes en un sistema donde la información se encuentra repartida. El principal problema en los sistemas de bases de datos múltiples, es la identificación de datos semánticamente correctos provenientes de diferentes DBMS. Es decir, poder extraer información de una base de datos y que tenga un significado en otra, que sea coherente para ambas.

Para satisfacer las necesidades de interpretación semántica entre manejadores de bases de datos hay varias tendencias y se ha buscado crear estándares en los aspectos más importantes de la representación y manipulación de la información. La Organización Internacional de Estándares (ISO: en inglés, *International Standard Organization*) ha propuesto metodologías para ello, y los fabricantes de manejadores de bases de datos, por su lado, han planteado sus especificaciones para el manejo de los datos y de la información en bases de datos múltiples y heterogéneas (Especificación XOpen/XA).

Así pues, se desarrollaron los conceptos de identificación semántica, lenguajes para definición y manipulación de bases de datos heterogéneas, estándares en los lenguajes de consulta (SQL), los módulos para desarrollo de aplicaciones de manejo y consulta de bases de datos y los módulos de mediación (mediadores) en los cuales uno de los más conocidos es el ODBC, que fundamentan el manejo de bases de datos heterogéneas.

La presente tesis contempla el problema de la comunicación entre bases de datos relacionales múltiples y las opciones que existen para lograr una vista unificada de los datos. Presenta los conceptos fundamentales de la comunicación de bases de datos heterogéneas y una descripción de las categorías de productos para la comunicación semántica entre manejadores de bases de datos heterogéneos.

1 DEFINICIONES GENERALES

En este capítulo se describen los conceptos que fundamentan el contenido de esta tesis. En todo el texto se supone que el lector cuenta con conocimientos generales de computación, especialmente en el área de sistemas de información y bases de datos. Sin embargo, para unificar las ideas respecto al tema, se describen definiciones básicas de bases de datos, beneficios del uso de un sistema de bases de datos sobre un sistema de archivos, tipos de manejadores de bases de datos. Con estos conceptos definidos, se delimitan algunas bases sobre las bases de datos distribuidas, bases de datos múltiples, bases de datos heterogéneas, esquemas cliente/servidor y finalmente, el papel que juegan las bases de datos en los sistemas abiertos.

1.1 Conceptos de bases de datos

Una base de datos no es más que un conjunto de datos persistentes (registros de datos) utilizados de una u otra manera por las organizaciones. Un sistema de base de datos es un conjunto de elementos computarizado con la función de mantener esta información y hacer que esté disponible cuando se necesite. Un sistema de base de datos se compone de tres elementos básicos: los datos, el hardware y el software.

Es importante distinguir entre datos e información. Se define **dato** como una unidad que describe una característica (número, caracteres, imágenes, sonidos) o un suceso que describe algo. Un dato, en el ámbito de la computación, es un conjunto de bytes que se almacena en la computadora y tiene un valor para el usuario. **Información** es un conjunto de datos relacionados de alguna manera y que representan algún conocimiento a quien lo recibe.

En términos generales, la información en una base de datos estará integrada de forma natural y podrá ser compartida. El hecho de estar integrada se refiere a que, aisladamente, los datos pueden ser almacenados en archivos independientes y redundantes. En una base de datos, cada elemento de información es considerado como parte de un todo y la posible redundancia es eliminada (en el proceso de normalización). Por otro lado, la información podrá ser compartida puesto que los datos podrán ser utilizados por diferentes usuarios concurrentemente, como consecuencia de la integración de los datos.

El conjunto de componentes físicos *(hardware)* del sistema de base de datos está constituido por unidades de almacenamiento secundario (generalmente **discos duros**), otros dispositivos de entrada/salida, controladores de dispositivos, canales de entrada/salida, y el procesador y la memoria principal necesaria para la ejecución del software de la base de datos.

Todo el software o conjunto de programas que se utilizan como interfaz entre los usuarios y la base de datos física (la información tal y como está almacenada en realidad) se conoce como el **manejador de bases de datos** o el sistema de administración de bases de datos (*DBMS*, del inglés *Database Management System*). El DBMS es quien realiza todas las operaciones de mantenimiento y consulta solicitadas por los usuarios, siendo un

intermediario entre los diferentes niveles, el nivel físico (almacenamiento en la computadora) y el nivel de usuario, también llamado externo, en el cual los datos representan información.

Con el objetivo de simplificar la interacción de los diferentes tipos de usuarios con el sistema de bases de datos, se definen tres niveles de abstracción, a cada uno de los cuales corresponde un esquema o una noción de definición de tipo en el lenguaje de programación:

1. **Nivel físico:** describe cómo se almacenan realmente los datos. A este nivel le corresponde el esquema físico.

2. **Nivel Conceptual:** nivel intermedio. Describe qué datos son realmente almacenados en la base de datos y las relaciones que existen entre los datos. Aquí se describe la base de datos en un número pequeño de instrucciones sencillas. A este nivel corresponde el esquema conceptual.

3. **Nivel de visión:** es el nivel más alto de abstracción y describe solamente parte de la base de datos completa. Cada tipo de usuario tiene una visión del sistema, según la parte de la base de datos que le interesa conocer. A este nivel corresponde un esquema externo.

La figura 1 resume la arquitectura de un sistema de base de datos.

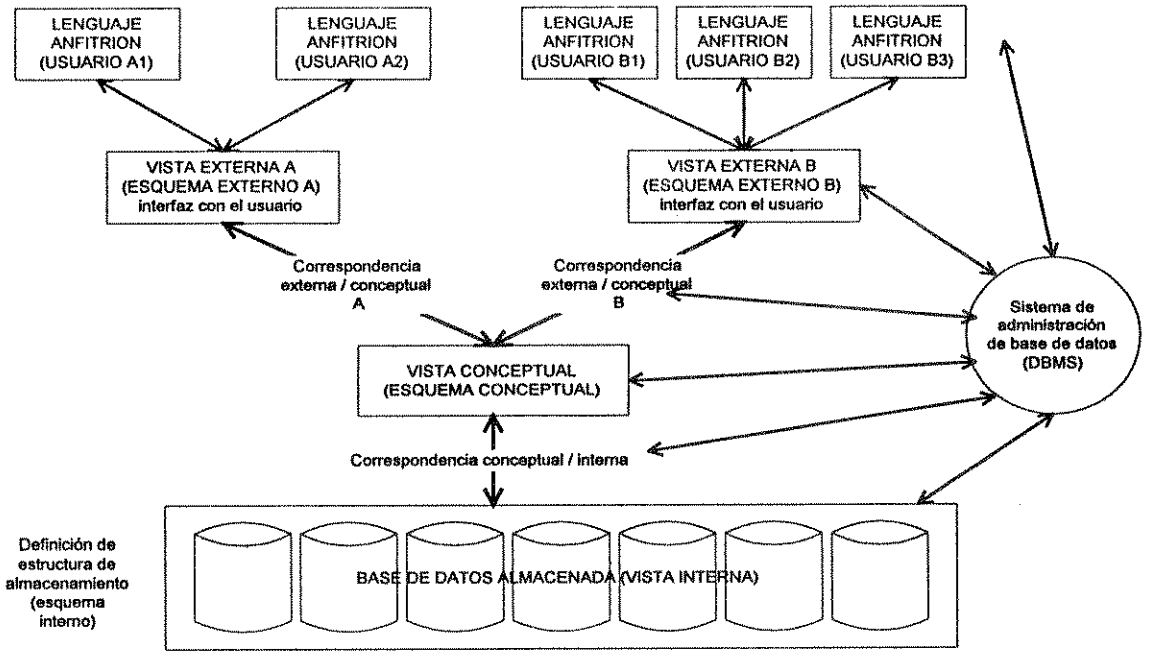


Figura 1: Arquitectura de un DBMS

1.2 Beneficios del uso de sistemas de bases de datos

La utilización de un sistema de bases de datos para el manejo de la información de una empresa tiene las siguientes ventajas sobre el uso de sistemas de archivos no integrados de forma natural:

1. **Se reduce la redundancia:** en un sistema de archivos no integrados, es muy posible que exista más de un archivo que contiene elementos de datos que representen lo mismo en diferente formato, es decir, puede haber información redundante. Esto aumenta los costos de almacenamiento y de acceso, así como puede producir inconsistencia. Una base de datos debe ser diseñada de modo que se elimine (en el mayor grado posible) la redundancia.
2. **Se evita la inconsistencia de los datos:** cuando hay redundancia, se incrementa la probabilidad de que exista inconsistencia. La inconsistencia se produce cuando tenemos duplicidad en los datos de diferentes archivos y las actualizaciones no se hacen de manera controlada (es decir, que si se modifica en registro en un archivo, se modifique obligatoria y casi simultáneamente el otro registro). Eliminar o reducir la redundancia tiene como consecuencia directa la reducción de la inconsistencia.
3. **Los datos pueden compartirse:** para mejorar el rendimiento global del sistema y obtener tiempo de respuesta más rápido, es deseable que múltiples usuarios puedan tener acceso para consulta y actualizaciones a los datos en forma concurrente. Un sistema de bases de datos tiene mecanismos para permitir esta utilización de la información y asegurar la consistencia de los datos.
4. **Se refuerzan estándares:** definir un estándar para los formatos de los datos almacenados es especialmente deseable como una ayuda al intercambio de los datos o a la migración de datos entre sistemas, así como para el entendimiento de la información generada a partir de los datos. Esto a su vez hace más barato y más sencillo el mantenimiento de las aplicaciones.
5. **Puede aplicarse restricciones de seguridad:** un sistema de bases de datos permite crear niveles de usuario correspondientes a niveles de acceso a los datos, y de esta manera, proteger la información.
6. **Puede mantenerse la integridad:** Como consecuencia de la reducción de la redundancia y de la inconsistencia, es más sencillo aplicar restricciones a cada valor almacenado en la base de datos (en inglés *constraints*), para validar cada ingreso y/o modificación de los elementos dato, y hacer estas validaciones independientes de los programas de aplicación.
7. **Pueden balancearse los requerimientos conflictivos:** el diseño de un sistema de bases de datos refuerza el análisis de la empresa como un todo, agrupando los requerimientos de los diferentes usuarios. Esto permite pulir el sistema a un nivel global con el objetivo de buscar el mayor beneficio para todos los niveles de la organización.
8. **Se refuerza la independencia de los datos:** la independencia de los datos se refiere a la capacidad de modificar una definición de un esquema en un nivel sin afectar la definición de un esquema en el nivel superior. Hay dos niveles de independencia: física, que implica poder modificar el esquema físico sin provocar que se tengan que escribir de nuevo las aplicaciones; e independencia lógica, que es la capacidad de poder modificar el esquema conceptual sin provocar una modificación en las aplicaciones.

1.3 Tipos de manejadores de bases de datos

Existe una gran variedad de manejadores de bases de datos, entre ellos podemos mencionar los relacionales, jerárquicos, de red, orientados a objetos. Cada proveedor de

DBMS ha diseñado sus representaciones los datos, su tecnología de almacenamiento y la manipulación de los elementos que componen el sistema y los lenguajes de programación. El contenido de este texto, sobre la integración de bases de datos heterogéneas, considera únicamente los manejadores de bases de datos heterogéneas.

Un manejador de bases de datos puede considerarse como relacional básicamente si:

- Los datos son percibidos por el usuario como tablas y nada más que tablas
- Las operaciones que el usuario puede realizar sobre los datos generan nuevas tablas a partir de las anteriores.

Los manejadores de bases de datos relacionales comenzaron a aparecer en el mercado en los últimos años de la década de los '70s y principios de los '80s. Actualmente hay gran cantidad de sistemas manejadores de bases de datos relacionales (*RDBMS*) que pueden correr en cualquier plataforma, los más conocidos actualmente en el mercado guatemalteco son: Informix, Oracle y Microsoft SQL Server.

1.4 Bases de datos distribuidas

Un sistema distribuido en términos generales es un sistema en el cual sus componentes están repartidos en diferentes localidades (máquinas) y en el cual un usuario en una máquina puede obtener acceso a cualquiera de los recursos del sistema, sin importar dónde se encuentran ubicados, de una manera transparente.

Existe una amplia gama de soluciones para compartir información en un sistema distribuido, que varían desde bases de datos distribuidas hasta sistemas interoperables, éstas varían de acuerdo al nivel de integración -acoplamiento- de los sistemas locales al global (integrado). Un sistema fuertemente acoplado significa que las funciones globales tienen acceso a las funciones internas (a bajo nivel) de los DBMS locales. Esto permite un alto nivel de sincronización entre las localidades y procesamiento global bastante eficiente, además, implica que las funciones globales tienen prioridad sobre las locales y, por lo tanto, el DBMS local no tiene control completo sobre los recursos locales. En un sistema levemente acoplado, las funciones globales utilizan las funciones locales a través de la interfaz externa del DBMS. La sincronización global y la eficiencia no son tan altas, pero el DBMS local tiene completo control sobre sus datos y el procesamiento de los mismos.

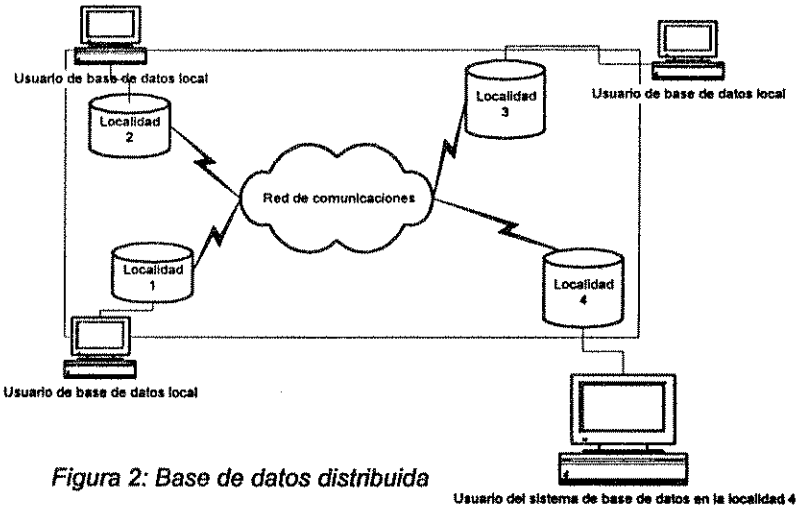


Figura 2: Base de datos distribuida

Formalmente, un sistema de bases de datos distribuido consiste en un conjunto de localidades que se comunican por medio de una red de comunicaciones, en donde cada localidad es en sí misma un sistema de bases de datos independiente y existe un acuerdo entre las diferentes localidades para compartir el acceso a los datos en cualquier parte de la red como si estuviese almacenado todo en una sola localidad.

Los sistemas de bases de datos distribuidos ofrecen los siguientes beneficios:

- **La información puede ser compartida por personas en diferentes localidades:** de la misma manera que las empresas normalmente están distribuidas (existe un departamento financiero, un departamento de recursos humanos, un departamento de producción, un departamento de ventas), sus datos están distribuidos. Un sistema distribuido permite visualizar la información como corresponde lógicamente, es decir, los datos locales son almacenados localmente, y los datos de otras localidades pueden ser consultados eventualmente.

- **Se mejora la relación costo/desempeño:** mantener la información de toda una empresa en una sola máquina o bien en una sola localidad, puede ser muy costoso porque esto demandaría equipo demasiado poderoso (capacidad de procesamiento muy alta, grandes cantidades de memoria primaria y almacenamiento secundario, una infraestructura de comunicaciones compleja y personal especializado). Un sistema distribuido permite manejar áreas relativamente pequeñas con equipo más sencillo, usualmente son computadoras pequeñas y se mejora el tiempo de acceso a los usuarios locales, quienes utilizarán el sistema con mayor frecuencia.

- **Modularidad:** en un sistema distribuido, las interfaces entre las diferentes partes del sistema deben ser diseñadas más cuidadosamente para asegurar una comunicación transparente entre los diferentes componentes del sistema, como consecuencia, generalmente los sistemas distribuidos son más modulares que los sistemas centralizados, están mejor acoplados y tienen un nivel adecuado de cohesión.

- **Capacidad de expansión:** los sistemas distribuidos favorecen el crecimiento proporcional (crecimiento incremental) de las capacidades del sistema.

- **Disponibilidad:** debido a que los sistemas distribuidos tienen cierto nivel de redundancia (replicación de los datos), la información puede estar disponible aunque una localidad sufra una falla.

- **Escalabilidad:** a diferencia de los sistemas centralizados, en un sistema distribuido no tiene una capacidad de crecimiento limitada, es decir, cuando una localidad ya llegó a su punto máximo, es factible crear una nueva.

- **Confiabilidad:** la disponibilidad es uno de los aspectos de la disponibilidad. Un sistema confiable no debe solamente estar disponible cuando hay fallas ocasionales, sino debe ser capaz de recuperarse de las mismas.

Los sistemas de bases de datos distribuidas se caracterizan por los tipos de datos que soportan y el tipo de distribución. Hay dos tipos de bases de datos distribuidas: homogéneas (si soportan únicamente un modelo de datos y un tipo de DBMS) y heterogéneas (si soportan modelos de datos distintos y más de un tipo de DBMS). Cada uno de estos tipos pueden contener cualquiera de estos niveles de replicación de datos:

- **Totalmente replicada:** todos los elementos de datos están presentes físicamente en cada nodo.
- **Particionada:** cada elemento de dato está físicamente presente en uno y únicamente un nodo.
- **Parcialmente replicada:** cada elemento de datos puede existir en cualquier número de nodos del sistema según se defina en la aplicación.

En el siguiente capítulo se describen más a fondo las características de los sistemas distribuidos, como parte fundamental de la integración de bases de datos heterogéneas.

1.5 Bases de datos múltiples

Existen aplicaciones de bases de datos que requieren datos de una gran variedad de bases de datos ya existentes, localizadas en diferentes ambientes de hardware y de software. El término Sistema de bases de datos múltiples (MDBMS: *multiple database management system*) se refiere a una de capa de software adicional superior a los sistemas de bases de datos existentes que se encarga de la manipulación de los datos. Un sistema de bases de datos múltiples proporciona al usuario de la aplicación una visión integrada de los datos sin requerir que exista una integración física de las bases de datos que componen el sistema.

Un sistema de bases de datos múltiples se define como un sistema distribuido que actúa como imagen frontal (*front-end*) a múltiples DBMS locales o está estructurado como un nivel de sistema global (una capa adicional) sobre los DBMS locales. El sistema global provee funcionalidad de bases de datos completa e interactúa con los DBMS locales a través de sus interfaces externas. Aunque cada nodo local debe mantener funcionalidad adicional para comunicarse con el sistema global, los DBMS locales son autónomos. El sistema global provee una forma de resolver las diferencias en la representación de los datos y funcionalidad entre las localidades, y es importante porque la misma información puede estar almacenada en múltiples localidades en diferentes formas.

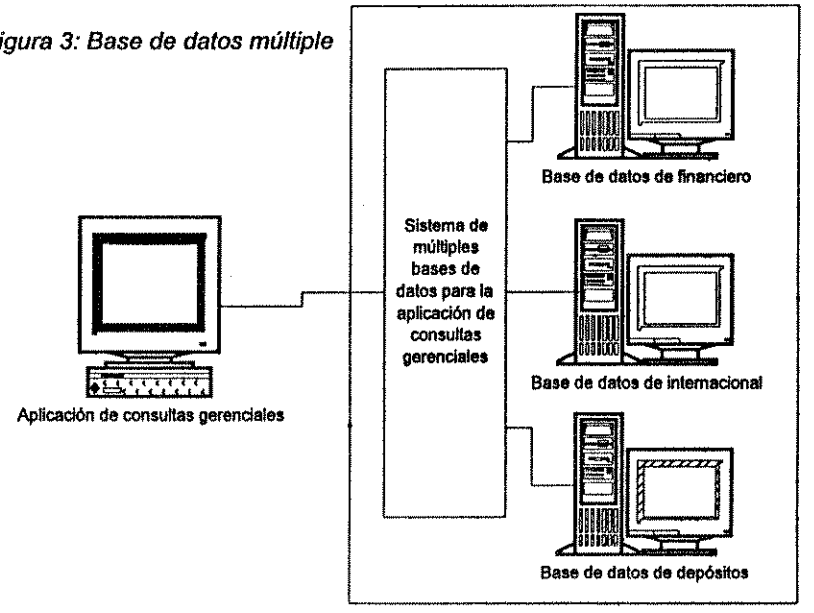
Un MDBMS extiende la capacidad de usuario, permitiendo a los usuarios acceder y compartir datos de una manera transparente (sin necesitar conocer cómo están estructurados internamente los diferentes sistemas de bases de datos). Además, permite unificar en un nivel los programas y procedimientos que operan sobre la aplicación unificada.

Como ejemplo tomaremos un banco. Un banco es negocio bastante complejo y generalmente está organizado en varias áreas: Administración, Contabilidad, Financiero, Depósitos, Tarjeta de Crédito. Debido a la extensión del negocio, se hace muy difícil implementar una base de datos centralizada que tenga la información completa de cada área

funcional, en cambio, se decide que cada una lleve sus propios registros de información (bases de datos) y se establecen normas de comunicación entre ellas. En un momento dado, el banco decide implementar un Sistema de Información Gerencial (SIG), y uno de los requerimientos es poder obtener datos específicos de cada una de las bases de datos de las áreas funcionales.

Hay dos posibles soluciones al problema: una es la integración física de los datos necesitados por la aplicación en una base de datos única. Esto tiene las siguientes desventajas: es caro, no permite el mantenimiento de los datos de manera independiente (actualizaciones en línea), conlleva replicación innecesaria de los datos y requiere un gasto bastante grande en conversión de aplicaciones.

Figura 3: Base de datos múltiple



La otra solución es la integración de los datos requeridos por la aplicación en una base de datos lógica. Esta integración crea una ilusión de un sistema de base de datos único y oculta al usuario todos los aspectos específicos de los diferentes DBMS y diferentes métodos de acceso. Provee a los usuarios un acceso uniforme a los datos contenidos en diferentes bases de datos sin trasladar los datos a la nueva base de datos y sin requerir que los usuarios conozcan la ubicación real de los datos, o las características específicas de cada base de datos y su correspondiente manejador. Este tipo de integración es lo que se conoce como un sistema de bases de datos múltiples, y el concepto comenzó a manejarse a principios de los 80's.

1.6 Bases de datos heterogéneas

Cuando los sistemas de bases de datos múltiples tienen componentes de diferentes tipos (utilizan un modelo de datos diferente - relacional, jerárquico, de red -, tienen diferentes tipos de manejadores, utilizan un lenguaje distinto) se dice que es un sistema de bases de datos múltiples Heterogéneo.

Un sistema de base de datos heterogéneo (de cualquier tipo), debe proveer una ilusión de configuración simple (como si fuera un sistema de bases de datos integrado), para ello se debe utilizar un modelo común de datos. Para seguir un estándar, se utiliza el modelo relacional, utilizando SQL como el lenguaje común de consulta, y existen varias herramientas que permiten la traducción de las sentencias en lenguaje de consultas (*SQL*) a un DBMS no relacional. Otro problema es la traducción semántica entre los diferentes esquemas. Además,

debe tenerse una consideración especial con el manejo de transacciones locales y globales en un sistema de bases de datos múltiples.

1.7 Esquemas cliente/servidor

Los esquemas cliente/servidor son una clase especial de sistemas distribuidos en donde hay dos máquinas (sistemas independientes y autónomos) que se relacionan entre sí, de modo que una tiene función de servidor (*back-end*) y otro tiene función de cliente (*front-end*). Normalmente, en un entorno de bases de datos, el servidor es la máquina que tiene los datos almacenados y se dice que es el **servidor** de bases de datos. El **cliente** tiene los programas de aplicación que manejan los datos.

Hay varios niveles dentro de los esquemas cliente/servidor, dependiendo del nivel de distribución del sistema.

1.8 Sistemas abiertos

Los sistemas abiertos nacieron debido a la necesidad de interoperabilidad. Cuando esta necesidad se hizo clara a principios de los 80s, las organizaciones militares de los Estados Unidos y otras organizaciones como IEEE (siglas en inglés de *Institute of Electric and Electronic Engineering*) e ISO (International Organization for Standardization) empezaron a crear estándares para la fabricación de hardware y de software. Estos estándares permitieron que empezaran a evolucionar los sistemas propietario (en los cuales solamente podía integrarse componentes autorizados por el fabricante) y aparecieron los Sistemas Abiertos.

Actualmente, se dice que un sistema es abierto si provee una manera de integrarse o de comunicarse con otros sistemas, sin importar si son o no del mismo fabricante, basándose solamente en los estándares definidos en común. Las tres características ideales de los sistemas abiertos son la portabilidad, la independencia del proveedor (tanto en software como en hardware y sistemas operativos) y la conectividad (teniendo claro por supuesto, que el objetivo final es la interoperabilidad).

La portabilidad se refiere a poder tomar el sistema operativo o un software de aplicación de una máquina específica (con características de hardware determinadas) y poder ejecutarlo en otra máquina (con características de hardware distintas), o bien tomar un elemento de hardware (como una tarjeta de red, o un módem) y que sea funcional en cualquier máquina.

Un ejemplo muy interesante de la relación de conectividad con interoperabilidad es el sistema telefónico. Podemos hacer una llamada telefónica a Francia y tenemos conectividad física, sin embargo, si no habmamos francés, y la persona que nos contesta no habla español, no habrá comunicación (y, por consiguiente, no habrá interoperabilidad).

Actualmente, hay estándares de sistemas abiertos para la fabricación de hardware, para los sistemas operativos, estándares de comunicación y estándares de bases de datos, entre otros. Estos tienen como objetivo alcanzar los ideales de los sistemas abiertos, aunque no existe a la fecha un sistema totalmente abierto, es importante verificar los estándares que cumple en relación a la solución que necesitamos.

1.9 Sistemas de bases de datos y los sistemas abiertos

Antes de que se empezara a definir los estándares para los sistemas abiertos, era virtualmente imposible tener dos sistemas de bases de datos independientes (cada uno en una máquina) y tener acceso a ambos desde una máquina. Mucho menos podía pensarse en integrarlos los dos sistemas en una sola aplicación.

Gracias a la definición de los sistemas abiertos, ahora puede integrarse dos fuentes de datos preexistentes localizados en ambientes de hardware y de software heterogéneos. Esto puede hacerse integrando las dos bases de datos *lógicamente* en una sola base de datos.

Para ello se han propuesto modelos para la definición de un estándar para los manejadores de bases de datos. Poseer dicho estándar ofrece las siguientes ventajas: portabilidad de aplicaciones a nivel de hardware y de sistema operativo, mejor desempeño en la productividad de los usuarios y programadores de aplicaciones y disminución de los tiempos de entrenamiento, simplificación de la evaluación y selección de los sistemas manejadores de bases de datos, reducción de costos, incrementar la viabilidad de la integración de datos entre diferentes manejadores de bases de datos.

La estandarización de sistemas manejadores de bases de datos debe considerarse desde varios aspectos:

- Análisis de los datos, que comprende el nivel de abstracción o de análisis de los datos (nivel interno, nivel conceptual, nivel externo), y el nivel de descripción de los datos (a nivel de esquemas: datos de la aplicación, esquema de los datos de la aplicación, diccionario de datos, esquema del diccionario de datos, modelo de datos, esquema del modelo de datos).

- Análisis funcional de los datos, que se basa en las operaciones de consulta, de actualización y de transformación de los datos.

- Herramientas de manejo de los datos para el usuario final, para el usuario de desarrollo de aplicaciones, para el administrador.

- Interrelación con el sistema operativo. Deben diseñarse protocolos adecuados para la comunicación del DBMS con el sistema operativo lo suficientemente generales para ser independientes de la plataforma.

1.10 Bases abiertas de datos

Un ambiente de bases de datos, para ser considerado abierto, debe tener las siguientes características:

1. Una interfaz basada en SQL (el método de acceso de datos estándar), para permitir obtener los datos sin las limitaciones de las herramientas e interfaces propietarias.
2. Una forma de acceso a los datos desde una amplia gama de herramientas cliente/servidor comerciales.

3. Capacidad de intercambiar información e interactuar con los ambientes de bases de datos comerciales.

Las bases de datos abiertas se adhieren a dos tipos de estándares SQL: uno que especifica el soporte a lenguaje SQL (ANSI ISO SQL-92) y otro que especifica la interoperabilidad cliente/servidor (ODBC). Cualquier base de datos que no se adhiere a estos estándares se considera una base de datos propietaria, y pueden caer en dos categorías:

- Bases de datos de legado (*"legacy databases"*) que fueron desarrolladas antes del surgimiento de las bases de datos relacionales (entre ellas las jerárquicas, los archivos planos, los archivos secuenciales indexados y las bases de datos navegacionales).

- Bases de datos especializadas a aplicaciones específicas, como las orientadas a objetos, texto y multidimensionales.

Convertir una base de datos propietaria en un sistema abierto significa proveer una interfaz estándar SQL para obtener acceso a ella. Esto requiere dos niveles de abstracción: el primero impone una vista relacional a los datos del sistema propietario (presentarlos en forma relacional), el segundo mapea construcciones de lenguaje SQL a la forma relacional provista por el primer nivel de abstracción.

Para definir una perspectiva relacional es necesario resolver diferencias semánticas en cuanto a la forma cómo los datos están almacenados y como se obtiene acceso a ellos, y con este propósito se utilizan técnicas, que permiten crear una vista relacional de cualquier conjunto de datos en un sistema propietario, así como interfaces relacionales a ellas.

Cuando existe una vista relacional a los datos las interfaces apropiadas, es posible definir construcciones de lenguaje de consulta (SQL) para la forma relacional. La traducción de éstas involucra varios niveles de traducción: análisis sintáctico y validación de la consulta, traducción de la misma a identificadores del sistema propietario, optimación de SQL y definición de un plan de ejecución, ejecución del plan de procesamiento y obtención de los datos.

La tecnología de acceso por SQL al sistema propietario debe, además de proveer todas las interfaces, funcionalidad y optimización soportadas por una base de datos relacional SQL, ser flexible y *portable*[1] a cualquier sistema propietario, en diferentes sistemas operativos, aprovechando las ventajas de cada sistema y aislando las aplicaciones SQL de los aspectos específicos de cada sistema:

---

[1] En computación ser portable se refiere a la capacidad de un sistema de funcionar en diferentes arquitecturas. En este caso, la tecnología de acceso SQL a la que se hace referencia, debería ser abierta para instalarse y poder ejecutarse en más de un sistema operativo.

- Arquitectura de procesamiento.
- Métodos de acceso a los datos.
- Tipos de datos y formatos de almacenamiento de los datos.
- Manejo de transacciones.
- Mecanismos de autenticación de usuarios.
- Detalles del diccionario de datos.
- Portabilidad de plataformas.
- Portabilidad respecto a protocolos de red.

2 FUNDAMENTOS PARA LA INTEGRACIÓN DE BASES DE DATOS

En este capítulo se definirán los fundamentos básicos que deben verificarse para lograr la integración de sistemas de múltiples bases de datos. No es posible separar el concepto de integración de bases de datos del concepto de sistemas de bases de datos distribuidos. Como consecuencia, este capítulo toma sus fundamentos en los sistemas distribuidos en general. En la primera parte se enumeran las reglas básicas de los sistemas distribuidos, se trata el concepto de interoperabilidad semántica, y los requerimientos respecto al manejo de la información y el uso de transacciones en los sistemas distribuidos.

2.1 Componentes de un sistema de información integrado

Un sistema de información integrado se compone de:

1. **Fuentes de datos y conocimientos (servidores):** básicamente, es necesario poseer dos o más sistemas de bases de datos (ubicados en la misma computadora servidora o en distintas) que contengan la información que deseamos integrar. Estas bases de datos pueden ser o no relacionales, aunque nuestro estudio se limitará a ellas.

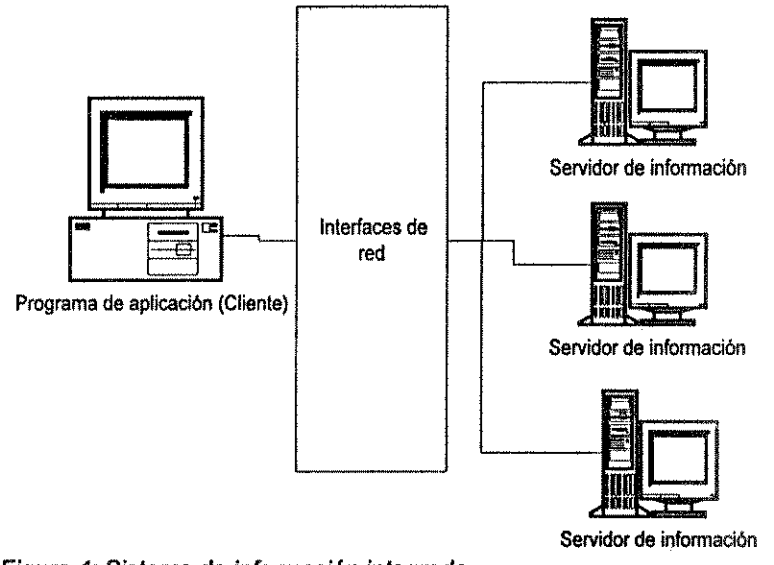


Figura 4: Sistema de información integrado

2. **Programas de aplicación para el usuario (clientes):** dado que el objetivo final es generar nueva información basada en datos tomados de diferentes sistemas preexistentes, es necesario poseer al menos una aplicación para la presentación de ésta. Estos programas deben permitir al usuario interactuar con el sistema activamente (crear sus propias consultas, almacenar los resultados intermedios y almacenarlos para compararlos posteriormente, y utilizar otras herramientas de análisis).
3. **Interfaces de red:** dicho en una forma resumida, debe existir una forma de comunicar todos componentes servidores y clientes, utilizando protocolos de comunicación (idealmente basados en sistemas abiertos).

2.2 Las doce reglas de los sistemas distribuidos

El principio fundamental de los sistemas distribuidos es que debe mantenerse ante el usuario final (usuario de aplicaciones) la ilusión de que es un sistema único

independientemente de la localidad y la representación interna de los datos, es decir, debe presentarse una *Vista Unificada de los Datos.* Este principio se deriva en doce reglas[2] que definen las bases para el entendimiento de un sistema distribuido y constituyen un marco de referencia para la definición funcional de un sistema distribuido:

1. **Autonomía local:** cada sistema de bases de datos participante en el sistema distribuido debe ser un sistema autónomo por sí mismo. Es decir, ningún sistema de base de datos debe depender de otro para su funcionamiento ni debe tener injerencia en el de otro. Además, cada uno debe tener un propietario y administración local de los datos.

2. **Independencia de una locación central:** no debe existir un sitio central que administre el funcionamiento o que proporcione información vital a las locaciones que componen el sistema distribuido, pues esto podría convertirse en un cuello de botella y/o hacer el sistema dependiente de un punto vulnerable a fallas.

3. **Operación continua:** el sistema debe tener la capacidad de permanecer activo (sin apagar) permanentemente, aún mientras se realicen funciones de mantenimiento como agregar localidades, o hacer modificaciones en la instalación de una localidad.

4. **Independencia de localización:** la información debe ser totalmente transparente al usuario de cualquier localidad, sin que tenga que conocer dónde están ubicados los datos. Esto simplifica los programas de los usuarios y sus actividades en la terminal, además hace posible la migración de los datos entre sitios sin invalidar los programas y/o actividades.

5. **Independencia de fragmentación:** un sistema distribuido está fragmentado si se divide en varios fragmentos que contienen la información suficiente para representarlo como un todo, estando cada uno almacenado en una locación física distinta. La fragmentación se realiza para optimar el desempeño, pues si los datos se almacenan en la localidad donde se utilizan con mayor frecuencia, reducen el tráfico en la red y agilizan las consultas. Un sistema distribuido debe presentar los datos al usuario en una vista lógica, como si no estuvieran fragmentados aún cuando lo estén.

6. **Independencia de replicación:** un sistema tiene replicación si una relación o fragmento, está representado físicamente mediante varias copias en diferentes localidades. Esta replicación debe ser transparente al usuario, es decir, el usuario no tiene por qué conocer que el sistema tiene más de una copia de los datos.

7. **Procesamiento distribuido de los queries (consultas):** dado que la información está distribuida, es necesario que al hacer una consulta, independientemente de la localidad donde estén ubicados físicamente los datos, la respuesta sea eficiente y totalmente transparente. Por esta razón, la estrategia que se utilice debe considerar la optimización de los queries, el costo de transmisión de datos en la red y la distribución de las tareas en las que se divide la consulta, y las políticas para el procesamiento de las mismas.

8. **Manejo de transacciones distribuido:** una transacción debe mantener la consistencia de los datos no importando si se realiza en una única localidad o a través de un sistema

---

[2] Data, C.J., "What is a Distributed Database System?", **Relational Database Writings, 1985-1989,** Reading, Mass.:Addison-Wesley, 1990, p. 128.

distribuido. Aspectos muy importantes y de especial atención en un sistema distribuido son el manejador de transacciones (coordinador) de cada localidad, la coordinación de transacciones que involucran más de una localidad, el interbloqueo, la concurrencia y la recuperación. Más adelante se amplía este tema.

9. **Independencia del hardware:** un sistema distribuido debería ser totalmente independiente del equipo que se utilice en cada localidad, presentando al usuario de una localidad una sola imagen del sistema.
10. **Independencia del sistema operativo:** muy relacionado con lo anterior, significa que un usuario del sistema distribuido debe ser totalmente ajeno al sistema operativo de cada localidad.
11. **Independencia de la red:** la estructura de la red de comunicación, su arquitectura y forma de comunicación debe ser transparente al usuario del sistema.
12. **Independencia del manejador de la base de datos:** esto implica que los manejadores de bases de datos de cada localidad manejen todos una única interfaz, para que pueda lograrse una comunicación entre ellos en el contexto de un sistema distribuido. Para lograr esto se han planteado modelos de referencia para la estandarización de los DBMS.

2.3 Manejo del diccionario de datos (metadata) en un sistema de múltiples bases de datos

El manejo de múltiples bases de datos genera un problema relacionado con la definición de las bases de datos que se quieren manejar. Una base de datos se define por su esquema interno y los datos mismos, por lo tanto, los conflictos de integración pueden darse por las diferentes representaciones simbólicas de los conceptos en las bases de datos, y éstos se clasifican en: conflictos de esquema, que resultan del uso de diferentes definiciones de esquema en los componentes, y conflictos de datos, originados por datos inconsistentes. Una posible clasificación[3] para estos conflictos es la siguiente:

I. Conflictos de esquema:

A. Tabla vrs. tabla
  1. Conflictos de tablas en relaciones uno a uno
    a) Nombres de tablas
      a.1) Nombres diferentes para tablas equivalentes
      a.2) Uso del mismo nombre para diferentes tablas
    b) Estructura de tablas
      b.1) Atributos faltantes
      b.2) Atributos faltantes pero implícitos
    c) Restricciones de integridad (*constraints*) a nivel de tabla
  2. Conflictos de tablas en relaciones muchos a muchos

---

[3] Kim, Won, et. al. Classifying Schematic and Data Heterogeneity in Multidatabase Systems. **Computer.** Diciembre, 1991, pp. 12-17.

B. Atributo vrs. atributo
   1. Conflictos de atributos uno a uno
      a) Nombres de atributos
      b) Valores por omisión
      c) Restricciones de integridad a nivel de atributo
         c.1) Variación en los tipos de datos
         c.2) Rrestricciones de integridad del atributo (dominio)
   2. Conflictos de atributos muchos a muchos

C. Tabla vrs. atributo

II. Conflictos de datos

A. Datos incorrectos
   1. Datos ingresados en forma incorrecta
   2. Datos obsoletos

B. Diferentes representaciones de los mismos datos o la misma representación para diferentes datos
   1. Diferentes expresiones
   2. Diferentes unidades
   3. Diferentes precisiones

2.4 Integración de esquemas y heterogeneidad semántica (interoperabilidad semántica)

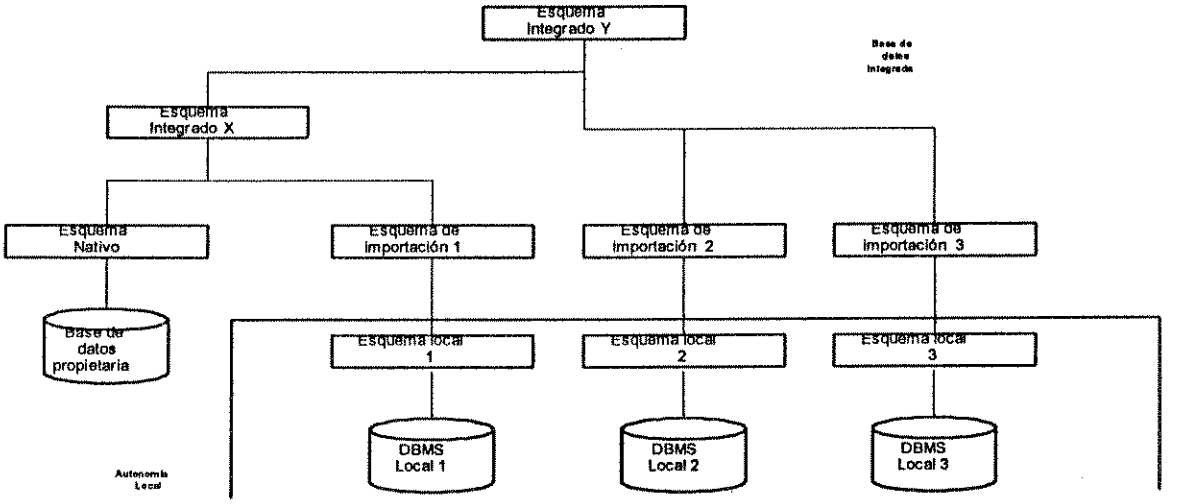
El concepto de integración de esquemas en un ambiente de bases de datos heterogéneas se resume en que debe existir, para un conjunto dado de esquemas de bases de datos, un esquema integrado de manera que cada esquema local pueda considerarse como una vista del esquema integrado.

Esto puede asegurarse de dos formas. Una de ellas es la creación de un esquema global que represente las múltiples bases de datos como una base de datos *virtual* única, que contiene en su definición, las definiciones de las bases de datos que están involucradas. Esta solución tiene como consecuencia la necesidad de considerar aspectos como la resolución de conflictos de nombres cuando datos semánticamente iguales se nombran de forma distinta o cuando datos semánticamente distintos se nombran igual, resolución de conflictos de representación de datos semánticamente idénticos en diferentes bases de datos (por ejemplo, un código que en una base de datos es un string y en otra es un byte), y en forma general, diferencias de estructuras de datos en las diferentes fuentes de datos así como de escalas (en una aplicación se almacena una distancia en metros y en la otra en kilómetros). Estos problemas se pueden resumir en *inconsistencia de dominios*.

La otra forma es crear un esquema que incluye únicamente la porción de cada base de datos individual que la base de datos integrada podrá utilizar, conocidos como esquemas de importación. Esta forma se conoce como el Enfoque Federado (*en inglés: federated database approach*), y se ilustra en la figura.

El modelo relacional y el lenguaje estructurado de consultas (ANSI SQL) se han utilizado para la integración de esquemas, por su simplicidad, adaptabilidad y soporte para

Figura 5: Modelo de Integración de esquemas



amplia variedad de consultas configurables. La integración de esquemas en un sistema federado se hace siguiendo los siguientes pasos (véase figura 6):

1. Traducción de los esquemas de las bases de datos locales definidos en sus modelos nativos (de red, jerárquico, relacional) a esquemas de componentes equivalentes definidos en el modelo relacional.
2. Integración de los esquemas de componentes al esquema federado. En este proceso se genera una vista unificada única de todos los datos de todos los proveedores de datos.
3. Agrupación de los datos en entidades utilizables y definición de operaciones de combinación de datos (*joins*) necesarias y vistas de usuarios. El resultado es el esquema externo de la base de datos.

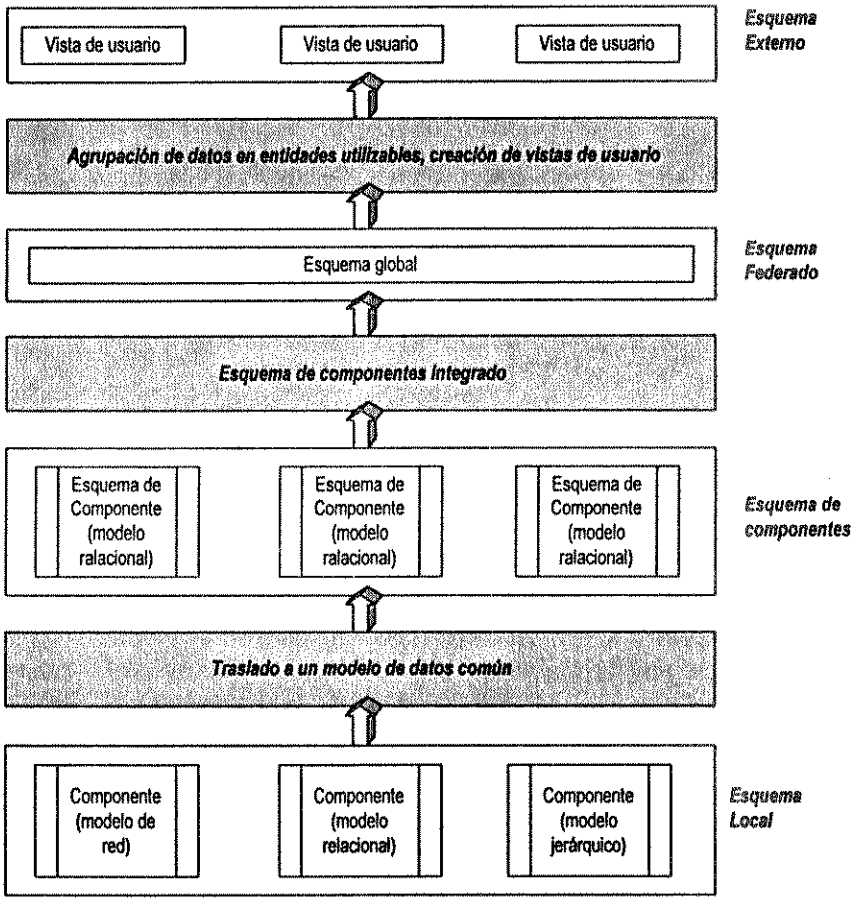


Figura 6: Proceso de integración

2.5 Lenguajes de manejo consulta de bases de datos

La integración de esquemas trae como consecuencia la necesidad de la existencia de un lenguaje único para la definición, manejo y consulta de datos. Es decir, un usuario del sistema múltiple utilizará un lenguaje único para formar construcciones que le permitan utilizar la información del sistema heterogéneo como un único ambiente. Este lenguaje debe permitir a los usuarios combinar datos de las diferentes bases de datos, transformar dinámicamente sus atributos y asignarles nuevos valores, consultar todas las bases de datos con una sola consulta. Estos deberán poseer también herramientas para definir esquemas de importación y definir derechos de acceso a los usuarios.

2.6 Procesamiento de consultas *(Queries)* en un ambiente distribuido heterogéneo

La operación de consulta es la forma por la cual los usuarios pueden solicitar información y obtener datos relevantes de una fuente o fuentes de datos.

El procesamiento de consultas expresado sobre un sistema heterogéneo requiere un mecanismo para la descomposición y traslado de la consulta global a subconsultas para cada base de datos componente, y la reestructuración de los resultados parciales al formato del esquema global. Este mecanismo depende de que las bases de datos que componen el sistema global proporcionen herramientas para procesar consultas externas.

La traducción de consultas a diferentes lenguajes de consulta puede ser muy complejo si cada componente utiliza diferente modelo de datos, por lo que el mecanismo de consulta deberá estar directamente relacionado con el diccionario de datos y la definición del esquema global para la localización de los datos que se desea consultar.

Otro aspecto a considerar es la optimización de las consultas globales, que determina una estrategia para el acceso a los componentes de bases de datos y la combinación de los resultados de las consultas, y debe ser un complemento a los mecanismos de optimización de consultas de cada componente (si existen).

El procesador de consultas globales realiza las siguintes funciones:

1. Chequeo sintáctico y semántico (*parsing*) de la consulta.
2. Validación de los requerimientos para definir si los datos se encuentran en alguno (o algunos de los componentes).
3. Traducción de la consulta original a subconsultas para enviar al procesador de consultas de cada componente.
4. Optimización y generación de un plan de ejecución, analiza cada subconsulta generada apoyandose en la información que le provee cada compente respecto a la cantidad de datos que posee y el costo de recuperación de información en cada uno.
5. Cada subconsulta es enviada a los procesadores locales y las respuestas de cada uno son recopiladas por el procesador global, en una tabla de resultados globales.

6. El procesador global opera los resultados parciales y les da el formato adecuado de acuerdo a la consulta original.

2.7 Manejo de transacciones en un ambiente distribuido heterogéneo

Una transacción es una forma de accesar y actualizar información almacenada en la base de datos, manteniendo la integridad del sistema. Varias transacciones pueden ejecutarse concurrentemente, de modo que el DBMS debe proteger los datos. Si ampliamos esta definición a un sistema de múltiples bases de datos, encontraremos la necesidad de coordinar transacciones que se realizan localmente (en cada DBMS autónomo) y las transacciones a nivel global.

En un sistema de múltiples de bases de datos (heterogéneas o no) debe asegurarse la consistencia global y la anulación de la posibilidad de interbloqueos en el sistema. Para ello, debe existir un coordinador centralizado (debido a que los DBMS individuales son incapaces de coordinar la ejecución de transacciones globales por ser independientes de los otros sistemas).

El coordinador de transacciones debe ser capaz de operar transacciones con un compromiso de dos fases, por lo que es necesario definir un *protocolo* común para los DBMS para el manejo de transacciones (como bloqueo de dos fases *-two phase locking-*, seriabilidad en el tiempo *-time stamp ordering-*, entre otros), específicamente para el bloqueo de registros. Además es necesario que cada DBMS pueda cooperar con el coordinador en un protocolo de compromiso de dos fases mediante operaciones como "Preparar Compromiso" (en inglés *prepare to commit*), de modo que si una transacción está en estado de "Preparar Compromiso" en los sitios S1 y S2, y falla en el sitio S1 antes que se complete, cuando ya fue completada en S2, el coordinador pueda ordenar a los dos sitios volver a las condiciones iniciales (en inglés *Rollback*) y evitar la inconsistencia en el sistema global.

En este caso, las funciones de coordinador pueden no pueden recaer en alguno de los DBMS autónomos que interactúan (S1 o S2), sino en la herramienta integradora, puesto que cada DBMS autónomo no conoce la existencia de los otros DBMS en el sistema de bases de datos múltiples.

El procesador de transacciones en un ambiente heterogéneo tiene la tarea de manejar diferentes protocolos de compromiso y diferentes métodos de control de concurrencia, y supone que ya se ha creado una especificación de metadata común (es decir, el procesador de transacciones manejará un esquema único para el sistema integrado, y está en un nivel superior (o una capa más alta) de la que resuelve los conflictos de la metadata y de la que resuelve los conflictos de comunicación, propios del sistema operativo, (que se explican con mayor detalle en los capítulos siguientes). El manejo de transacciones distribuidas necesita un ambiente cooperativo, es decir, uno que permita intercambiar información de control entre cada uno de los componentes de bases de datos.

2.7.1 Control de concurrencia heterogéneo

Cada DBMS utiliza su propio mecanismo de control de concurrencia (que se refiere al uso de los recursos en forma compartida), y esto introduce varios problemas al manejador de

transacciones herogéneo. El primero de ellos es que diferentes algoritmos de control de concurrencia producen diferentes resultados aún cuando se procesa el mismo conjunto de transacciones en la misma secuencia. El segundo problema es que una transacción global puede no ser serializable[4] aún cuando todas sus subtransacciones lo sean. Por lo tanto, se requiere un mecanismo para garantizar que las subtransacciones de una transacción global puedan ser todas serializadas en el mismo orden. La figura ilustra la serialización de transacciones.

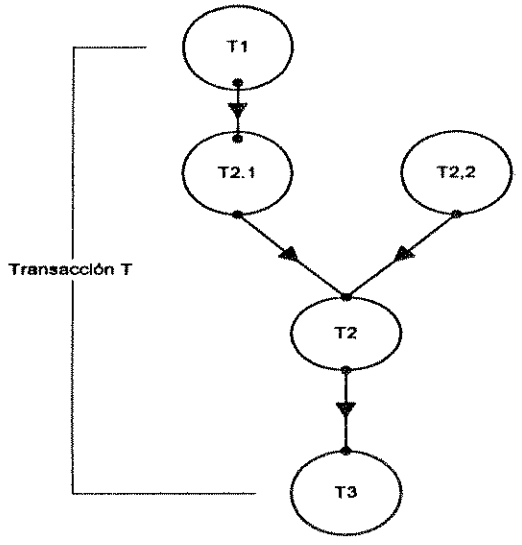


Figura 7: Serialización de una transacción

El objetivo es apoyar el procesador de transacciones global en los mecanismos locales de control de concurrencia para garantizar la serialización y luego agregar un chequeo global para proveer una serialización global. Este método se basa en una estructura de datos llamada elemento-orden (*elementoO*), que es una representación del orden de serialización de transacciones en un componente de base de datos. Si la transacción T_1 precede a T_2, se denota $T_1 \rightarrow T_2$, y por lo tanto, elementoO(T_1) $\leq$ elementoO(T_2), en cada componente de base de datos local.

Un vector-orden (*vectorO*) es la concatenación de elementosO de los componentes de base de datos, entonces, si tenemos los componentes A y B, vectorO(T_1) es (elementoO(T_1^A), elementoO(T_1^B)). El orden de los vectoresO se define estrictamente así: vectorO(T_1) <= vectorO(T_2) si y solo si, para todas las bases de datos componentes X, elementoO(T_1^X) <= elementoO(T_2^X).

Esta definición dice que si el vectorO(T_1) $\leq$ vectorO(T_2), todas las transacciones son serializables en el mismo orden, por lo tanto, puede verificarse la serialización global de una transacción global (*supertransacción*) comparando su vectorO contra los vectoresO de transacciones globales previamente finalizadas (o comprometidas). Mantener un orden total de los vectoresO garantiza la seriabilidad y es independiente de los métodos de control de concurrencia de los componentes.

Mientras pueda especificarse el orden de serialización de los componentes, una base de datos múltiple puede asegurar la serialización de las transacciones globales, y puede componer protocolos de compromiso de dos fases, marcas de tiempo y métodos de control de concurrencia optimistas. Además, permite que una base de datos múltiple pueda tener su propio conjunto de vectoresO, para que pueda ser un componente de una base de datos más general (en una definición recursiva).

---

[4] La serialización se refiere a poder separar una transacción en sus componentes atómicos y procesar primero aquellos que pueden ser operados en paralelo y luego los requieren los resultados de los anteriores para comenzar (es decir, que deben procesarse en serie).

2.7.2 *Recuperación de transacciones en ambientes heterogéneos*

Las transacciones distribuidas homogéneas utilizan un protocolo de acierto (*agreement protocol*) para asegurar que todas las subtransacciones de una transacción global se completen o se reversen en conjunto (*global commit* o *global rollback*). En la arquitectura de una base de datos múltiple, se supone que todas las bases de datos componentes proveen un protocolo de acierto y que todas las subtransacciones participan en estos protocolos de compromiso.

Los protocolos de compromiso se dividen en dos familias: simétricos y asimétricos. Los protocolos asimétricos (como el protocolo de compromiso de dos fases) tienen un coordinador que recopila la información, toma una decisión e informa a los componentes subordinados. Los protocolos simétricos dan a cada participante la misma información y cada uno puede finalizar sus transacciones independientemente, cuando reciben la confirmación de todos los componentes.

El problema en estos casos es que el sistema integrado (múltiples bases de datos) necesita entender los protocolos de cada componente (para los componentes que funcionan con protocolos asimétricos él es el coordinador de las transacciones y para los componentes simétricos debe traducir los protocolos para redireccionarlos a los componentes). Dado que el sistema integrado conoce la información de los estados de los componentes, puede participar para los componentes simétricos como un componentes más, y retardar su confirmación hasta que ha recibido la confirmación de todos los componentes que utilizan protocolos asimétricos, asi mismo, como coordinador de los componentes asimétricos, no enviará la orden de compromiso hasta que ha recibido la confirmación de todos los componentes simétricos. De esta forma, el nuevo agente formado por la base de datos múltiple determina el compromiso de ambos grupos.

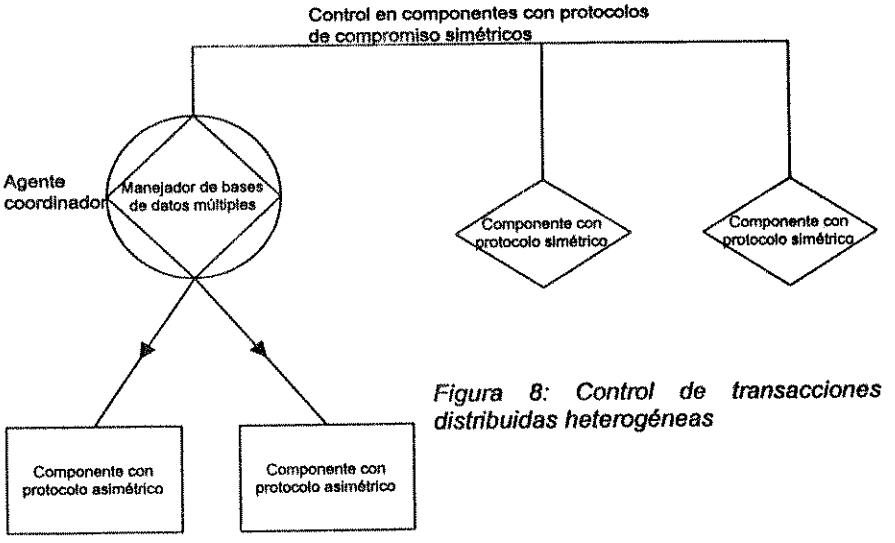


Figura 8: Control de transacciones distribuidas heterogéneas

El manejador de transacciones de bases de datos múltiples debe utilizar su propia bitácora independiente de las bitácoras de los componentes.

2.8 Requerimientos mínimos de un sistema integrado

Todo sistema integrado de múltiples bases de datos debe asegurar la satisfacción de los siguientes requerimientos:

1. **Acceso a los datos independiente de la forma de almacenamiento o de la ubicación:** como se describió anteriormente, debe asegurarse la interoperabilidad en un sistema

distribuido, sobre la base de una transparencia funcional respecto a cada sistema (relacional y no relacional) y transparencia respecto a la localidad de los componentes y la forma de almacenamiento. Esto significa que una consulta hecha desde un punto del sistema, que involucre datos de todas las localidades, debería poder realizarse con la misma facilidad que una consulta local. Debe proveerse a los usuarios un vista lógica de los datos y un sistema de acceso uniformemente funcional.

2. **Integración de esquemas y dominios:** los usuarios de un sistema integrado deben poseer un esquema que les facilite determinar qué información está disponible en todos los componentes del sistema. Este ambiente se llama frecuentemente la base de datos federada. Debe resolverse toda inconsistencia semántica (variantes en nombres, tipos y unidades de medida).
3. **Consistencia de la información:** la información debe ser confiable para cualquier usuario en cualquier punto del sistema.
4. **Seguridad:** el acceso a los datos distribuidos no debe comprometer la seguridad de los datos en las bases de datos destino. El modelo de seguridad debe proveer acceso autorizado a un esquema integrado sin violar la seguridad de las fuentes de datos autónomas que se están integrando.
5. **Rendimiento:** debe minimizarse el costo adicional que se genera con integrar datos de múltiples fuentes, reduciendo el volumen de datos que se transfiere a través de la red y maximizando la cantidad de información que se procesa en una unidad de tiempo.
6. **Capacidad de integración con otros sistemas (apego a otros sistemas abiertos):** cualquier sistema integrado debe incluir herramientas y aplicaciones con interfaces de SQL estándar, tanto a nivel de llamada (ODBC) o a nivel de lenguaje (ANSI SQL). Debe proveer y habilitar acceso a los datos a través de los protocolos disponibles en las arquitecturas más comunes de red, como (como TCP/IP y SNA).
7. **Facilidad de administración del sistema múltiple y de sus componentes:** esto incluye facilidad de configuración, mantenimiento y uso.

3 INTEGRACIÓN SEMÁNTICA DE UN SISTEMA DE BASES DE DATOS

3.1 El concepto de mediación

Un mediador es un módulo de software que se vale de conocimientos codificados acerca de ciertos conjuntos o subconjuntos de datos para crear información para una capa más alta de aplicaciones. Dicho de otro modo, un mediador es visto como una capa de abstracción intermediaria entre la base de datos y las aplicaciones que la utilizan. Esta capa es necesaria principalmente para poder interpretar la semántica individual de los diferentes dominios en una aplicación que los integra.

La necesidad de crear el concepto de mediación se deriva de que es prácticamente imposible uniformar todos los sistemas de bases de datos bajo un modelo semántico único (por razones de mercadeo y por que diferentes dominios necesitan diferentes modelos) y que lo más usual es que todos los dominios cambiarán con el tiempo.

Con el objetivo de asegurar la apertura de los sistemas, se utiliza una arquitectura basada en capas[5]. Los mediadores son módulos que aplican una capa activa y explícita entre las aplicaciones del usuario y las fuentes de datos. Para comprender el uso de una arquitectura de tres capas, visualicemos el problema como un sistema cliente/servidor, donde el cliente son las aplicaciones de los usuarios en sus estaciones de trabajo y el servidor son las diferentes bases de datos a las que el usuario desea tener acceso (comprendiendo también que en este caso, el servidor es en sí un sistema de servidores, formado por uno o más DBMS). En este caso, se definen claramente las dos capas de los extremos, la capa de usuario (cliente) y la capa base (servidor), y la capa del mediador puede entenderse como la forma o el medio como el cliente puede llegar al servidor.

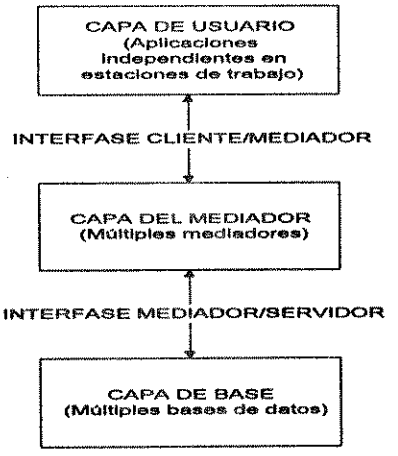


Figura 9: Arquitectura de capas de un sistema de múltiples bases de datos.

En palabras sencillas, un mediador simplifica, abstrae, reduce, mezcla y explica los datos disponibles en los servidores de bases de datos, compensando las diferencias a nivel del

[5] Las arquitecturas basadas en capas son la base para la construcción de redes de ordenadores, por las ventajas que presenta respecto a modularidad, acoplamiento y cohesión. Para ampliar esta información puede consultarse el libro Redes de Ordenadores de Andrew Tanembaum.

sistema manejador de bases de datos, estructura de la base de datos y la representación y el significado de los valores de los datos, para presentar al cliente una interfaz única.

Los mediadores deben tener dos interfaces:

1. **Interfaz cliente/mediador:** se implementa a través de funciones o rutinas que traducen las solicitudes del usuario. Usualmente el usuario final no sabe que existe el mediador (es decir que es transparente a la aplicación), por lo que no es necesario que esta interfaz sea amigable al usuario final, sino flexible, abierta, robusta y fácil de utilizar por el programador de aplicaciones.
2. **Interfaz mediador/servidor(es):** los estándares para el acceso a bases de datos como el SQL y el RDA (ver adelante) proveen la base de comunicación a los mediadores. La interfaz que se utilice dependerá de la generalidad del mediador que se desee (mediador cliente/base de datos única o mediador cliente/múltiples bases de datos), pues un mediador sencillo puede ser muy particular respecto a los protocolos de acceso, y en uno para múltiples bases de datos será provechoso basarse en estándares, pues además, necesitará administrar operaciones de consulta y modificación de los datos entre varios sistemas.

La ventaja de la arquitectura basada en capas es que la implementación del mediador podría basarse en un módulo único (que podríamos llamar un módulo gordo, pues él contiene las interfaces con todos los manejadores involucrados y presenta una única interfaz al cliente). Otra forma de implementación puede ser que el cliente se comunique con más de un mediador con una sola interfaz a la capa base (uno por cada DBMS involucrado), teniendo que tomar el cliente la tarea de coordinar la comunicación, o por medio de un sistema de mediadores basado en un mediador inteligente que funcione como coordinador de múltiples mediadores independientes básicos (uno hacia cada DBMS).

En un entorno de integración de múltiples bases de datos, las funciones del mediador pueden comprenderse mejor con el siguiente ejemplo. Se tiene un programa de aplicación de usuario final que utiliza información almacenada en tres bases de datos, DBX, DBY y DBZ. Esta aplicación necesitará hacer consultas (queries) que involucren las tres fuentes de datos.

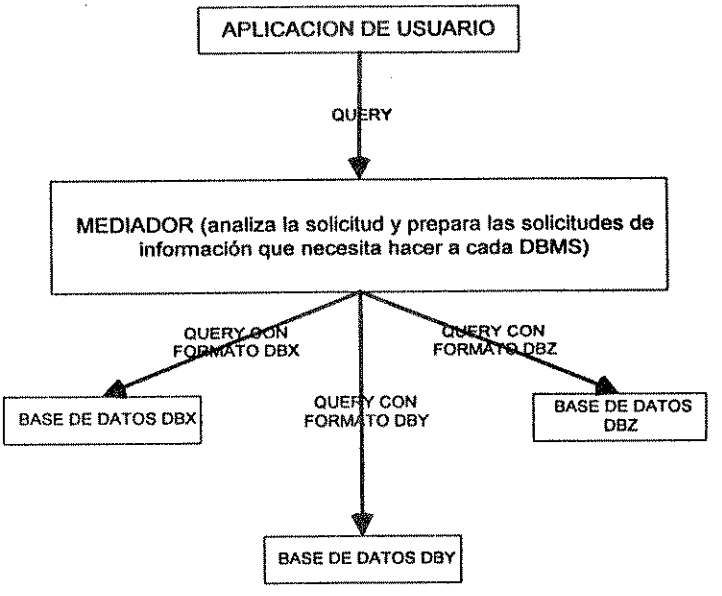


Figura 10: Consulta a múltiples bases de datos a través de un mediador

En la primera fase, el cliente genera la consulta, el mediador la recibe a través de la interfaz cliente/mediador y analiza la solicitud, dividiendo la consulta original en tres consultas que prepara para enviar a cada DBMS (y en este punto cabe mencionar que el

mediador debe conocer la ubicación específica de cada componente, y podría requerir utilizar protocolos de comunicación diferentes para comunicarse con cada uno, pues pueden estar distribuidas en diferentes puntos de la red).

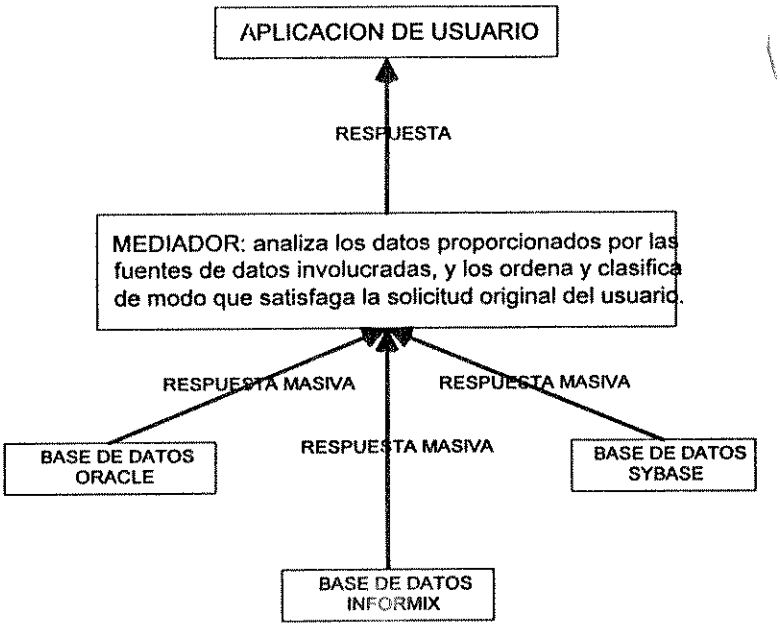


Figura 11: Respuesta a la consulta

Cada componente procesará la solicitud y generará una respuesta, que el mediador recibirá. Estas respuestas independientes deben ser analizadas y organizadas de modo que satisfagan la consulta original del usuario, realizando operaciones sobre los datos y generando hacia el usuario una vista única que satisface su solicitud.

Durante este proceso, el mediador debe asegurar la consistencia de la información creando un ambiente que maneje seguridad e integridad a un nivel superior que involucre los tres componentes de datos. La función del mediador se hace más compleja cuando hay diferencia entre los diferentes miembros del sistema, es decir cuando encontramos componentes heterogéneos.

3.2 Reconocimiento de la heterogeneidad

Un sistema heterogéneo es aquel en el cual al menos uno de los componentes de es distinto a los demás. Extendiendo esta definición, puede existir varios tipos de heterogeneidad entre los componentes de un sistema de bases de datos múltiple (en el cual se intercambia información entre dos o más sistemas autónomos de base de datos). Estos pueden ser:

1. **Heterogeneidad en el modelo conceptual de la base de datos:** diferencias en el modelo conceptual de la base de datos (técnica de especificación de la metadata y lenguaje). Esto significa que puede existir un componente relacional, un componente jerárquico y otro orientado a objetos.
2. **Heterogeneidad ideológica o terminológica:** variantes en la abstracción específica de la aplicación. Por ejemplo, un componente puede ser una aplicación bancaria (cuentas de depósitos monetarios) y otro componente puede ser una aplicación de clientes de una compañía telefónica. En cierto momento, desea integrarse ambos sistemas para lograr que se haga débitos automáticos a las cuentas telefónicas de los cuentahabientes del banco. Cada una es independiente de la otra, pero se necesita una aplicación que integre componentes de ambos sistemas como uno solo.

3. **Esquema conceptual:** en el modelo interno de un sistema de bases de datos (de tres niveles: interno, conceptual y externo), el nivel conceptual se refiere a la perspectiva del usuario respecto a los datos que son almacenados. La heterogeneidad en el esquema conceptual se refiere a diferentes perspectivas de cada componente respecto a definición de los datos (respecto a longitud y tipo) y las relaciones entre ellos, es decir, la especificación de la metadata.
4. **Contenido de la base de datos:** las bases de datos que se desean integrar pueden ser similares terminológicamente, y respecto a su esquema conceptual, sin embargo, pueden almacenar información referente a universos distintos. Este sería el caso, por ejemplo, si queremos integrar los Módulos de Contabilidad de dos (o más) empresas.
5. **Formato de los datos:** Se refiere a la forma en que se representan y almacenan físicamente los elementos atómicos de información. Por ejemplo, una fecha puede representarse en un sistema como un entero largo (4 bytes), en otro representarse como una cadena de caracteres, y en otro como una estructura tipo registro (un campo para día, otro para mes y otro para año). Se refiere al nivel interno de la base de datos.
6. **Heterogeneidad en la herramienta de manipulación (DBMS):** Este tipo de heterogeneidad se da cuando las diferencias están en la herramienta utilizada para el almacenamiento y manipulación los datos. Por ejemplo, podemos tener dos componentes relacionales, que utilizan diferente manejador de base de datos (SQL Server en un sitio e Informix en Otro).

Otros tipos de Heterogeneidad	Ejemplo	
	Dato base	Abstracción
Granularidad	Detalle de ventas	Totales por producto
Generalización	Datos de productos	Tipo de producto
Temporal	Ventas diarias	Ventas mensuales por temporada
Relativa	Costo de productos	Tendencias ajustadas por inflación
Reconocimiento de excepciones	Detalle contable	Evidencia de fraude
Cálculos relacionados con rutas o caminos	Calendarización de aerolíneas	Duración y costo de viajes

Figura 10 : Clasificación de la heterogeneidad semántica

Esto involucra varios problemas que deben considerarse: la definición de un modelo de integración que deberá ser lo suficientemente abierto para caputar las relaciones conceptuales entre las unidades de información y los objetos en la base de datos, la integración de esquemas, metodología de mapeo (para traducir el esquema unificado a localidades distintas en los diferentes componentes del sistema) y funciones de administración de los datos.

3.3 Heterogeneidad semántica

La heterogeneidad semántica puede definirse como variaciones entre los componentes de sistemas de bases de datos en la estructura, organización y descripción conceptual de

unidades de información, unidades de comportamiento (procedimientos), y restricciones de integridad semántica (heterogeneidad de los tipos 2, 3, 4 y/o 5).

Un aspecto clave en la identificación y resolución de la heterogeneidad semántica entre un conjunto de componentes a integrar requiere la integración del contenido de éstos (o porciones del mismo), lo cual involucra tanto estructuras, unidades de comportamiento y restricciones de integridad semántica, como datos que representan hechos de la empresa. Los problemas usualmente se derivan de las diferentes representaciones simbólicas de los sistemas componentes de base de datos. Un base de datos se define por esquema y datos, por tanto, a alto nivel la heterogeneidad se clasifica en conflictos de esquema y conflictos de datos.

Los conflictos de esquema se producen por el uso de diferentes estructuras (tablas y atributos) para la misma información, por el uso de diferentes especificaciones para la misma estructura (incluyendo nombres, tipos de datos y restricciones para los datos) y pueden resumirse en conflictos tabla/tabla, atributo/atributo y tabla/atributo.

Los conflictos de datos pueden originarse de dos formas: datos incorrectos (entradas válidas en un sistema que ocasionan problemas de integridad en el otro, o información redundante entre los dos sistemas), y diferentes representaciones para los mismos datos, respecto a expresión, unidades y precisión.[6]

En general, antes de iniciar cualquier análisis de heterogeneidad de sistemas debe tomar en cuenta lo siguiente:

1. El significado semántico de la información se encuentra vinculado con el modelo de la base de datos, el esquema conceptual, programas de aplicación y la interpretación del usuario, es decir, es relativo a cada sistema. Esta semántica debe hacerse lo suficientemente explícita de modo que pueda se utilizada pera resolver la heterogeneidad en forma semiautomática.
2. La heterogeneidad es relativa a la aplicación, las diferencias y conflictos entre los componentes dependen de los elementos existentes y del nivel de integración deseado.
3. El análisis semántico de las aplicaciones incluye un análisis de las aplicaciones existentes. Muchos elementos pueden entenderse claramente hasta que se revisa el código de los programas (en caso que no exista documentación adecuada).
4. Para resolver la heterogeneidad es indispensable definir un lenguaje común para la identificación de los componentes. Definir una técnica específica proveerá homogeneidad del modelo conceptual como base para el compartirmiento e intercambio de información.
5. Con el objetivo de reutilizar el conocimiento obtenido del proceso de resolución de heterogeneidad semántica, puede definirse un diccionario semántico, orientado a las aplicaciones.
6. Es importante identificar la heterogeneidad semántica, tanto estructural como de comportamiento del sistema, que puede existir a nivel de esquema conceptual y del

---

[6] Kim, **op. cit.** pp. 12-17.

contenido de la base de datos. La resolución de la heterogeneidad encontrada es crucial para producir sistemas lo suficientemente abiertos a nuevas aplicaciones y modificaciones.

Ejemplos de heterogeneidad semántica	Ejemplo		
	Dominio 1	versus	Dominio 2
Diferencia de llave	Alan Turing. El enigma *(Referencia para un lector)*		QA29.T8H63 *(Referencia para un bibliotecario)*
Diferencia de ambiente	Empleados pagados *(Incluyendo retirados)*		Empleados disponibles *(Incluyendo consultores)*
Granularidad de abstracción	Ingreso personal *(de un empleado)*		Ingreso familiar *(Para análisis de pagos de préstamos)*
Base temporal	Presupuesto mensual *(de oficinas centrales)*		Producción mensual *(registros de fábrica)*
Semántica del dominio	Código postal *(uno puede cubrir muchos lugares)*		Nombres de pueblos *(puede incluír muchos códigos postales)*
Semántica del valor	Pago en exceso *(por servicio interno)*		Pago en exceso *(por junta directiva)*

Figura 11: Ejemplos de heterogeneidad semántica

3.4 Manejo de la heterogeneidad de esquema

La resolución de los conflictos de esquema puede llevarse a cabo de varias formas:

1. **Agrupación de todos los componentes en una sola base de datos global.** Esto implica la anulación de los sistemas individuales como se conocen, todos los usuarios y las aplicaciones existentes en cada componente modifican su funcionalidad para utilizar una base de datos global, posiblemente mejorada y ampliada.
2. **Definición de una base de datos 'virtual'**, consistente de partes de cada componente integrado en un sistema global. Los usuarios y las aplicaciones siguen accesando sus bases de datos locales y utilizan la base de datos virtual global para acceso de información externa.
3. **Utilización de un esquema federado.** Se agrupan sistemas individuales formando una entidad en la cual cada uno se integra a una especificación que describe las relaciones de un componente con los demás. Para compartir e intercambiar información puede utilizarse un mediador centralizado para, o bien pueden crearse conexiones directas entre los componentes para este propósito.
4. **Conexiones punto a punto.** Los componentes pueden interactuar por parejas, intercambiando porciones de su información local con otros componentes.

Las ventajas y desventajas pueden resumirse en la figura 12:

Técnica	Autonomía de los componentes	Nivel de organización del sistema global	Cambios en las aplicaciones existentes antes de la integración
Agrupación total	menor ↑	mayor ↑	mayor ↑
Base de datos virtual	↕	↕	↕
Federación de esquemas	↕	↕	↕
Conexiones por parejas	mayor ↓	menor ↓	menor ↓

Figura 12: Tipos de integración de esquemas

Otro aspecto importante que debe ser considerado es el hecho de que cada componente a integrar es un sistema dinámico por sí mismo, y el proceso de integración por si mismo puede alterar los puntos que se deberán compartir. En esto radica la importancia de las restricciones de integridad semántica y la definición del comportamiento de las aplicaciones de las unidades específicas en el proceso de integración. Es necesario definir técnicas de integración que consideren tanto los requerimientos de cada componente específico como los del sistema global.

Una forma de resolver la heterogeneidad a nivel de esquemass es el modelo de sumario de esquemas[7], que se ha desarrollado como una extensión de los sistemas de múltiples bases de datos para proveer un soporte a nivel de lenguaje que identifique automáticamente entidades semánticamente similares con distintos términos de acceso. Un esquema de base de datos es un grupo de términos de acceso (nombre de base de datos, nombres de relaciones, nombres de tablas y atributos), que describen la estructura y contenido de los datos disponibles en la base de datos. Un sumario de esquemas es una descripción concisa y abstracta de los contenidos semánticos de un grupo de esquemas de entrada.

El modelo de sumario de esquemas utiliza relaciones lingüísticas específicas entre los términos del esquema para construir una estructura de datos global jerárquica que describe la información disponible en todas las bases de datos locales (que forman parte del sistema) en una forma cada vez más abstracta. La estructura de datos relaciona términos de acceso local que son semánticamente similares, proporcionando una forma de acceso inteligente y amigable para los sistemas de múltiples bases de datos.

---

[7] Bright, M.W. et. al. "Automated Resolution of Semantic Heterogeneity in Multidatabases". **Sigmod Record.** Vol. 19, No. 3. Estados Unidos. pp. 53-59.

3.5 Uso de valores semánticos para resolver la heterogeneidad a nivel de datos

La información que se intercambia entre sistemas al momento de integrarlos puede tener diferente significado para cada componente según el *contexto* del sistema. Para facilitar la interoperabilidad semántica de sistemas, la información del contexto debe ser un componente activo de los sistemas. El contexto de una pieza de información de define como la metadata relacionada con su significado, propiedades y organización. Un valor se intercambia de un sistema al otro por medio de un proceso de conversión de su contexto original a su contexto destino.

Cuando la información de contexto está explícitamente definida, un sistema de bases de datos puede determinar automáticamente la validez del intercambio de datos y del proceso de conversión, reduciendo errores y simplificando las aplicaciones. Además, se documentan mejor las diferencias entre los ambientes de cada sistema, haciendo posible entender y predecir el impacto de cambios semánticos en los datos.

Un valor simple es una instancia de un tipo y su semántica está definida únicamente por su tipo. Por ejemplo, si 9 es una instancia de tipo quetzales, significa 9 quetzales y no puede compararse con instancias de metros o de dólares. Sin embargo, la interpretación semántica se complica cuando una misma aplicación tiene diferentes dimensiones para los tipos. Por ejemplo, una aplicación que maneje valores numéricos podría tener diferentes escalas para la interpretación de los valores (interpretados en unidades, centenares o miles), o bien, un valor porcentual (una tasa de interés) almacenada como 24.0, pero puede aplicarse. Ambos valores son correctos, y depende de la aplicación la forma en que ésta se aplica.

Formalmente, se define un valor semántico como un valor simple con su contexto, donde el contexto es un conjunto de propiedades con valores semánticos (en una definición recurrente).

En el ejemplo de la tasa de interés en una aplicación bancaria, podría definirse así: *24.0 (Tipo: Nominal, **Periodicidad**: Anual)*, donde 24 tiene dos propiedades: Tipo (que puede se Nominal o efectiva, si es que universalmente se reconoce únicamente estos dos tipos de tasa de interés) y Periodicidad (que podría ser anual, mensual, diaria). Denotada de este tipo, el valor de tasa de interés 24.0 puede ser interpretado correctamente por cualquiera que tenga acceso a este dato.

Una consecuencia de esta forma de especificar los datos es que dos valores semánticos que son sintácticamente distintos pueden tener el mismo significado, como 4 (Unidad_longitud='metros') y 400 (Unidad_longitud = 'centímetros'), y pueden ser intercambiables porque existe un proceso de conversión entre ellos.

Una función de conversión para la propiedad P es un procedimiento que convierte un valor simple de un valor de P a otro, por ejemplo, cvt_Unidad_longitud (4, 'metros', 'centímetros') convierte el valor 4 de metros a centímetros. Las funciones de conversión pueden ser implementadas en cualquier lenguaje de programación, y usualmente requieren consultas a tablas (como las que convierten unidades de medida y unidades monetarias),

fuentes de datos de tiempo real o reglas lógicas. Es posible que una propiedad tenga más de una función de conversión definida, y esto dependerá únicamente de la forma en que la aplicación utilice los datos.

Las funciones de conversión pueden ser definidas por el sistema o por cualquiera de las bases de datos o aplicaciones que conforman el sistema. Luego son almacenadas en librerías de conversión, disponibles para todo el sistema. Las funciones de conversión pueden tener características que son útiles para la comparación semántica. Hay tres caracterizaciones de las funciones de conversión:

1. **Total:** según esté definida para todos los argumentos. Una función total puede ser una que convierta de una unidad de medida de longitud a otra.
2. **Independiente del número de pasos:** formalmente una función es independiente del número de pasos si: cvtP(a, p1, p1) = a y cvtP(a, p1, p3) = cvtP(cvtP(a, p1, p2), p2, p3), o dicho de otra manera, es equivalente *cvt_ulongitud*(24, 'centímetros', 'kilómetros') = *cvt_ulongitud* (*cvt_unidad_longitud* (24,'centímetros','metros'), 'metros', 'kilómetros').
3. **Preservadora de orden:** si los valores simples asociados a dos valores semánticos no cambian si orden si son convertidos a otro contexto. Por ejemplo, la unidad 3 pulgadas siempre es menor que 4 pulgadas, no importando el contexto en el que se traduzca, como *cvt_ulongitud* (0.1, 'metros', 'centímetros') < *cvt_ulongitud*(15,'metros','centímetros') . Una función no preservadora de orden puede ser una que convierta Tipo de Código: por ejemplo, 48 (Tipo_Código = 'ASCII') y 240 (Tipo_Código = 'EBCDIC') corresponden ambos al caracter '0', sin embargo la función de conversión *cvt_Tipo_Código* (48, 'ascii', 'ebcdic') > *cvt_Tipo_Código* (65,'ascii','ebcdic'), aún cuando 48 < 65.

Toda función de conversión tiene asociado un costo de conversión (que estima el costo de ejecución del proceso), es decir, una función que consulte fuentes de datos en tiempo real es más costosa que una que simplemente involucre operaciones aritméticas. Considerar el costo de conversión es importante para el análisis de ejecución de consultas (queries).

La comparación de valores semánticos depende del contexto en el cual se desea hacer la comparación (llamado contexto objetivo). El valor de la comparación se define convirtiendo ambos valores semánticos al contexto objetivo y comparando los valores simples asociados con los resultados, es decir, se hace una comparación relativa, porque el valor real depende del contexto objetivo, el cual puede ser, inclusive, distinto de los contextos de los valores comparados, con la única restricción que las propiedades del contexto objetivo estén contenidas en la intersección de los conjuntos de propiedades de los valores comparados, además, es posible realizar operaciones aritméticas (como la suma y la resta) sobre valores semánticos; estas operaciones deben ser evaluadas en un contexto y el resultado será un valor semántico en el contexto objetivo.

3.6 Arquitectura para intercambio de valores semánticos y mediadores de contexto

Los componentes de los sistemas de información convencionales (aplicaciones y DBMS) están diseñados para intercambiar valores simples, que usualmente no es información significativa para un sistema real, como lo son los valores semánticos. Sin embargo, las

aplicaciones y fuentes de datos actuales no están preparadas para enviar o recibir datos como valores semánticos, como no pueden evaluar propiedades, determinar grado de similitud semántica, seleccionar contextos objetivos y resolver conflictos.

Con el fin de facilitar el intercambio de valores semánticos entre componentes de los sistemas de información, se requiere una arquitectura de sistemas específica, que provea una solución general a la interoperabilidad semántica e involucre los componentes de los sistemas existentes y varios modelos de datos.

Sciore, Siegel y Rosenthal[8] propusieron una arquitectura basada en cinco clases de componentes: sistemas de información, ambientes de datos, mediadores de contexto, librerías de conversión y ontologías (terminologías) compartidas.

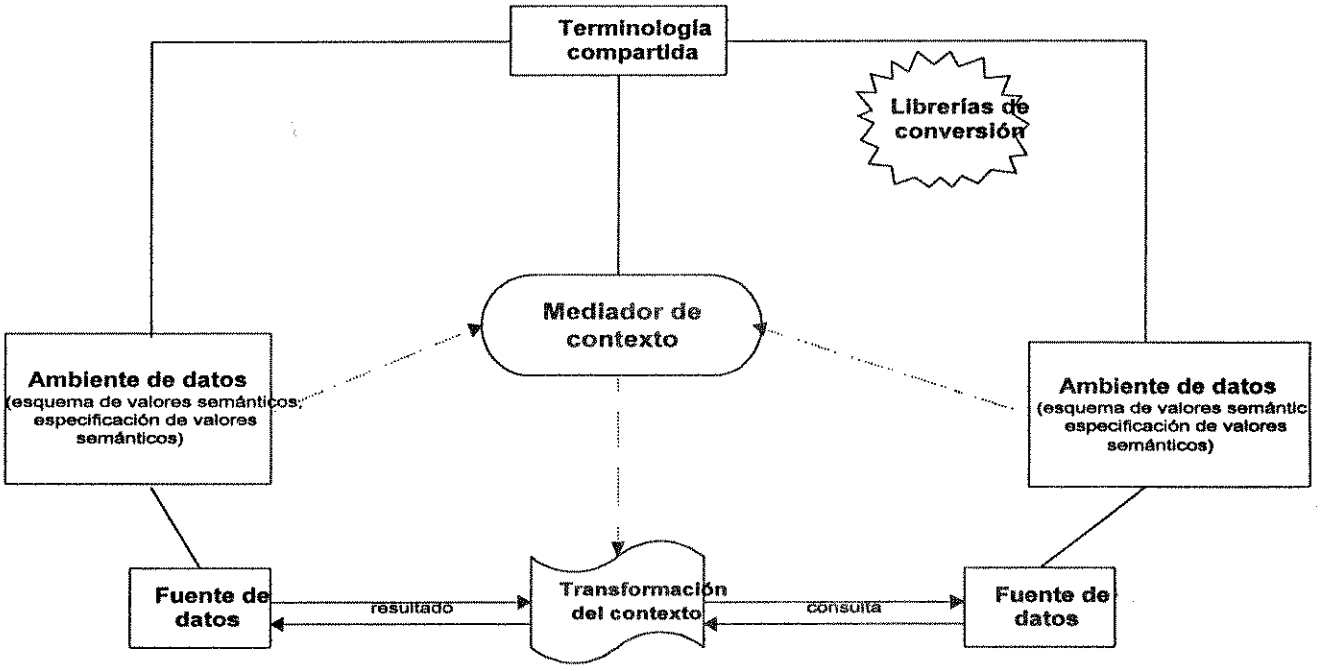


Figura 13: Arquitectura para un sistema con dos sistemas de información y un mediador de contexto

El componente central de esta arquitectura es el mediador de contexto. El mediador de contexto es el agente que dirige el intercambio de valores de un sistema de información hacia el otro, y provee servicios tales como el mapeo de atributos entre los sistemas, evaluación de propiedades y conversión. Todo el intercambio de datos se realiza a través del mediador de contexto. Se diseña un mediador de contexto para funcionar entre componentes

[8] Sciore, Edward, "Using Semantic Values to Facilitate Interoperability among Heterogeneus Information Systems", **ACM Transactions on Database Systems**, Vol. 19, No. 2, 1994, pp. 254-290.

de sistema específicas (como SQL o RPC[9]) y se modifica únicamente cuando la interfaz es modificada. Las ventajas de este enfoque es que es no pone restricciones en los modelos de datos utilizados y limita el número de interfaces requeridas por cada componente de sistema.

Cada componente de sistema puede tener un ambiente de datos asociado, que contiene dos partes: su esquema de valores semánticos, que define los atributos y sus propiedades, y la especificación de valores semánticos, que detalla valores para algunas o todas estas propiedades. El mediador de contexto utiliza los ambientes de datos para determinar si un intercambio de datos solicitado es posible, y si lo es, las conversiones necesarias.

La terminología compartida especifica un mapa de términos. Este mapa describen equivalencias de nombres entre los componentes, de modo que las referencias a atributos, propiedades y sus valores en un sistema de información pueda ser trasladado a los nombres equivalentes en otro.

El componente final es una librería de conversiones, que contiene funciones de conversión. Existe una librería global y varias librerías locales para cada componente de información. Una propiedad puede tener múltiples funciones de conversión en varias librerías y una librería puede tener funciones de conversión para muchas propiedades.

Una arquitectura como ésta es modular y por lo tanto fácilmente extensible.

---

[9] Las llamadas a procedimientos remotas (RPC) es un método de bajo nivel de comunicación entre procesos en la que el procedimiento que se ejecuta puede estar en otra computadora de la red. El modelo de RPC fue definido con los estándares de ambiente de computación distribuida (conocido por sus siglas en inglés DCE (Distributed Computing Environment) de OSF (Open Software Foundation). Una computadora remota actúa como el "servidor de cómputo", compartiendo ciclos del procesador con otras computadoras en la red.

4 CONSIDERACIONES EN EL DISEÑO DE UNA HERRAMIENTA PARA LA INTEGRACIÓN DE BASES DE DATOS HETEROGÉNEAS

4.1 Clasificación de los requerimientos de integración

Antes de iniciar una integración de sistemas, es necesario determinar cuáles son los requerimientos de la organización. Generalmente, estos requerimientos caen en una o más de estas categorías:

1. **Integración transaccional de los datos:** se caracteriza por consultas simples que recuperan o actualizan volúmenes pequeños de información en la base de datos. Requiere acceso en tiempo real a los datos reales (no una copia en datawarehouse).
2. **Integración de queries ad-hoc:** se caracteriza por consultas moderadamente complejas que recuperan información de múltiples fuente de datos, en grandes volúmenes de datos. No necesariamente se requiere información de tiempo real y en línea, aunque son convenientes.
3. **Integración a nivel de análisis de datos:** las aplicaciones de integración analítica de datos generan reportes de información a nivel corporativo con queries que toman información de casi todas las fuentes de datos disponibles en el sistema, por lo que deben manipular volúmenes muy grandes de datos. Usualmente no utilizan consultas en línea (podría implementarse por medio de un datawarehouse).

4.2 Interacción con los sistemas operativos

El sistema operativo provee la base sobre la cual se construyen los DBMS. El sistema operativo es básicamente un administrador de recursos[10], y es parte del ambiente del sistema de bases de datos.

En muchos sistemas administradores de bases de datos, los servicios del sistema operativo se construyen como internamente dentro del manejador, lo que generalmente resulta más eficiente en cuestiones de rendimiento. Los servicios básicos que el manejador necesita del sistema operativo (y que puede utilizar o emular, según la implementación específica) son:

1. **Control del procesador:** los accesos a la base de datos constituyen procesos que se ejecutan en la máquina servidora. El DBMS debe ser capaz de utilizar uno o más procesadores.

[10] Si desea profundizar en este tema, se recomienda consultar el libro Sistemas Operativos, conceptos fundamentales de A. Silberschatz.

2. **Control de la memoria:** el DBMS utiliza los servicios de alojamiento dinámico de memoria principal para administrar las estructuras de datos, de control y almacenamiento temporal utilizados para sus operaciones básicas.
3. **Control de dispositivos de entrada/salida:** el sistema operativo es responsable por el control básico y la programación del uso de los recursos de entrada/salida como dispositivos y canales. Adicionalmente, provee servicios de manejo de caché o buffer.
4. **Seguridad:** el sistema operativo provee al DBMS la estructura básica del control de seguridad, sin embargo, necesita un control de seguridad a un nivel más detallado (permisos por grupos, a nivel de tabla, registro y atributo).
5. **Administración del almacenamiento secundario / manejo de archivos:** el manejo de archivos y la administración del almacenamiento secundario son el corazón del administrador de base de datos. Normalmente, estas funciones son realizadas por el DBMS, lo que le permite una mejor correlación entre el manejo lógico y físico de los datos.

Si bien la herramienta integradora debe ser independiente de la forma cómo los diferentes componentes realizan estas tareas, es importante que conozca las implementaciones específicas de cada una, para optimizar el rendimiento general. Además, si se ve a la herramienta integradora un manejador de alto nivel (que hará las veces de coordinador), ésta necesitará a su vez, utilizar estos servicios. La relación de dependencia de la herramienta de integración respecto al sistema operativo es inversamente proporcional a la portabilidad de la misma y este puede ser un buen parámetro para considerar la implementación interna de algunos servicios.

4.3 Arquitectura de red y comunicaciones

Un aspecto implícito en la interconexión de bases de datos heterogéneas es que cada componente del sistema de bases de datos múltiples reside en una computadora que forma parte de una red. La comunicación de una red de computadoras es un aspecto ampliamente estudiado en la actualidad.

Aunque la estructura de la red no es tan importante, si es necesario mencionar que debe poseerse una Arquitectura de red diseñada en jerarquías. Esto significa que debe estar definido un conjunto de capas y protocolos que permitan a los componentes de la red comunicarse.

Por otro lado, si un DBMS va a formar parte de un sistema de múltiples bases de datos (heterogéneos o no), debe poseer herramientas de comunicación de datos, puesto que debe sincronizarse con los protocolos de comunicación de la red.

El modelo de referencia para la interconexión de sistemas abiertos[11] y el modelo de integración de bases de datos deben complementarse como parte de un modelo más completo

---

[11] Mejor conocido como el modelo OSI (Open Systems Interconnection) de la ISO (International Standards Organization), diseñado básicamente para la interconexión de redes de computadoras.

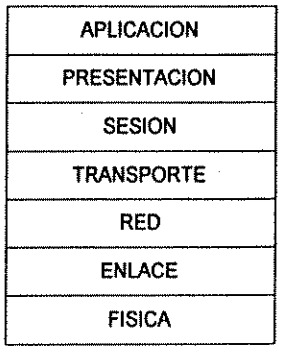


Figura 14: Arquitectura de Capas del modelo OSI

para sistemas de información basados en computadoras. Las bases de datos distribuidas requieren un marco de referencia para la interconexión como el provisto por modelo OSI.

La capa de aplicación del modelo OSI provee servicios de información a las aplicaciones, como el acceso remoto a archivos y las funciones básicas de almacenamiento y recuperación de datos. También incluye funciones de integridad y seguridad, que permiten asegurar la consistencia de la base de datos (manejo de transacciones, control de bloqueos y recuperación).

La capa de presentación organiza información en una forma reconocible por las aplicaciones. Es la más importante para el manejo de datos heterogéneos. Esta capa administra la entrada, intercambio, despliegue y control de los datos estructurados. Incluye funciones de transformación de datos que soportan almacenamiento y recuperación. Un ejemplo es el protocolo de archivos virtuales, otras funciones típicas son el formato de información y selección de sintaxis.

La utilización de las capas de sesión e inferiores es indirecta (lo hace a través de las capas de aplicación y presentación).

Si los datos están distribuidos en múltiples localidades, puede haber muchos grados de dispersión de las funciones del DBMS para las múltiples localidades. Por ejemplo, el DBMS completo puede estar distribuido, en múltiples localidades, con protocolos hasta la capa de aplicación para la interconexión de sistemas. En otro caso, solamente las funciones de la capa de presentación (e inferiores) pueden estar distribuidas o bien, pueden estar distribuidas solamente las funciones de manejo de archivos en la capa de sesión, limitando los protocolos entre sistemas a las capas inferiores.

En un sistema de bases de datos heterogéneo distribuido (que es el caso más común), debe minimizarse las interacciones con el sistema operativo local, que serán únicamente a través de las capas de sesión y presentación, con el fin de establecer circuitos de comunicación entre los nodos (procedimientos de login) y traducción de formatos de datos.

4.4 Requerimientos funcionales de un sistema de bases de datos heterogéneas

En un sistema de bases de datos parcial o totalmente replicado, el integrador de sistemas deberá incluir soporte para distribuir la metadata para soportar el procesamiento distribuido de consultas, actualización y recuperación de nodos, específicamente, respecto a:

1. Información de esquemas
   - Distribución y replicación de datos en los nodos físicos
   - Información global de esquemas
   - Información local de esquemas
   - Nivel de consistencia de los elementos de datos requeridos en cada nodo

2. Mapeo del esquema global del sistema a los esquemas locales.
   - Información dinámica de disponibilidad del sistema
   - Estado de los nodos de la red
   - Información de temporización necesaria para detectar y resolver interbloqueos a nivel global.
3. Reglas para resolver conflictos entre los datos de diferentes nodos del sistema.

Puesto que un sistema heterogéneo es una superestructura construida en base a un conjunto de manejadores de bases de datos y sistemas operativos ya existentes, uno de los principales objetivos es no impactar a los usuarios locales de cada uno de los sistemas de bases de datos en cada nodo.

Como consecuencia de esto, puede decirse que para el integrador del sistema heterogéneo, cada usuario local debe aparecer como simplemente otro usuario de la aplicación global, y a su vez, las interfaces de los DBMS locales, deben ver al integrador de sistemas como otro usuario más (en forma local).

4.5 Modelo de referencia para la estandarización del manejador de la base de datos

Para facilitar la tarea de integración de bases de datos heterogéneas, debe avanzarse en la tarea de estandarización de los manejadores, específicamente, de las interfaces de comunicación con el sistema operativo y con las herramientas de aplicación. El modelo planteado por el grupo DAFTG[12] presentó en 1985 es buen marco de referencia para dividir la tarea de estandarización en piezas manejables, y la relación entre ellas.

El modelo de referencia se resume en la figura 15:

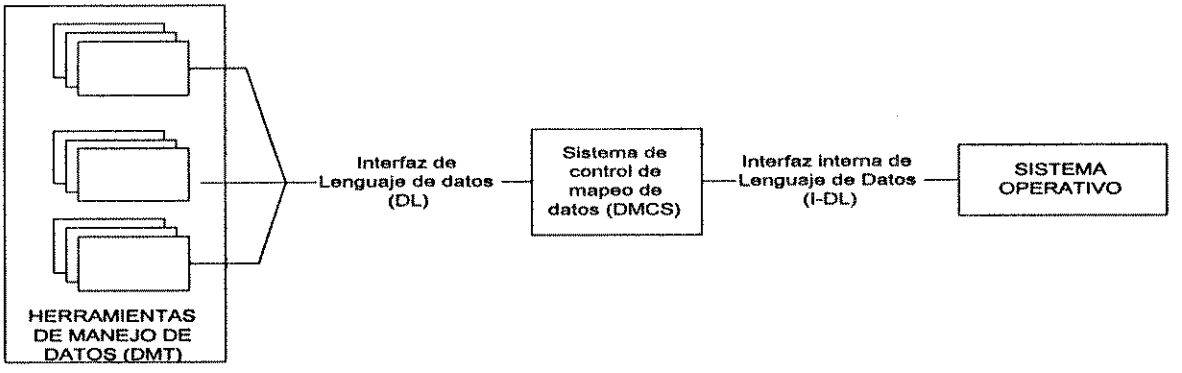


Figura 15: Modelo de referencia para la estandarización de manejadores de bases de datos

El papel del sistema operativo ya fue indicado anteriormente (y recordemos que la interfaz de comunicación y de red es parte de las funciones de éste).

El Sistema de Control de Mapeo de Datos, DMCS (por sus siglas en inglés *Data Mapping Control System*) es un administrador de bases de datos de nivel superior, que provee operaciones tanto para manipulación y descripción de datos. La descripción de los datos se logra aplicando operaciones de manipulación de datos a estructuras de datos que describen otras estructuras de datos, mediante un esquema de modelo de datos que se describe a si mismo. El DMCS fundamenta su labor en una traducción apropiada de las herramientas de manejo de datos, DMT (por sus siglas en inglés *Data Management Tools*), que le permite soportar modelos relacionales, de red, jerárquicos, orientados a objetos, entre otros.

El lenguaje de manipulación de datos del modelo de DMCS es la Interfaz de Lenguaje de Datos, DL, (por sus siglas *Data Language*). Dado que el esquema del modelo de datos se describe a sí mismo, el la interfaz de lenguaje de datos provee todas las definiciones, recuperaciones y manipulaciones.

Las herramientas de manejo de datos son componentes de software que se comunican con el DMCS a través de la interfaz de lenguaje de datos. Estas herramientas proveen interfaces de base de datos orientadas hacia aplicaciones más específicas o funciones que la interfaz de propósito general del interfaz DL, como lenguajes de consulta de alto nivel, sistemas gráficos, reporteadores y herramientas de diseño de datos.

Todos los datos entre el DMCS y el sistema operativo que lo soporta, pasan por la interfaz interna de lenguaje de datos (I-DL).

Este modelo de referencia, se fundamenta en una descripción de datos de cuatro niveles[13], descrita en la figura 16.

El esquema del modelo de datos describe y controla todas las operaciones de la clase de esquemas que pueden definirse por el modelo de datos DMCS, y el mismo se define en términos del modelo de datos, lo que significa que es miembro de la clase de esquemas que describe (se autodescribe). Este esquema es muy importante porque permite una forma de acceso estándar a todos los datos del esquema, que permite que sea un modelo abierto.

El esquema del diccionario de datos contiene toda la información del manejo y el uso del sistema de base de datos, incluyendo el manejo y el uso de esquemas en el sistema de base de datos, conceptos como usuarios, autorización, programas y esquemas, además, todas las

---

[12]DAFTG, Database Architecture Framework Task Group for the ANSI/X3/SPARC Database System Study Group, conformado por miembros de la corporación MITRE, el National Bureau of Standards de Estados Unidos, la Computer Science Corporation, la Universidad de Maryland, la Planning Research Corporation, General Electric Information Services, Co. y la NASA.

[13] La dimensión ortogonal "***intension/extension dimension***" (término en inglés) descrita por el grupo DAFTG.

reglas (estáticas y dinámicas) de quién puede utilizar el diccionario de datos y cómo lo utiliza. El esquema del modelo de datos está descrito en el esquema del diccionario de datos.

Los esquemas de aplicación son parte de los datos del diccionario de datos, y contienen la aplicación específica, que indican cómo una aplicación específica utiliza los esquemas de aplicación, la seguridad respecto a usuarios de las aplicaciones y los permisos que tienen de utilizar los datos a través de esquemas de aplicación específicos.

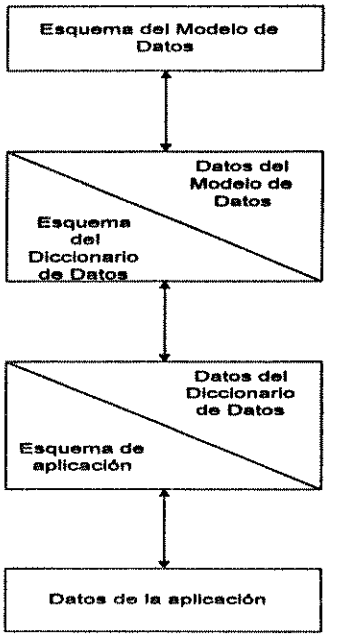


Figura 16: Arquitectura de cuatro niveles

Cada uno de estos niveles contiene en sí mismo una representación de los tres niveles lógicos de visión de una base de datos (físico o interno, conceptual y externo o de visión) y por esto se dice que es una representación ortogonal.

El DMCS realiza tres clases de funciones:

1. Funciones básicas de referencia, eliminación y creación de objetos de datos.
2. Funciones básicas de transformación de datos entre vistas externas conceptuales y vistas internas conceptuales.
3. Funciones compuestas construidas a partir de funciones básicas y otras funciones compuestas, que incluyen procesamiento de lenguaje de datos, control de integridad, seguridad, concurrencia y rendimiento, traducción de solicitudes en lenguaje de datos a lenguaje interno de datos (para satisfacer las necesidades de conversión del nivel externo o de usuario al nivel interno de la base de datos) y acceso físico.

Los procesos básicos para la ejecución de estas operaciones se resumen así (refiérase a la figura 17):

1. Se abre el diccionario de datos, para establecer el esquema del diccionario de datos (que contiene los datos del modelo de datos, y por ende, el esquema del modelo de datos, que se autodescribe).
2. Se abre el esquema de aplicación, que es interpretado por el DMCS por medio del esquema del diccionario de datos.
3. Se solicitan datos (y la solicitud es interpretada utilizando el esquema de la aplicación).
4. El solicitante recibe los datos.

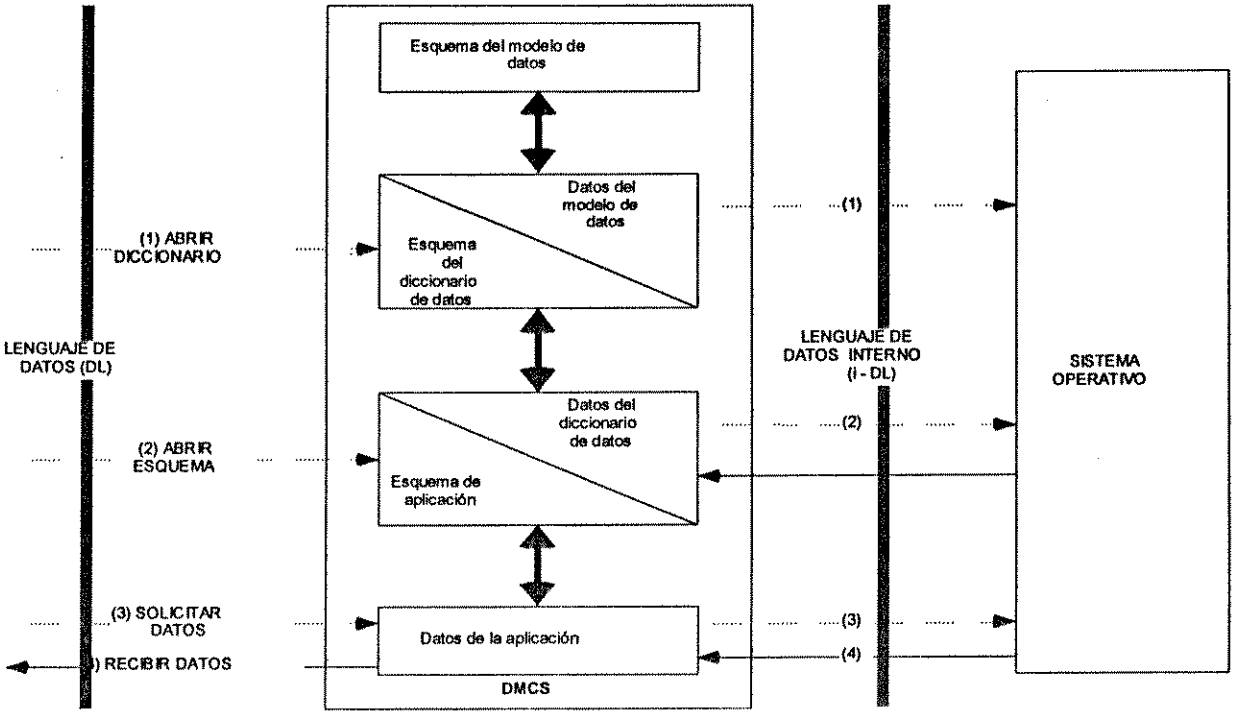


Figura 17: Ejecución de operaciones del DMCS

Los servicios del sistema operativo que son requeridos para soportar el DMCS se dividen en aquellos que soportan directamente el lenguaje de datos interno (entrada y salida de objetos de la base de datos, búsqueda y recuperación de registros basados en valores específicos de atributos, creación de nuevos registros, administración del almacenamiento secundario utilizado por el DMCS, mecanismos de bloqueo para control de concurrencia) y aquellos que proveen el ambiente apropiado para la ejecución del DMCS (administración de memoria, planificación del procesador para el multiprocesamiento de tareas, un mecanismo de soporte de llamadas de servicios a través de las interfaces de lenguaje de datos interna y externa por medio de llamadas a subrutinas o llamadas de servicios ejecutivas, seguridad en el manejo de los datos, carga y ejecución del DMCS, manejo de interrupciones y control de los parámetros apropiados para las mismas, manejo copias de seguridad, robustez para el resistencia a fallas del DMCS o del Sistema Operativo, estadísticas de rendimiento).

Adicionalmente, un sistema de múltiples bases de datos distribuido, heterogéneo o no, requiere un sistema operativo que soporte manejo de comunicaciones (control de enlaces lógicos, detección de problemas de red, ruteo y distribución apropiada de mensajes, detección y corrección de errores y formato de datos) y funciones de control (acceso remoto, comunicación entre procesos, manejo de nombres globales, administración de procesos y sincronización, administración lógica de la configuración de la red, administración de recursos globales y planificación del uso de los mismos).

4.6 Estándares la interoperabilidad de bases de datos

Dado que el objetivo es que los sistemas sean abiertos, es necesaria una forma común de obtener acceso a los datos, es decir, debe estandarizarse el lenguaje de datos, de modo que cualquier herramienta de manejo de datos pueda obtener la información del DMCS. Las dos propuestas más apoyadas para el DL son el ANSI SQL y el OQL.

4.6.1 ANSI SQL

El lenguaje estructurado de consultas (*Structured Query Language* / SQL) ha sido el lenguaje tradicional de consulta de los manejadores de bases de datos relacionales, utilizando tanto como lenguaje de consulta interactiva como lenguaje de programación de base de datos. Casi todos los DBMS incluyen su versión de SQL. Para regular esto, la ISO (*International Standards Organization*) creó sus normas para la estandarización de este lenguaje en las normas ISO/IEC 9075 y ANSI X3.135-1992[14], que han sido generalmente aceptadas.

El ANSI SQL especifica las reglas sintácticas y semánticas del lenguaje SQL para las funciones utilizadas para definir y accesar bases de datos. Estas funciones incluyen:

1. Definición de esquemas, para declarar las estructuras, reglas de integridad y privilegios de acceso a la base de datos.
2. Manipulación de esquemas, para alterar la definición de esquemas.
3. Manipulación de datos, para operar el contenido de (dar altas, bajas y cambios) la base de datos.
4. Administración de transacciones
5. Administración de conexiones
6. Administración de sesiones
7. SQL dinámico para la construcción y ejecución de consultas.
8. Administración de diagnósticos, para comunicar violaciones de integridad y avisos a las aplicaciones.
9. Tablas de información de esquemas, para proveer una descripción de SQL de las definiciones.
10. Enlaces con los lenguajes de programación, para definir procedimientos de bases de datos que pueden ser llamados desde varios lenguajes.

---

[14] Para ampliar esta información, puede consultar las normas:

- ISO/IEC 9075:1992 Information technology - Database languages - SQL,
- ISO/IEC 9075-3:1995 Information technology -- Database languages -- SQL -- Part 3: Call-Level Interface (SQL/CLI),
- ISO/IEC 9075-4:1996 Information technology -- Database languages -- SQL -- Part 4: Persistent Stored Modules (SQL/PSM)

11. SQL Embebido, para definir como sentencias de SQL pueden ser sintácticamente enlazadas en los siguientes lenguajes de programación: Ada, C, COBOL, FORTRAN, MUMPS, Pascal, y PL/I.

El propósito de la estandarización es promover la portabilidad y la interoperabilidad de los programas de aplicación de bases de datos, facilitar el mantenimiento de los sistemas de bases de datos entre ambientes heterogéneos de procesamiento de datos, y permitir el intercambio de programadores entre diferentes proyectos de administración de datos.

4.6.2 OQL

Más recientemente, el grupo de investigación I³/POB[15], propuso que para facilitar la comunicación de sistemas heterogéneos se defina un estándar basado en un lenguaje mínimo de consulta basado en el Lenguaje de Consulta Orientado a Objetos (OQL por sus siglas en inglés *Object Oriented Query Language*), propuesto por el grupo de administración de bases de datos orientadas a objetos (*ODMG committee*). El OQL es un sublenguaje con sintaxis sencilla (muy parecida a la de SQL), que implementa las operaciones básicas del álgebra relacional (seleccionar, insertar, eliminar, mezclar y restringir los datos a nivel de filas y/o de columnas) y tiene la capacidad de describir las fuentes de datos según sus tipos (relaciones o clases, atributos o instancias).

4.6.3 RDA

Otro estándar importante a considerar es el RDA[16] (Acceso remoto a bases de datos), creado para normar la comunicación de datos en ambientes cliente/servidor, y que abarca métodos de establecimiento de conexión, control de diálogo, protocolos de comunicación, estructura de los paquetes y sintaxis para el acceso de datos.

---

[15] El grupo para Integración Inteligente de Información/ Bases de datos Persistentes de Objetos de la Agencia de Proyectos de Investigación Avanzados del departamento de la Defensa de Estados Unidos. (DARPA Intelligent Integration of Information/Persistent Object Databases), en los congresos efectuados en San Diego en enero de 1996 y en la Universidad de Maryland en abril de 1996.

[16] Puede ser consultado ampliamente en los estándares de la ISO:

- ISO/IEC 9579-1:1993 Information technology -- Open Systems Interconnection -- Remote Database Access -- Part 1: Generic Model, Service and Protocol
- ISO/IEC 9579-2:1993 Information technology -- Open Systems Interconnection -- Remote Database Access -- Part 2: SQL specialization
- ISO/IEC 9579-3:1996 Information technology -- Open Systems Interconnection -- Remote Database Access -- Part 3: SQL specialization Protocol Implementation Conformance Statement (PICS) proforma

5 CATEGORÍAS DE PRODUCTOS PARA LA INTERCONEXIÓN DE BASES DE DATOS

5.1 Hardware necesario para la integración

Cuando se desea integrar sistemas de bases de datos heterogéneos, usualmente éstos están en localidades distintas (diferentes departamentos, cada uno en un servidor diferente). Esto implica naturalmente la necesidad de existir una red de comunicaciones entre las computadoras del sistema. Existen varias opciones para esto, dependiendo de las necesidades y los recursos de la organización, desde las redes locales (LAN) con conexiones tipo Ethernet o Token Ring hasta redes de área global (WAN) basadas en fibra óptica, sistema telefónico o microondas[17].

Cualquiera de las tecnologías que se elija para la comunicación, debe considerar lo siguiente:

- Debe definirse un medio de comunicación (cable coaxial, twisted pair, líneas telefónicas, microondas o mixto), y una arquitectura de red.
- Cada computadora que tendrá acceso al sistema debe contar con dispositivo de comunicación con otras computadoras (o dicho de otra forma debe tener un puerto de comunicación, un modem o una tarjeta de red).
- Debe existir un protocolo de comunicación entre los componentes, las opciones mas utilizadas son TCP/IP y NetBios.

5.2 Software para la integración de bases de datos

5.2.1 Categorías de productos para la integración

Hay tres tipos de productos de integración de bases de datos, cada uno con sus propias fortalezas y debilidades.

1. **Integración de datos basada en el cliente:** este tipo de productos son ideales para la integración transaccional de datos. Proveen un rendimiento muy bueno en relación costo/beneficio y, dependiendo de la optimización, son capaces de proveer carga mínima a la red y a la base de datos. Puede proporcionar acceso a datos en tiempo real. Son fáciles de instalar y configurar, para ambientes de desarrollo limitado. No pueden satisfacer las necesidades de Integración a nivel de análisis de datos porque la cantidad de datos que se debe manipular en este caso, (que puede sobrecargar la red y la máquina cliente).

[17] Para ampliar esta información, se recomienda consultar cualquier libro de Telecomunicaciones y Redes.

El tipo de herramienta de integración basado en el cliente más común es el ODBC, que se describirá más adelante.

2. **Integración de datos basada en el servidor:** este tipo de productos satisfacen la integración transaccional y la integración ad hoc. Proveen acceso a datos en tiempo real (con un rendimiento superior). Este tipo de productos son preferidos cuando se desea integración de datos a nivel de empresa, pues su administración es más centralizada. También se utilizan mucho en ambientes Internet/intranet. La carga que agregan a la red y al servidor de base de datos son similares a los productos de basados en el cliente. Difieren únicamente en el rendimiento y en el grado de sofisticación de las optimizaciones que implementan. No satisfacen los requerimientos de integración de análisis. Generalmente se implementa por medio de gateways.

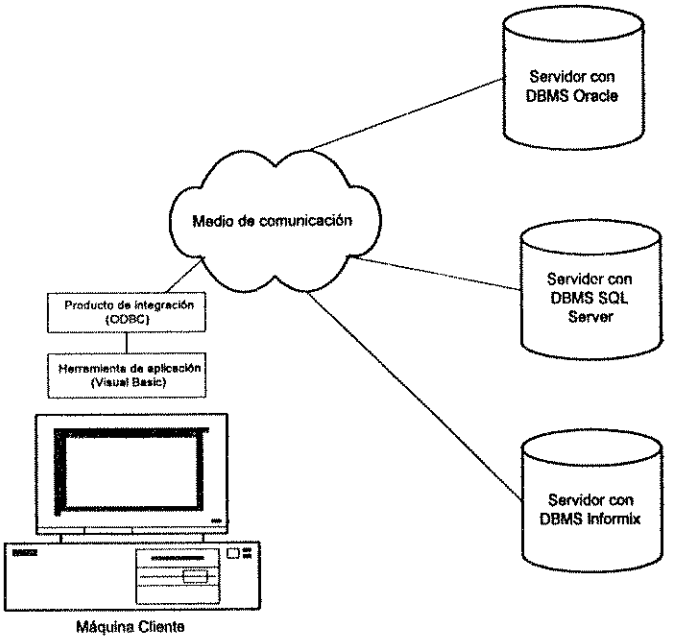


Figura 18: Uso de productos de Integración

3. **Integración de datos vía datawarehousing:** los productos de datawarehousing satisfacen las necesidades de integración a nivel de análisis. Trabajan en base a copias (periódicamente refrescadas) de los datos ubicadas en una sola localidad, de los sistemas transaccionales.

5.2.2 API

El API (de las siglas en inglés de Application Program Interface) es un formato de lenguaje y mensajes que define cómo los programas interactúan con funciones en otros programas, sistemas de comunicación o dispositivos de hardware.

El programador ve al API como un conjunto de rutinas que puede utilizar para construir programas rápidamente para un sistema específico. Un API para múltiples plataformas define sentencias de programación que pueden utilizarse para construir aplicaciones que trabajan sobre diferentes sistemas operativos.

Hay tres tipos de API para comunicaciones entre programas y para comunicaciones entre programas y servidores en ambientes cliente/servidor: API de conversación , Llamadas de procedimientos remotos (RPC: Remote Procedure Calls) y API de mensaje.

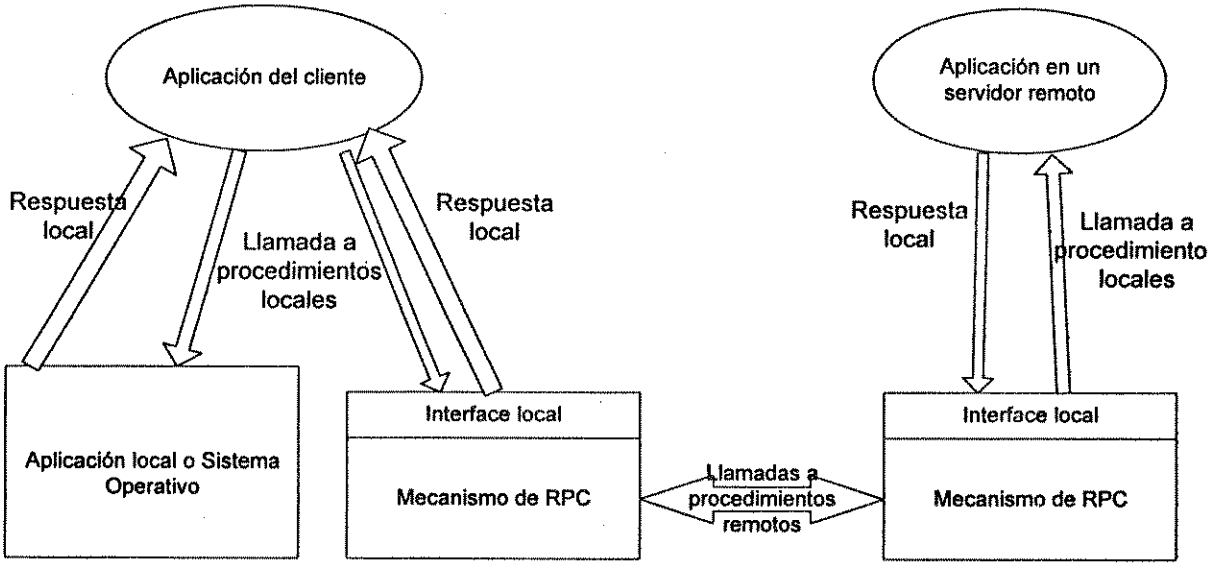


Figura 19: Mecanismo de ejecución de procedimientos basado en RPC

5.2.3 Componentes intermedios (Middleware)

Middleware es una infraestructura de aplicaciones distribuidas, constituido por conjuntos de software distribuido que permite a elementos de las aplicaciones interoperar a través de enlaces de red. Se refiere un conjunto de productos que incluyen software de red (como drivers TCP/IP), software de red específico de las bases de datos y drivers de ODBC. El Middleware tiene su principal utilización en los enlaces entre computadoras, a un nivel mucho más alto que los propios enlaces físicos, a un nivel lógico entre los elementos de las aplicaciones y su objetivo es localizar transparentemente servicios en la red, para interactuar con otra aplicación o servicio, en forma independiente de los servicios de la red, ser confiable, funcional, escalable y estar siempre disponible. Intenta solucionar las posibles incompatibilidades entre los protocolos de comunicación, los lenguajes de consulta de base de datos, la lógica de las aplicaciones y los sistemas operativos para lograr interoperabilidad total.

Provee cinco servicios principales: Independencia de hardware, intercambio de componentes claves de software (como DBMS), independencia de la red, ahorro operativo y ahorro administrativo.

Hay varias categorías de middleware:

1. **Middleware de bases de datos:** utilizado en ambientes específicos de bases de datos. Provee el enlace entre cliente y servidor cuando la aplicación de cliente que utiliza información de la base de datos (en el servidor) ha sido diseñada para utilizar únicamente un tipo de base de datos. En este caso, el middleware traduce los dos esquemas.
2. **Middleware basado en RPC:** es una solución de propósito general para la computación cliente/servidor que el middleware de base de datos. Las llamadas de procedimientos remotos se utilizan para obtener acceso a una amplia variedad de fuentes de datos desde una aplicación única.

3. **Middleware basado en intercambio de mensajes:** este tipo de middleware amplía la filosofía del RPC al solucionar el problema de las fallas en el sistema cliente/servidor. Provee conectividad sincrónica y asincrónica entre el cliente y el servidor, de modo que los mensajes pueden ser enviados inmediatamente o ser almacenados y transmitidos por demanda.
4. **Middleware basado en objetos:** este tipo de middleware trae los beneficios de la tecnología cliente/servidor a la computación distribuida mediante los ORB (Object Request Brokers, término en inglés para el agente receptor y distribuidor de solicitudes). Los ORB empacan y manipulan los objetos distribuidos, que pueden contener información mucho más compleja acerca de una solicitud en un ambiente distribuido que una llamada de RPC o el paso de mensajes y puede ser utilizado específicamente para datos no estructurados o no relacionales. Esta tecnología está basada en el IPC (Inter-Process communication o comunicación entre procesos) pues basa su comunicación no en direcciones estáticas de procedimientos como el RPC, sino en los tipos de datos del objeto y sus propiedades, lo que significa que los patrones de comunicación entre los módulos involucrados pueden ser modificados según sea necesario, aún cuando la aplicación está ejecutándose.
5. **Monitores de procesamiento de transacciones (TP monitors):** los monitores de transacciones son actualmente una tecnología de middleware que puede proveer un API único para escribir aplicaciones distribuidas, pues generalmente incluyen un conjunto de herramientas de administración que agregan controles para ambientes distribuidos abiertos.
6. **Middleware inteligente:** el middleware inteligente es una categoría especial que esconde la complejidad del middleware brindando una capa de protección entre los programas de aplicación y la tecnología de infraestructura que le sirve de base. Esta constituido por un conjunto de herramientas para automatizar el uso de middleware y un conjunto de servicios ejecutables que incrementan la robustez del ambiente operativo, facilitando la tarea del desarrollador de aplicaciones, al separar las mismas de las interfaces del middleware.

5.2.4 Componentes del Middleware

El Middleware logra cumplir sus objetivos proporcionando protocolos y formatos a nivel de aplicación, acceso a los servicios de aplicación, soporte para uno o más modelos de aplicación y facilidades administrativas. Sobre esta definición, todos los productos de middleware tienen los mismos componentes básicos, pues están destinados a cubrir las tres fases del desarrollo de aplicaciones: desarrollo, ejecución y producción (incluso administración), aunque se han desarrollado más en el área de ejecución.

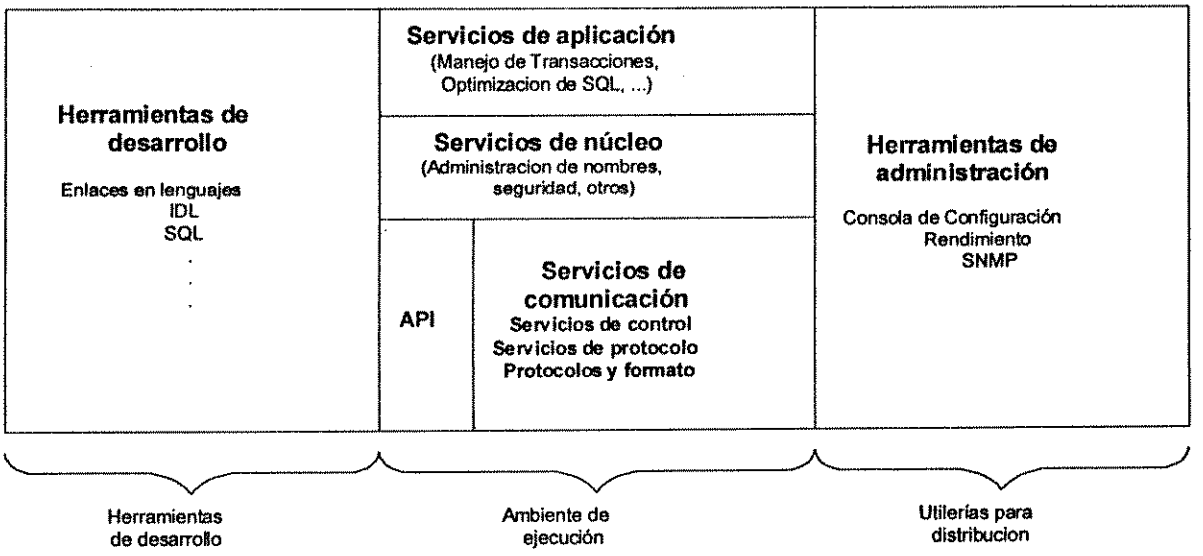


Figura 20: Variantes en las aplicaciones del Middleware

El middleware ha evolucionado desde sus orígenes, donde consistía básicamente en software de comunicaciones, a servicios de aplicación en ambientes de ejecución (como monitores de transacciones, ruteo y optimización de SQL y replicación de bases de datos), y combinaciones de ambos en servicios de integración.

La base del middleware son los servicios de comunicación. Los protocolos de comunicación y formatos que utiliza describen interacciones entre los componentes de aplicación, y no deben confundirse con los protocolos de transporte de redes (como TCP/IP, IPS/SPX). El middleware provee tres servicios básicos de comunicación: formatos y protocolos, servicios de protocolo y servicios de control.

Un formato describe la estructura del mensaje que viajará a través del medio de comunicación (alambre), incluyendo la sintaxis para crear la estructura. El protocolo define la representación del mensaje en la línea. La tendencia en middleware es soportar más de un protocolo y formato en el mismo producto.

Los servicios de protocolo añaden características útiles a las comunicaciones básicas, que pueden incluir ordenamiento de mensajes, traducciones de formato de datos entre plataformas, compresión de mensajes, traducción del protocolo de transporte y encriptamiento de mensajes.

Los servicios de control soportan uno o más modelos de comunicación o estilos de procesamiento distribuido. Cada modelo difiere en la forma en que estructura las comunicaciones entre los elementos de las aplicaciones. Los servicios de control proveen los

protocolos de señalización, colas de mensajes, acoplamiento de mensajes y otros servicios requeridos para soportare cada modelo de comunicación.

Existen diez modelos de comunicación soportados por los servicios de control del middleware, según la tabla:

Modelo de comunicación	Ejemplo dependiente de la red	Ejemplo independiente de la red
Datagrama Mensaje unilateral restringido	Notificación de estado de dispositivo de red	Sistema de paginación
Una pasada Mensaje unilateral restringido, respuesta única	Servidor de cálculo basado en la red.	Transferencia de archivos
Consulta (Query) Mensaje unilateral restringido, respuestas encadenadas	Consulta de base de datos basada en red local	Búsqueda en Web
Asimétrico Múltiples mensajes y respuestas; única sesión.	Centro de servicio al cliente (vía telefónica	Aplicación de publicaciones basadas en Web.
Simétrico Múltiples mensajes y respuestas; en dos canales dedicados.	Sistema de reservaciones con cargas altas.	Aplicación de publicaciones de Web, con cargas altas.

Figura 21: Modelos de comunicación del Middleware

Una característica clave para cada modelo es la necesidad de un enlace disponible e inmediato a la red. Los modelos dependientes de la red tienen la facilidad de colas de mensajes que permiten posponer efectivamente la terminación de una interacción si no hay enlace disponible.

El ambiente de ejecución incluye además los servicios de núcleo, entre los cuales los más importantes son los de manejo de nombres lógicos y seguridad. Los servicios de administración de nombres (usualmente llamados directorios), mapean los nombres lógicos de los módulos a sus direcciones físicas en el ambiente. Los servicios de seguridad controlan el acceso a los módulos de aplicación.

Otros servicios de núcleo incluyen administración de memoria distribuida, pasarelas a sistemas externos, y comunicaciones servidor a servidor, que pueden segmentar ambientes grandes en dominios.

El ambiente de ejecución incluye además servicios de aplicación que sirven de base a las funciones de aplicación, como acceso a bases de datos por SQL, procesamiento de transacciones, distribución de correo electrónico, flujo de trabajo y administración de documentos.

El middleware ofrece 10 modelos de aplicación, descritos en la tabla (ordenados del más específico al más abstracto desde el punto de vista del usuario).

Modelo de aplicación	Función
Almacene y envíe / Publica y subscribe	Dirige un mensaje al receptor lanzando los datos o atrayéndolos. (Es la base del correo electrónico)
Flujo de trabajo (Workflow)	Dirige los mensajes a los receptores de acuerdo a condiciones o políticas establecidas.
Transacciones distribuidas	Administra actualizaciones simultáneas a múltiples bases de datos bajo el control de transacciones.
Acceso remoto a archivos	Redirecciona las solicitudes de archivos a través de la red.
Acceso remoto a bases de datos	Transmite solicitudes de SQL a través de la red a los servidores.
Interacción de objetos distribuidos	Sirve de apoyo para el intercambio de mensajes entre objetos a través de la red.
Acceso a funciones remotas	Redirecciona las llamadas de programas a funciones a través de una red.
Administración de bases de datos distribuidas	Mantiene una única base de datos local a través múltiples de múltiples bases de datos físicas.
Replicación de bases de datos	Sincroniza copias de una única base de datos.
Presentación distribuida	Hace las funciones de presentación de una aplicación disponibles a un cliente remoto.

Figura 22: Modelos de aplicación del middleware

Las herramientas de middleware pueden encontrarse desde ambientes completos de desarrollo hasta sencillos API, siendo la principal diferencia cuan transparente será la computación distribuida.

5.2.5 Arquitectura cliente servidor de tres niveles (Three Tier)

La arquitectura cliente servidor de tres niveles se conoce generalmente como el enfoque de middleware, porque se basa en el middleware como un conjunto de librerías que se utilizan para desarrollar el software del cliente y el servidor. Estas librerías deben ser independientes de lenguaje, de la herramienta, del DBMS y del RDBMS y deben proveer gran flexibilidad desde el punto de vista del desarrollador de aplicaciones y enlaces para el control de operaciones, una herramienta para la depuración de programas, seguridad, control de versiones y distribución de software. Aquí, el middleware funciona básicamente como un controlador de mensajes y de ruteo. El cliente envía un solicitud (en forma de mensaje) al servidor, obtiene acceso a la base de datos y bien procesa el requerimiento, da formato a la respuesta y la envía de regreso al cliente.

5.2.6 DCE (Distributed Computing Environment) y CORBA

El DCE (Ambiente de Computación Distribuida) de la OSF (Open Software Foundation) es un producto de middleware que nació con el objetivo proveer una infraestructura común para la interoperabilidad de sistemas distribuidos a través de diferentes plataformas. Actualmente, ha sido tan aceptado que hay una gran variedad de

productos nuevos que utilizan la tecnología de DCE para la integración de sistemas. De hecho, DCE se ha convertido en un estándar de facto para middleware.

DCE provee los servicios claves requeridos para el soporte de aplicaciones distribuidas: como llamadas a procedimientos remotos (RPC) entre clientes y servidores, un directorio de servicios para permitir a los clientes encontrar a los servidores y servicios de seguridad para verificar el acceso seguro entre clientes y servidores.

La tecnología de DCE está basada en implementaciones de RPC. El DCE permite a los componentes del servidor a operar threads[18]. Un thread es una secuencia independiente de operaciones o instrucciones en un proceso. Un proceso puede mantener simultáneamente varios threads. Los threads de DCE permiten a múltiples llamadas de procedimiento de los clientes a la misma aplicación, sin requerir que el servidor cargue el mismo proceso una y otra vez, y como consecuencia, los servidores son mucho mas eficientes.

El DCE también ofrece directorios de servicios, un sistema de archivos distribuido (DFS: Distributed File System), y servicios de tiempo distribuido (DTS: Distributed Time Services). La figura muestra el ambiente de computación distribuida creado por OSF a través de DCE.

Figura 23: Arquitectura de DCE

Una variante del DCE es el middleware orientado a mensajes, que es típicamente asincrónico y punto a punto. El control del programa no se transfiere explícitamente de una porción de la aplicación a otra. Esto hace más sencillo distribuir el control entre los componentes de la aplicación que interactúan. Cuando recibe solicitudes de una aplicación, el middleware orientado a mensajes es responsable de ver que el destinatario reciba el mensaje. Si no está disponible, lo intenta varias veces, y si no es posible, envía un mensaje de error al emisor. La mejor aplicación del middleware orientado a mensajes son las aplicaciones orientadas a eventos, y es muy compatible con los sistemas orientados a objetos, porque aplica el mecanismo conceptual de la comunicación entre objetos.

La evolución actual del DCE es el soporte de objetos (considerando un objeto como un conjunto de datos y métodos, enlazado con sus propiedades de encapsulamiento, herencia y

---

[18] En inglés *threads* significa hileras o enlaces, el concepto equivalente en sistemas operativos no tiene una traducción aceptable.

polimorfismo[19]) y se han considerado varias opciones para ello, una de ellas es la integración de CORBA al producto. CORBA (que se deriva de las siglas en inglés de Common Object Request Broker Architecture), es un conjunto de estándares para interfaces para varios sistemas distribuidos, producidos por el Grupo para el Manejo de Objetos (OMG). Como el DCE, las especificaciones de CORBA están orientadas a definir una solución abierta para construir una infraestructura distribuida común. Sin embargo, difiere de éste en que es explícitamente orientado a objetos, y los diferentes estándares de CORBA definen únicamente interfaces y no implementaciones específicas.

De hecho, DCE incluye ya grandes porciones que se califican como orientadas a objetos. Un servidor DCE puede visualizarse como un agrupamiento de datos y métodos (en la forma de procedimientos remotos), además, DCE ya soporta encapsulamiento y polimorfismo, dos de las tres características requeridas para ser orientado a objetos. El encapsulamiento se puede ver en la forma en que se relacionan un cliente y un servidor, utilizando el lenguaje de definición de interfaz (IDL). El polimorfismo puede verse en la forma en que un servidor de DCE puede permitir a dos o más conjuntos de implementaciones de procedimientos remotos ser accesados a través de la misma interfaz. Un cliente identifica cuál de las implementaciones desea invocar especificando un identificador universal único (UUID) para un objeto particular. Sin embargo, DCE no soporta herencia.

Los estándares de CORBA definen un ambiente para la computación de objetos distribuidos, un mundo en el cual los objetos pueden comunicarse como clientes y servidores. Una pieza clave en este ambiente el Agente de solicitud de Objetos (de ahora en adelante citado como ORB, por sus siglas en inglés *object request broker*), quien provee los procedimientos por los cuales los clientes envían y reciben solicitudes y respuestas. Un cliente invoca un método en algún objeto a través del ORB, quien traslada la solicitud al objeto apropiado. Este método se ejecuta y el resultado (si lo hay) es enviado de regreso, de nuevo a través del ORB. Toda la comunicación entre los objetos reales en un ambiente CORBA se apoya en el ORB.

Si aplicamos el concepto a la integración de sistemas heterogéneos, el ORB verá a cada sistema de bases de datos como un objeto.

CORBA define dos formas diferentes para que los clientes invoquen operaciones en los objetos del servidor. La primera, se conoce como una interfaz estática, funciona de forma muy similar al RPC. Un objeto define una interfaz utilizando el lenguaje de definición de interfaz (IDL). Esta definición es compilada para producir un canal de interfaz para el cliente y un esqueleto para el servidor, que es código que usualmente se enlace a los servidores del cliente y el servidor respectivamente. Para invocar un método en el objeto del servidor, el cliente hace un llamado a una función, solicitud que, a través del ORB, es recibida y ejecutada en el objeto destino. El cliente se bloquea hasta que la función retorna una respuesta de modo que esta interfaz es muy similar a los procedimientos remotos del DCE.

---

[19] Se recomienda consultar bibliografía sobre análisis orientado a objetos para comprender a fondo las implicaciones de esto.

Una alternativa para los clientes se conoce como la interfaz de invocación dinámica (DII por sus siglas en inglés de *Dynamic Invocation Interface*). Con DII, un cliente hace una solicitud en forma dinámica , construyendo los parámetros uno a uno, y luego la envía al objeto destino, quien ejecuta el método requerido y si hay algún resultado, es enviado de regreso. La variante es que el cliente que hizo la solicitud puede elegir entre bloquearse esperando el resultado, o continuar ejecutándose sin espera, en cuyo caso, hará un chequeo posterior (periódico) para revisar si la respuesta ha llegado. La ventaja del DII es que un cliente no necesita el canal de interfaz predefinido que se utiliza en la interfaz estática, y dado que puede construir las solicitudes en forma dinámica, significa que puede ser utilizado para comunicarse con objetos completamente nuevos mientras se ejecuta (no es necesario parar el procesamiento del cliente, compilar de nuevo los procesos, generar nuevos canales y volver a arrancarse).

Para soportar esta forma dinámica de operar, CORBA utiliza un repositorio de interfaces, que contiene las interfaces de los objetos en el ambiente del ORB. Cuando el cliente encuentra un objeto o clase nueva, puede consultar el repositorio para obtener la interfaz del objeto, "aprenderla" y luego utilizar el DII para construir e invocar en forma dinámica solicitudes sobre el nuevo objeto.

Para asegurar la interoperabilidad, el estándar de CORBA requiere que la comunicación entre ORB esté basada de algún modo en el Protocolo de Internet Inter-ORB (IIOP por sus siglas en inglés de *Internet Inter-ORB Protocol*) y en forma optativa por Protocolos Inter-ORB de ambientes específicos (ESIOP : Environment specific Inter-ORB protocols).

5.2.7 La perspectiva de Microsoft: OLE

La perspectiva análoga a la de DCE y CORBA es la implementada por Microsoft con su tecnología OLE (siglas en inglés de *Object Linking and Embedding* que significa enlace e implante de objetos) fundamentada en COM, (siglas en inglés de *Component Object Model,* modelo de componentes de objetos).

La familia de productos basados en la tecnología OLE incluye una gran variedad de productos que pueden desde crear documentos que combinan un procesador de palabras con una hoja electrónica, hasta interfaces estándares para el acceso a bases de datos.

COM especifica reglas y provee servicios para definir y utilizar los objetos, basados en la reutilización de los servicios. Como CORBA, tiene un lenguaje de definición de interfaz (IDL) para especificar las interfaces de los objetos e incluye servicios que permiten a los objetos instanciarse y comunicarse entre sí, que hacen específicamente lo que el ORB hace en el ambiente CORBA.

OLE puede describirse como una tecnología objetos de sistema extensibles. Una tecnología de objetos de sistema significa que puede combinarse los principios orientados a objetos en el contexto de un sistema operativo encapsulado, polimórfico y reutilizable, y que los componentes reutilizables pueden coexistir e interoperar como entidades binarias. Nuevos componentes, desarrollados e instalados en cualquier momento, pueden ser agregados al sistema en línea, de forma que inmediatamente se extiendan los servicios

ofrecidos a las aplicaciones e inclusive a las aplicaciones que ya se están ejecutando. Un sistema que soporta componentes de software debe soportar una abstracción de servicios genérica, o sea, una arquitectura que defina como todos los tipos de componentes se presentan y como son manipulados.

COM es una combinación de especificaciones y estándares de implementación que enfrenta los principales problemas que se presentan en una arquitectura de componentes de sistema[20]:

Problema	Solución
Dependencia de rutas de acceso (Path) y múltiples proveedores de un servicio.	Utilizar un registro de componentes para mapear de identificadores de clases abstractas hacia localidades absolutas del servidor, así como entre categorías e identificadores de clases (de objetos).
Definición de identificadores descentralizada.	Uso de Identificadores globales únicos (GUID) generados por un algoritmo para garantizar unicidad en tiempo y espacio, eliminando la necesidad de un mapeo centralizado de identificadores.
Manejo específico de API para cada categoría de servidos.	Proporcionar un API de administración muy genérico y universal que acomoda todas las categorías de servicios, llamado "Localidad de Implementación".
Compartir instancias de objetos entre procesos y máquinas.	Implementar un controlador de punteros de interfaz con transparencia local/remota proporcionando la capacidad de implementar un servidor como un .EXE o un .DLL.
Modelos diferentes de programación entre procesos, local y remotos.	Diseñar un modelo único para todos los tipos de conexiones cliente/objeto soportados a través de una estructura de interfaz de transparencia local/remoto.
Administración del ciclo de vida de servidores y objetos. Múltiples servicios por módulo de servidor.	Implantar conteo de referencias universales a través de la interfaz básica desconocida (la clase básica) que todos los objetos soportan y desde la cual todas las otras interfaces se derivan.
Control de versiones	Soportar el concepto de interfaces múltiples e inmutables, así como interfaces que están fuertemente tipificadas tanto en tiempo de compilación como en tiempo de corrida.

Figura 24: Problemas en una arquitectura de componentes

OLE está construido sobre la arquitectura COM. Todas las funciones de OLE pueden clasificarse en alguna de las tres categorías siguientes:

1. Funciones API e interfaces para los servicios nativos de OLE incluyendo muchas funciones de ayuda y objetos de ayuda.
2. Funciones API e interfaces para la personalización de los servicios nativos.
3. Funciones API e interfaces para la creación de servicios personalizados de acuerdo a varias especificaciones.

---

[20] Si desea ampliar la información sobre la arquitectura básica de COM y las estructuras fundamentales, consulte el documento What Ole is Really About. (ver bibliografía)

OLE provee los siguientes servicios nativos:

- Implementación de localidad
- Transparencia local/remota
- Asignación de memoria para tareas
- Almacenamiento estructurado
- Tabla de objetos en ejecución
- Librería de tipos, creación y administración de información de tipos
- Rutinas de conversión de tipos
- Intercambio de datos a través del portapapeles ("cut and paste")
- Intercambio dinámico de datos ("Drag-and-drop")
- Caché de datos

Los primeros tres servicios son fundamentales para dar a los objetos y a los clientes la habilidad de comunicarse a través de cualquier distancia. Son parte del núcleo de una implementación COM. Los otros servicios son implementaciones de un estándar y soportan interoperabilidad de alto nivel. Por ejemplo, el almacenamiento estructurado describe un servicio de "sistema de archivos dentro de un archivo" que estandariza la forma en que los componentes de software pueden cooperar en el proceso de compartir un archivo de disco, y dado que ésto es indispensable para la interoperabilidad, OLE implementa un servicio estándar llamado "Archivos Compuestos".

En base a estos servicios básicos, se han construido servicios personalizados (que son especializaciones de estos servicios básicos) para cualquier necesidad particular, una de ellas es el intercambio de datos en tiempo real, que puede ser una solución muy elegante para la integración de sistemas heterogéneos, en la cual los objetos hacen accesos directos a las bases de datos y los clientes las aplicaciones de usuario final.

La idea básica es que, no importando quién define el servicio, muchas personas pueden implementar clientes u objetos de acuerdo a la especificación, que describe la abstracción que cualquiera de los dos lados (el cliente o el objeto) utiliza para ver al otro, de modo que todos los clientes son polimórficos para los objetos y todos los objetos de la categoría son polimórficos para los clientes.

Lo más importante es que COM es una arquitectura abierta en la cual cualquiera puede crear y definir una nueva categoría de servicios. Crear "componentes" nuevos que permitan a una aplicación comunicarse fácilmente con cualquier fuente de datos sin necesidad de diseñar de nuevo la aplicación, pues esta no es más que un conjunto de conexiones entre componentes.

5.2.8 Compuertas (Gateways)

Las compuertas son una solución más rígida al problema de la integración, y es "simular la homogeneidad estricta en ambientes heterogéneos".

Para explicar el concepto, supongamos que existen dos localidades que se desean integrar, X y Z. La localidad X está compuesta por un sistema de bases de datos ORACLE y Z tiene un sistema en INFORMIX. Como primer paso, debo elegir un sistema base (es decir, utilizaremos todos el DBMS de X o el de Z). Si la elección es INFORMIX, todo el sistema debe programarse de modo que toda la aplicación "crea" que los datos están sobre INFORMIX.

Para ello, INFORMIX debe proporcionar un pequeño módulo de aplicación que se ejecuta sobre ORACLE (en el sitio Y) que hace que éste parezca INFORMIX. Este programa se conoce como Compuerta (o Gateway) y sus funciones básicas son las siguientes:

1. Proporcionar protocolos para el intercambio de información entre los dos sistemas, lo cual implica la traducción de proposiciones de consulta y manejo de datos y de respuestas.
2. Ofrecer la función de Despachador de consultas (una función parecida a la del ORB descrito anteriormente), para el manejo de SQL dinámico.
3. Correspondencia de tipos (identificación semántica).
4. Traducción de sentencias SQL.
5. Correspondencia de la información de control entre los dos sistemas.
6. Correspondencia entre los dos diccionarios de datos.
7. Funcionar como agente en el manejo de transacciones entre los dos sistemas (verificando la integridad de las transacciones por medio de protocolos de compromiso de dos fases).
8. Control de bloqueos entre los dos sistemas.

Este tipo de solución, aunque es más cerrado (generalmente solo se soporta para sistemas relacionales), ofrece mayor control para el desarrollador de aplicaciones, pues permite (dado que se eligió un DBMS estándar), programar como si existiera un único DBMS y se desarrollara para un sistema sencillo.

5.2.9 Sistemas de administración de múltiples bases de datos (MDBMS)

Un sistema de administración de múltiples bases de datos (desde ahora referido como MDBMS) permite realizar operaciones a través de múltiples componentes de base de datos autónomos. Provee a los usuarios con un sistema de una vista única de los datos distribuidos en una gran cantidad de bases de datos y sistemas de archivos heterogéneos. El MDBMS interopera con los componentes individuales de base de datos en forma similar a la que un procesador de SQL en un DBMS interopera con el sistema de almacenamiento de registros.

Un MDBMS relacional se compone típicamente de las siguientes unidades de procesamiento:

- Parser de SQL e Interfaz de Lenguaje de Aplicación (API)
- Sistema de datos relacionales
- Administrador de catálogos globales
- Compilador y optimizador de Queries distribuidos
- Sistema de ejecución distribuido
- Administrador de transacciones distribuidas
- Compuertas para obtener acceso a diversas fuentes de datos

Los catálogos para la metadata pueden estar organizados en varias formas, las dos mas comunes son el catálogo autónomo y el catálogo integrado.

En un catálogo autónomo, el MDBMS mantiene su propio catálogo en una base de datos separada. Este catálogo describe los datos disponibles en el sistema de múltiples bases de datos. Para todos los datos que residen en una base de datos relacional, las definiciones de metadata de objetos de tabla, índices y en adelante, son importados (replicados) al catálogo. Para datos que residen en otras fuentes de datos (por ejemplo hojas electrónicas y archivos planos), el catálogo del MDBMS contiene una descripción relacional para esa fuente de datos.

En un catálogo autónomo, el MDBMS está integrado con un sistema de bases de datos que es capaz de obtener acceso a objetos (datos y metadata) en bases de datos remotas y distintas. Un servidor de compuertas es responsable de hacer parecer la base de datos distinta como homogénea (aunque remota). Para los datos almacenados en bases de datos relacionales, el servidor de compuertas almacena vistas de las relaciones del sistema en esa base de datos. Para datos residentes en un sistema de archivos o en una hoja electrónica, el servidor de compuertas almacena la descripción de la metadata de estos datos en una base de datos separada.

En el siguiente capítulo se describirá a detalle DBI de Digital, que es un integrador basado en el concepto de MDBMS, y se ampliará la arquitectura básica, así como la estructura completa del sistema.

5.2.10 ODBC

Este estándar, llamado así por sus siglas en inglés *Open database Connectivity (ODBC)* fue diseñado por Microsoft para permitir a cualquier aplicación comunicarse con cualquier manejador de bases de datos. Se basa en el trabajo del SQL-Access Group, un grupo de proveedores de hardware, software y productos de red, que definieron un método común de acceso a bases de datos para simplificar la computación en ambientes cliente/servidor. El objetivo principal de este grupo era definir una sintaxis de SQL que fuese común a todos los manejadores de bases de datos. Microsoft tomó esta idea básica y desarrolló una interfaz de nivel de llamadas que llamó ODBC, a través de la cual el usuario final tiene acceso a datos centralizados directamente desde el escritorio del usuario.

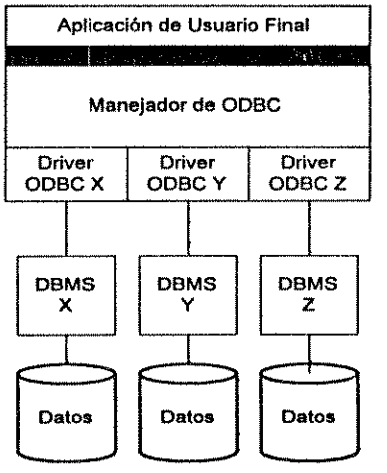


Figura 25: Arquitectura de una aplicación que utiliza ODBC

El estándar ODBC define tres niveles de compatibilidad para API: núcleo, nivel 1 y nivel 2. El API es un conjunto de 54 llamadas de funciones, de las cuales 23 pertenecen al núcleo, 15 al nivel 1 y 16 al nivel 2. Se dice que un driver[21] es conforme a un nivel si implementa todas las funciones de ese nivel y de los inferiores. Un sistema similar de tres niveles se aplica al conjunto de construcciones de SQL que son soportadas.

La arquitectura de ODBC se resume en la vista esquemática del interfaz, presentada en la gráfica. Las aplicaciones hacen llamadas a un manejador del dispositivo de ODBC (comúnmente implementado como un DLL en ambientes Windows). Este manejador del driver carga el driver de ODBC apropiado para la base de datos solicitada y hace las llamadas apropiadas a éste. La base de datos es accesada en forma relacional haciendo cadenas de comandos SQL.

La arquitectura ODBC tiene cuatro componentes básicos: la aplicación, el manejador de drivers, los drivers, y las fuentes de datos. Una aplicación que utiliza interfaces ODBC provee usualmente las siguientes utilerías al usuario:

- Procedimientos de conexión y desconexión de un DBMS
- Ejecución de queries y áreas de almacenamiento y formato de resultados de consultas.
- Procesamiento de transacciones en línea
- Características externas a la aplicación ODBC.

Hay dos tipos básicos de driver, el dispositivo de un nivel, que manipula los archivos de bases de datos directamente (que son utilizados para bases de datos nativas para

---

[21] Comúnmente se llama *driver* a un dispositivo de software que proporciona un conjunto de funciones.

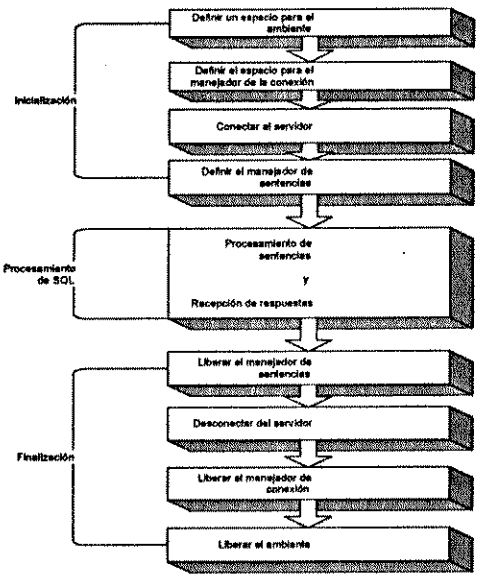


Figura 26: Control de flujo de un aplicación ODBC

computadoras personales, como xBase, Paradox y Btrieve; y drivers de múltiples niveles que generan solicitudes de SQL que son enviadas al servidor de bases de datos para ser procesadas.

Los drivers de múltiples niveles (multi-tier drivers) vienen en muchas formas, algunos de los cuales requieren el uso de el software de red del proveedor del sistema de bases de datos (como Ingres/Net o SQL*Net). El driver es un API de Windows que se carga por el manejador de drivers de ODBC. Este driver envía la solicitud de SQL al software de red del proveedor del sistema de bases de datos, que se comunica con el software de red del anfitrión (la máquina servidora de la base de datos que se desea accesar).

5.2.11 Configuraciones comunes de ODBC:

Configuración 1: El DMBS se encarga de la comunicación cliente/servidor

Este esquema se basa en los productos para el procesamiento distribuido disponibles para la mayoría de los DBMS del mercado (como INFORMIX/NET de Informix, SQL*Net de Oracle) y en una arquitectura de múltiples niveles, utilizando un driver que se comunica con dicho producto. Estos productos usualmente solamente permiten que un producto del mismo proveedor se conecte con un servidor del mismo tipo. En este caso, el cliente puede utilizar el dispositivo de ODBC para la(s) base(s) de datos que desea accesar, que serán administrados por un manejador de drivers de ODBC (véase la figura 14).

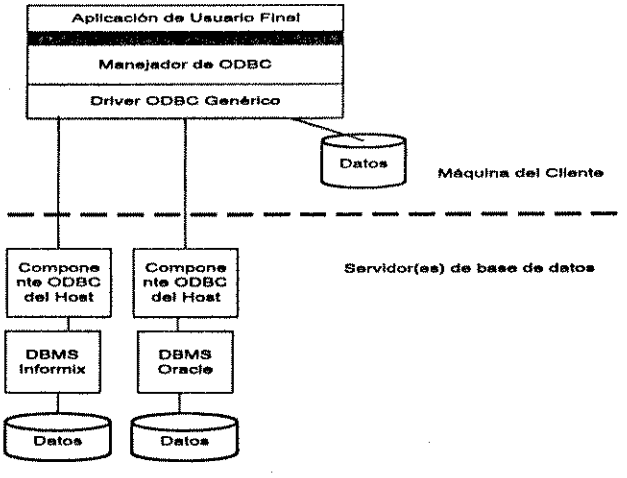


Figura 27: Configuración de ODBC Genérico

Configuración 2: ODBC Genérico

Estos drivers genéricos no requieren el software de red de los proveedores de DBMS, pues tienen uno o mas componentes que se ejecutan en la máquina cliente y otros componentes que se ejecutan en el o los servidores de bases de datos.

Una ventaja particular de esta configuración es que solamente se requiere un driver en el cliente, sin importar el número de servidores de bases de datos a los que se desea conectarse.

Configuración 3: Agente de Solicitudes

En este caso, las aplicaciones compatibles con ODBC establecen una sesión con el ODBC genérico. El agente de solicitudes (en el cliente) establece entonces una sesión con la base de datos a través de el receptor y distribuidor de solicitudes (request broker) en el servidor. Esta configuración está basada en el middleware orientado a objetos. El receptor esparce o replica uno o más agentes de base de datos y los asocia con el agente de solicitudes. Los agentes de base de datos son los únicos que son componentes específicos de la base de datos, y actúan como clientes de los procesos del DBMS de la base de datos que se desea consultar.

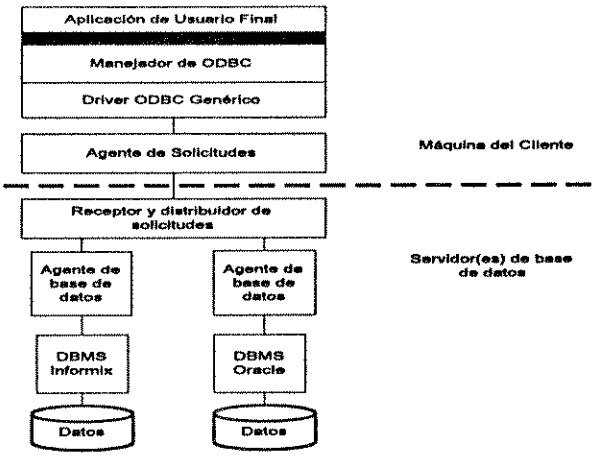


Figura 28: Agente de Solicitudes

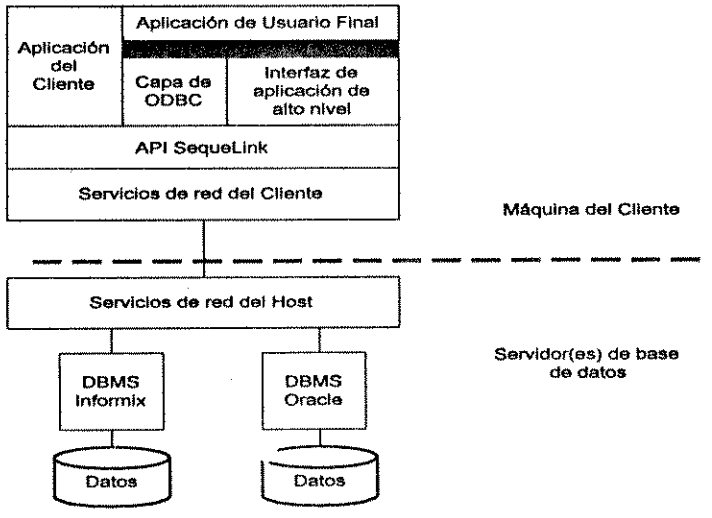


Figura 29 :Servicios de red

Configuración 4: Uso de los servicios de red.

Esta configuración soporta múltiples aplicaciones concurrentes de cliente a través de una instancia única del driver. Una variante de esta configuración utiliza permite utilizar API propietarios además del ODBC y se comunica a más bajo nivel (utilizando los servicios de la capa de red).

5.2.12 Internet como Middleware

Las tendencias actuales para la integración de sistemas de bases de datos utiliza el concepto de Internet (e Intranets) para crear un nuevo modelo de middleware. Este concepto se ha extendido ampliamente, y ofrece la comunicación abierta (interoperabilidad entre dominios heterogéneos), escabilidad e integridad que el middleware intenta proveer, de una forma transparente. Una ventaja es que la mayoría de sistemas operativos existentes, ya ofrecen el acceso a Internet (y las intranets para redes privadas) como parte de los servicios básicos, lo cual soluciona el problema de comunicación entre máquinas.

Este nuevo modelo de middleware que debe crearse deberá apoyarse en una orientación a objetos, fuertemente integrado y con interfaces suficientemente transparentes al usuario.

El objetivo del middleware es transportar datos a través de múltiples plataformas. La tecnología Internet ya ofrece esta conectividad, y la tendencia es (a través de la estandarización hacia los lenguajes de programación interplataforma como Java) la creación de aplicaciones en API comunes con transparencia en la plataforma, las herramientas y las bases de datos. Estas aplicaciones pueden integrarse a través de CORBA (que es a su vez un tipo de middleware).

CORBA ofrece dos componentes muy importantes para la construcción de aplicaciones dinámicas en Internet:

1. ORBs para el ruteo de mensajes

2. IIOP (siglas en inglés para *Internet Interoperable ORB Protocol)* el protocolo de comunicación que permite que dos ORBs se comuniquen a través de ambientes distribuidos en forma transparente, que pasa aún sobre los protocolos CGI (Common Gateway Interface) y HTTP, eliminando una capa completa de software.

Estas aplicaciones que integran sistemas heterogéneos se construyen definiendo módulos activos (tanto en el cliente como en el(los) servidor(es)), que se conocen como *applets* que contienen las funciones que cada objeto ofrece como interfaz. Sin embargo, es necesario que estas applets incluyan servicios de traducción de SQL, procesamiento de consultas, de RPC y de mensajes. Para modularizar esto, deberá construirse un modelo que pueda integrar niveles de funciones básicas: Comunicación (implícita en Internet), Presentación y Acceso a Datos.

La herramienta más factible para la construcción de esta tecnología es Java[22], que ofrece las utilerías necesarias para accesar las capas de red de las redes locales (LAN) o redes de área extendida (WAN).

5.2.13 JDBC

JDBC (siglas en inglés para Java database Connectivity) es un tipo de middleware alternativo para el ODBC y está basado en los mismos principios. La diferencia entre ambos es que las implementaciones de ODBC están realizadas en lenguaje C (dependiente de la plataforma) y JDBC es código nativo Java, totalmente abierto, y es la solución para la construcción del middleware para Internet.

JDBC transporta cualquier query en forma de cadena de caracteres a un driver de DBMS, de modo que una aplicación puede utilizar toda la funcionalidad del SQL.

JDBC está constituido por una serie de interfaces abstractas de Java que permiten a abrir conexiones a bases de datos particulares, ejecutar sentencias SQL y obtener resultados. Las interfaces más importantes son:

---

[22] Java fue desarrollado por SUN Microsystems en 1995 y por sus ventajas al ser independiente de plataformas, orientado a objetos y de propósito general se ha popularizado y estandarizado como herramienta para la construcción de applets (APIs dinámicos).

1. El Manejador de Drivers de SQL (java.sql.DriverManager) que administra la carga de los drivers de DBMS y provee soporte para la creación de nuevas conexiones.
2. Conexión SQL (java.sql.Connection) que representa una conexión a una base de datos particular.
3. Sentencias SQL (java.sql.Statement), que actúa como contenedor para una sentencia SQL en ejecución en una determinada conexión y tiene derivaciones para sentencias pre-compiladas, para llamadas a procedimientos almacenados y para la obtención de resultados.

Existen tres tendencias para la construcción de aplicaciones para sistemas de múltiples bases de datos utilizando JDBC:

1. **Sistemas de un nivel:** cuando el driver JDBC está escrito completamente en Java. Por ejemplo, en sistemas unificados (no en red), el código de la aplicación del cliente, el manejador de drivers JDBC y el Driver(s) JDBC están todos en el mismo nivel, en la máquina del cliente. En sistemas tipo intranet/Internet, el código de la aplicación del cliente (en Java), el manejador de drivers y el (los) driver(s) JDBC son accesados a través del servidor de Web. El cliente de la base de datos puede conectarse a una base de datos remota en el mismo hostal desde el cual obtiene las clases (el servidor de WEB).
2. **Sistemas de dos niveles:** son aplicables cuando el driver de JDBC requiere una librería de código nativa para traducir las funciones de JDBC a un lenguaje de consulta específico de un DBMS, en este caso, el código de la aplicación de Java, el manejador de drivers y el (los) drivers de JDBC conforman un nivel. La librería específica del DBMS son un segundo nivel (que normalmente reside únicamente en el servidor de bases de datos).
3. **Sistemas de tres niveles:** utiliza un nivel intermedio entre los dispositivos de JDBC y el DBMS para crear sesiones de comunicación entre la aplicación y el DBMS, y manipular el flujo de información desde y hacia el DBMS.

5.2.14 Evaluación de Cargas

Debe considerarse además cuáles son los requerimientos de rendimiento del sistema, pues cualquier aplicación para la integración de datos incrementará la carga total del sistema, específicamente en el tráfico de la red, carga en las bases de datos de producción y demandas en el sistema mientras se ejecuta la aplicación.

Es necesario balancear la capacidad y las demandas actuales del sistema con los tipos de cargas que los diferentes productos de integración de datos le incrementarán:

1. **Carga en la red:** Una de las consideraciones más importantes, en la evaluación de los productos para integración de datos, es la técnica que utiliza para reducir la carga que agrega a la red. Los productos de integración pueden utilizar una de las siguientes técnicas para transferir la menor cantidad de datos a través de la red y reducir esta carga:
   - Forzar el procesamiento de los queries en el servidor de bases de datos remoto
   - Procesar los joins de datos de tablas que pertenecen a las mismas bases de datos en el servidor que las contiene.

- Optimización de queries.

2. **Carga en la base de datos:** cada operación que se ejecuta en el DBMS genera un proceso que consume tiempo del procesador de la máquina servidora de bases de datos. Algunas técnicas que se utilizan para reducir el número y el tipo de solicitudes hechas a la base de datos remota son:
   - Enviar las solicitudes de ordenamiento de datos a la fuente de datos remota, únicamente cuando ésta posee los índices apropiados en el orden requerido.
   - Utilizar un query remoto único cuando la consulta debe buscar un número grande de filas (registros), lo cual reduce la sobrecarga necesaria para parsear, optimizar y ejecutar los queries múltiples.
3. **Carga de integración:** se refiere a la carga que el mismo producto de integración de datos añade al sistema local, a menor carga, menor tiempo de respuesta para las solicitudes de integración de datos. Algunas técnicas utilizadas para reducir la carga de integración son:
   - Decidir el ordenamiento más rápido para hacer join de tablas considerando el número de filas presentes (la cardinalidad) y el número de filas a retornar (selectividad) desde tablas remotas.
   - Reconocer los índices de las tablas remotas y seleccionar la mejor forma de efectuar la consulta basándose en ellos.
   - Implementar índices dinámicos cuando no pueden utilizarse (o no existen) índices remotos.

6 DESCRIPCIÓN DE PRODUCTOS PARA SOLUCIONES DE INTEGRACIÓN

Este capítulo describe tres productos que pueden ser considerados para la integración de sistemas: DBIntegrator de Digital Research, las soluciones Dharma y la tecnología OLE DB de Microsoft.

6.1 DBIntegrator

DB Integrator es un producto de la compañía Digital Research, basado en la arquitectura de MDBMS. Entre sus principales características ofrece optimización de queries heterogéneos, transparencia de localidad, consistencia global, resolución de diferencias semánticas y controles de seguridad.

Recordando la definición de un MDBMS, podemos decir que DBI es un sistema federado de múltiples bases de datos heterogéneas, levemente acoplado pues permite que cada componente sea autónomo). El enfoque de DBI a la administración de múltiples bases de datos pone énfasis en las siguientes características:

1. **Administración global y autónoma del catálogo de metadata:** esto le permite a DBI aparecer como un único servidor de integración. Las aplicaciones integradas no necesitan conocer en ninguna forma las complejidades de la distribución real de los datos. El catálogo global de DBI es un repositorio en el cual se conservan las descripciones de los datos distribuidos. Éste permite a DBI proveer a las herramientas y aplicaciones un punto único de acceso al ambiente de bases de datos federadas y puede ser administrado como un base de datos independiente, de hecho puede residir en un DBMS relacional: SYBASE, ORACLE y ser manipulado como tal. Proporciona además mecanismos para asegurar la alta disponibilidad del catálogo, como replicación del catálogo y discos espejo.
2. **Modelo de integración de tres niveles:** en un modelo de integración de dos niveles (uno en el servidor y otro en el cliente), una vez que los datos se obtienen del servidor, toda la integración se realiza en el cliente. Esto puede provocar sobrecarga de operaciones en el cliente, en un modelo de tres niveles, la capa intermedia (donde DBI está ubicado) es quien efectúa la mayor parte de procesamiento de integración, reduciendo la transferencia innecesaria de datos hacia el cliente, y simplifica la configuración del cliente (no necesita conectarse con todas las fuentes de datos, sino únicamente con la capa intermedia).
3. **Modelo de dispositivos de compuerta sencillos** (e integrados) para el acceso a las fuentes de datos, inclusive otros sistemas DBI. El núcleo de DBI interactúa con las compuertas de cada lenguaje a través de una interfaz de datos (SDI: Strategic Data Interfaz) diseñada especialmente para ello. Un dispositivo de compuerta (gateway driver) se implementa como una capa relativamente liviana de software que es compatible con SDI y es responsable de manejar la traducción entre diferentes modelos de datos, lenguaje de consulta (dialectos de SQL) y otras características de los componentes.
4. **Soporte a las características de base de datos de distribuidas:** una base de datos múltiple de DBI se compone de un conjunto de tablas que DBI crea para mantener la metadata (es

decir el catálogo) y los datos distribuidos que están disponibles al usuario cuando se conecta al catálogo de DBI.

Además de los objetos regulares de un sistema de base de datos relacional (tablas, columnas, índices), DBI utiliza objetos, enlaces y agentes de despacho (en inglés se utiliza el término *Proxy*) para enlazar los componentes.

Un enlace le dice a DBI cómo conectar una fuente de datos (la base de datos enlazada). El objeto de enlace tiene tres componentes, el nombre del enlace, la cadena de acceso utilizada para conectarse con la base de datos enlazada y, opcionalmente, información de seguridad utilizada por el dispositivo de compuerta de DBI para información de autenticación al sistema enlazado. El agente de despacho se asocia con el objeto de enlace, y puede ser utilizado para especificar información de autenticación de usuario para enlaces individuales. Si no se utiliza el agente, los usuarios deben especificar la información de autenticación para una base de datos específica cada vez que se conectan a DBI.

DBI permite crear vistas para crear definiciones de tablas formadas por múltiples fuentes de datos y son un mecanismo muy útil para resolver diferencias semánticas entre diferentes bases de datos. Las vistas en DBI pueden ser de dos tipos: vistas regulares de SQL (compatibles con la especificación de ANSI SQL92) y vistas particionadas horizontalmente (HPV siglas en inglés para *Horizontally Partitioned Views*), que consisten de un nombre de vista, una columna de partición y las especificaciones de partición.

DBI soporta procedimientos almacenados (*stored procedures*), que permiten a los usuarios enlazar la lógica de la aplicación con la base de datos, haciendo que el código de la aplicación sea fácilmente compartible y facilitar a DBI mantener dependencias entre el código de la aplicación y los objetos base de datos, además de reducir el tráfico de mensajes entre el cliente y el servidor.

La arquitectura del sistema DBI se apoya en dos interfaces externas, la interfaz de SQL (utilizada por los clientes para generar solicitudes al servidor de integración) y las interfaces de dispositivos de metadata y los interfaces de dispositivos de datos (MDI/DDI), que son utilizados por DBI para hacer llamadas a dispositivos de compuerta (*gateway drivers*) a las bases de datos, así como dos interfaces internas, el DSRI (una interfaz relacional estándar entre el parser de SQL de DBI y el motor de procesamiento de DBI) y el SDI, que especifica una interfaz de datos estricta que protege el núcleo de DBI de las interfaces específicas de las fuentes de datos y facilita el desarrollo modular.

Los componentes principales de DBI son compatibles con las especificaciones de un sistema de múltiples bases de datos:

1. El ambiente SQL y ODBC cliente/servidor que provee un API de lenguaje y funciones para interpretación de SQL (soporta ANSI SQL y DEC SQL, que traduce a una lenguaje interno nativo de DBI).
2. El dispositivo API y el manejador de contexto apoyan la administración de transacciones distribuidas y parte del sistema de ejecución distribuido. El dispositivo API es responsable del control de flujo de alto nivel de las solicitudes de los clientes al núcleo de DBI. Acepta llamadas (en formato DSRI) de las aplicaciones y las despacha. El administrador de

contexto ejecuta la propagación (por demanda) del contexto de ejecución a los dispositivos de compuerta y mantiene el contexto distribuido de las sesiones activas, transacciones y solicitudes.

3. El manejador de metadata provee la administración de un catálogo global. Es responsable de la administración general y el acceso a la metadata. Los servicios que provee caen en las categorías de administración de catálogo, definición de datos, administración de caché de metadata y acceso (por queries) a las relaciones del sistema DBI.
4. El compilador soporta al optimizador de queries distribuidos. Provee servicios para trasladar sentencias SQL y procedimientos almacenados (*stored procedures*) a planes de ejecución DBI. El optimizador de queries (basado en reglas) permite la reescritura de los queries, enumeración de diferentes estrategias de ejecución y factores de capacidades funcionales de las diferentes fuentes de datos. Cada estrategia de ejecución se asocia con un costo basado en estimados de selectividad de predicados, cardinalidad de las tablas, disponibilidad de índices, ancho de banda de la red, entre otros, eligiéndose la estrategia del costo mínimo. El compilador genera código que puede ser procesado por el componente ejecutor y los drivers de compuerta.
5. El ejecutor sirve de apoyo a la parte restante del sistema de ejecución distribuido. Es responsable del procesamiento del plan de ejecución que el compilador produce. Estas actividades incluyen: intercambio de datos entre el DBI y el cliente, y entre el núcleo de DBI y las bases de datos enlazadas, ejecución de procedimientos de manipulación de datos inmediatos (como joins y funciones agregadas), administración del ambiente de trabajo y la pila de memoria intermedia (buffer) para manipular eficientemente grandes cantidades de datos intermedios, y soporte de procesamiento paralelo.
6. El despachador de SDI y los dispositivos de compuerta proveen acceso a las fuentes de datos. El despachador SDI separa el núcleo de DBI del espacio del dispositivo de compuerta. Localiza y carga imágenes compartidas que representan los drivers de compuerta y rutea las llamadas de SDI a las entradas correspondientes del dispositivo de compuerta.

6.1.1 Consideraciones técnicas de DBI.

6.1.1.1 Ejecución distribuida

Para el soporte del procesamiento de consultas distribuidas, DBI propaga el contexto de ejecución (contexto de conexión y/o contexto de la transacción) a las fuentes de datos objetivo. Las herramientas y las aplicaciones ven únicamente una sesión de usuario sencilla y una transacción establecida con el servidor de integración DBI.

DBI utiliza una organización de árbol para el control del contexto de ejecución distribuido. Cuando un usuario se conecta a la base de datos DBI, se crea una sesión de usuario. Este contexto de sesión es utilizado como base para las transacciones activas, sentencias de SQL precompiladas. así como el caché de metadata creado para cada usuario que se conecta con DBI. Cuando DBI cede el control al dispositivo de compuerta, se establecen el contexto de sesión y de transacción con la fuente de datos destino.

Con el objetivo de asegurar que las transacciones distribuidas soporten la concurrencia (y la consistencia) entre manejadores de bases de datos autónomos, DBI utiliza un administrador de transacciones distribuidas con protocolo de compromiso de dos fases (el DDTM *Digital Distributed Transaction Manager*), además el interbloqueo distribuido se soluciona mediante una técnica sencilla basada en tiempo expirado de una transacción.

Lo que distingue DBI de otras arquitecturas para integración de datos son los dispositivos de compuerta (*Gateway Drivers*), que fue diseñada para ser simples traductores de datos y protocolos. Su función principal es notificar las capacidades del interfaz de fuente de datos (API y lenguaje SQL) al núcleo DBI y posteriormente servir de herramienta para mapear entre el protocolo de la interfaz SDI y la interfaz de fuente de datos. Los dispositivos de compuerta usualmente comparten su contexto de proceso con el proceso del servidor DBI, reduciendo la necesidad de que un proceso servidor de compuertas intermedio resida entre el servidor DBI y el servidor de datos (como SYBASE), así como el cambio de contexto y la transferencia de mensajes entre procesos.

Los dispositivos de compuerta son responsables de traducir la semántica SDI a las primitivas de interfaz provistas por la fuente de datos objetivo (es decir al dialecto de SQL propio de las bases de datos).

6.1.1.2 Manejo de la metadata distribuida

La manipulación de la metadata se apoya en tres áreas importantes: manejo del catálogo, seguridad y caché de metadata.

Para asegurar la independencia de las bases de datos, DBI no requiere la presencia de un DBMS para el almacenamiento de la metadata, ya que fue diseñado para funcionar como una capa superior sobre los DBMS relacionales, de modo que se utiliza una interfaz de catálogo que permite elegir entre varios DBMS para almacenar el catálogo.

DBI utiliza los mecanismos de seguridad de los sistemas de bases de datos para autorización para la conexión de una base de datos a través de DBI y el acceso a datos a través de ella y autorización para la ejecución de varias operaciones de integración de base de datos (CREATE, DROP, SELECT, INSERT, UPDATE, DELETE).

El caché de metadata es una tabla en memoria que tiene un doble propósito, facilitar el acceso rápido a la metadata por el compilador DBI y servir como almacenamiento temporal para las relaciones del sistema DBI que pueden ser consultadas por las herramientas y aplicaciones. El caché de metadata está estructurado como una tabla de dispersión (hash) única que representa un espacio de nombres plano sobre todos los objetos DBI.

6.1.1.3 Procesamiento de Queries distribuidos heterogéneos

El procesamiento de queries distribuidos en un ambiente de bases de datos heterogéneas presenta ciertos problemas específicos: las fuentes de datos se comportan en forma distinta respecto al costo de la transferencia de datos y soportan diferentes construcciones de lenguaje. DBI provee un optimizador de queries que incluye algoritmos de

descomposición que reducen el flujo de datos y proveen una ejecución de queries de alto rendimiento.

Cuando un query tiene varias formas diferentes de producir el mismo resultado, el plan que tiene el costo mínimo es elegido. Se utilizan estadísticas por tabla, columna e índice para estimar el tamaño del resultado después de varias operaciones relacionales. El costo de transmitir los datos de la base de datos origen hacia el servidor de integración se consideran al elegir la forma de enviar el query a la base de datos de origen. El costo de transmisión en la red es medido en forma dinámica para cada sesión de usuario, una vez por cada conexión a la compuerta. El costo asociado con la ejecución de una operación relacional se agrega también al costo global.

DBI ofrece alta disponibilidad al utilizar vistas horizontalmente particionadas y replicación del catálogo. Una vista horizontalmente particionada (HPV *Horizontal partitioned view*) es un tipo especial de vista de donde DBI se provee de información sobre la forma como los datos están distribuidos en las tablas de las bases de datos integradas.

Adicionalmente al optimizador de queries distribuidos, DBI utiliza una serie de técnicas para incrementar la velocidad de procesamiento de los queries específicamente en las siguientes áreas:

1. **Transferencia de datos:** el motor de ejecución de DBI utiliza transferencia de datos en masa por medio de los mecanismos de búsqueda en masa (bulk fetch) que provee la interfaz SDI, que permiten que un único mensaje de solicitud retorne muchas tuplas en un único mensaje de respuesta.
2. **Administración de memoria:** para utilizar eficientemente la memoria del sistema operativo y reducir las fallas por falta de página, el ejecutor de DBI utiliza conjuntos de trabajo diferentes. Un conjunto de trabajo se organiza como una secuencia lineal de páginas de tamaño fijo. El mecanismo estándar de paginación identifica las páginas utilizadas y el estado de los registros indicando si determinada página está en memoria o si está en almacenamiento secundario. El administrador de conjuntos de trabajo controla el acceso equitativo a toda la memoria a través de los conjuntos de trabajo activos de un determinado usuario de DBI.
3. **Procesamiento de asociación (*Join Processing*):** DBI soporta los mecanismos regulares de asociación (join) y varias variantes híbridas.
4. **Paralelismo:** DBI utiliza dos tipos de paralelismo: independiente y direccionado (pipelined).
5. **Procedimientos almacenados:** presentan una mejora crítica en el rendimiento para el procesamiento cliente/servidor. Permiten al administrador de la base de datos encapsular conjuntos de sentencias SQL con lógica de control, a la vez que reducen el tráfico en la red.

6.1.2 Configuración del Servidor DBI

Un servidor de DBI se compone de un proceso monitor, un proceso despachador y un conjunto de procesos ejecutores de DBI. El proceso monitor soporta el sistema de administración en línea de la configuración del servidor. Uno o más procesos despachadores

manejan el contexto de comunicación de todos los clientes. Los despachadores rutean los mensajes del cliente a un proceso ejecutor de DBI apropiado a través de las colas de memoria compartidas de alta velocidad.

En el ambiente del servidor DBI, un cliente ODBC se conecta lógicamente a un servicio que provee acceso a una base de datos específica. Un servicio es instanciado por un conjunto de procesos ejecutores que contienen la imagen de DBI. El conjunto de procesos del conjunto es configurable, en línea y fuera de línea, para equilibrar éste número a los requerimientos de capacidad de procesamiento de una base de datos.

Los procesos ejecutores DBI pueden configurarse como reusables en sesión (que significa que un cliente está enlazado a un proceso ejecutor durante todo el tiempo que dura la sesión con la base de datos) o reusables en transacción (múltiples clientes pueden compartir el mismo proceso ejecutor, y un cliente puede utilizar un ejecutor por cada transacción que desee).

6.2 D-HARMA

Dharma Systems es una empresa pionera en el desarrollo de tecnología para interfaces SQL que transforma las bases de datos propietarias en sistemas abiertas, e incluye drivers ODBC, software de redes cliente/servidor, un motor de ejecución de SQL, herramientas SQL y dispositivos de compuerta. Los diferentes productos que ofrece se describen a continuación:

6.2.1 ODBC SDK Paquete de desarrollo de software:

El paquete de desarrollo de ODBC de Dharma, es la forma más rápida y sencilla para proveer conectividad a cualquier base de datos propietaria, para obtener interoperabilidad con una amplia variedad de herramientas para Windows y para Internet, accesibilidad desde cualquier plataforma cliente a cualquier sistema operativo de servidor, y agilidad en el proceso de desarrollo.

La tecnología de ODBC de Dharma (en inglés es conocida como "*near-shrink-wrap*"), está distribuida como un cliente (un DLL) que se ejecuta en clientes Windows 3.1, Windows 95 y Windows NT y una librería para enlace de servidores que se ejecuta en el servidor que sirve de anfitrión en el sistema de almacenamiento propietario, tal como se aprecia en la figura:

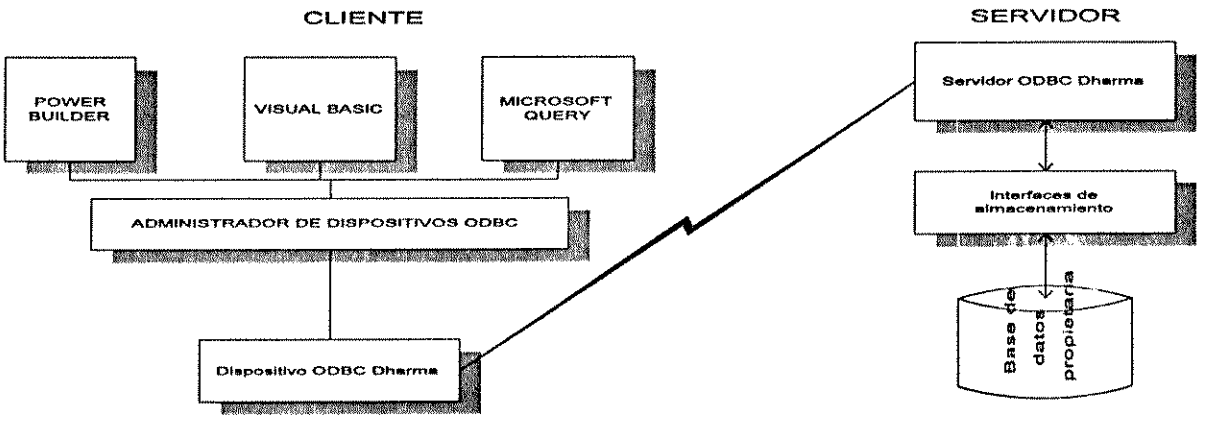


Figura 30: Configuración de la comunicación a través del ODBC de Dharma

El procedimiento para proveer acceso de ODBC a los datos de una base de datos propietaria tiene dos etapas: ingresar la metadata que describe el diseño de la base de datos y creación de plantillas de almacenamiento para obtener acceso a los datos en el sistema propietario.

El dispositivo de ODBC de Dharma/SQL es compatible con la versión 2.1 del estándar ODBC de Microsoft, soporta todas las funciones API del núcleo y del nivel 1 y toas las funciones de nivel 2 requeridas por las herramientas de Windows y de Web, ha sido probado como compatible con herramientas populares de desarrollo: PowerBuilder, Visual Basic, Visual C++, entre otros.

6.2.2 Dharma/SQL

El paquete de productos Dharma/SQL es una solución que puede ser adaptada a las necesidades de interoperabilidad y un lenguaje de consulta para sistemas propietarios, convirtiéndolas en sistemas abiertos. Provee un lenguaje estructurado de consulta estándar (SQL), Interconectividad de bases de datos abiertas (ODBC) y acceso de Web para cualquier sistema propietario.

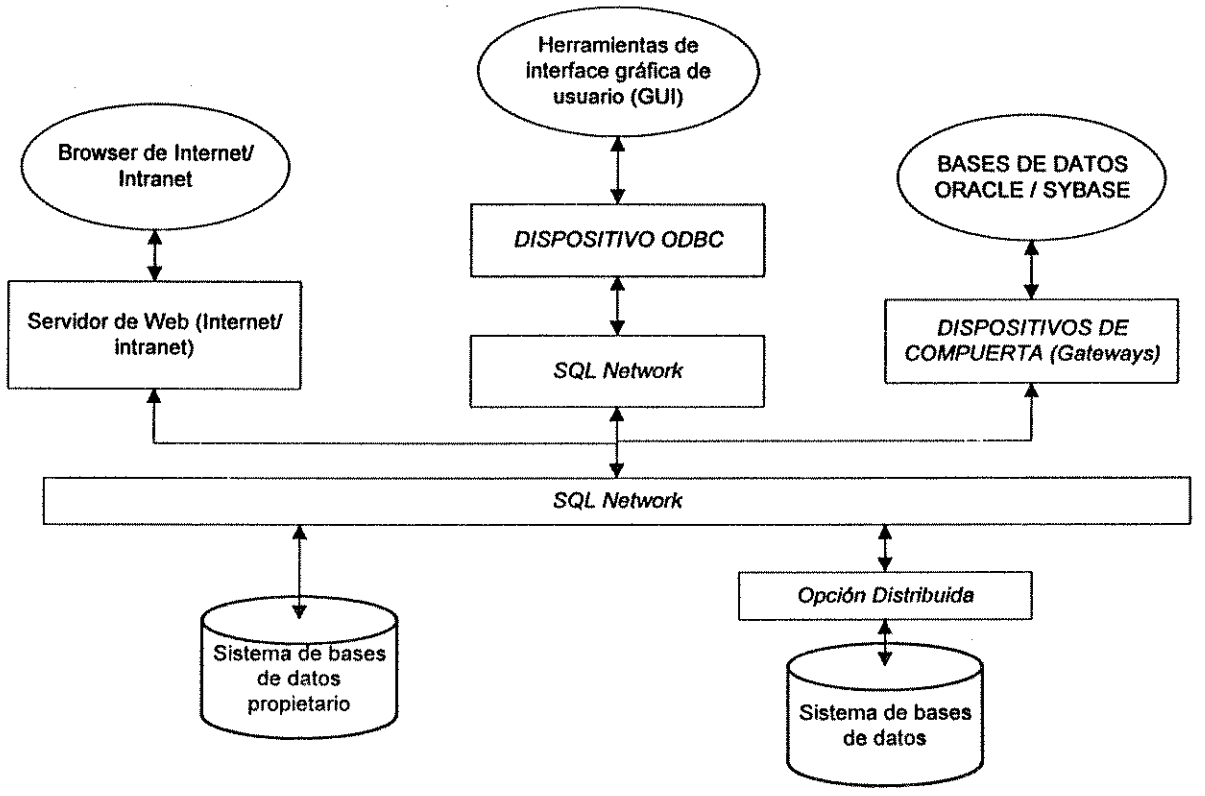


Figura 31: Componentes de Dharma/SQL

Dharma/SQL está basado en una tecnología de componentes SQL, los cuales son distribuidos en código fuente, lo que permite al desarrollador de software la flexibilidad para crear una gran variedad de soluciones.

Dharma/SQL Access es el componente fundamental del paquete. Provee una interfaz SQL compatible con las especificaciones ANSI/ISO SQL92, ODBC, JDBC y Oracle7, y provee un alto nivel de rendimiento y cualidades de optimización para el procesamiento de transacciones en línea y aplicaciones de soporte para la toma de decisiones. Dharma/SQL define un conjunto flexible de interfaces de acceso al almacenamiento que permiten obtener el máximo rendimiento posible de los sistemas de almacenamiento propietarios.

La opción distribuida permite a las aplicaciones obtener información de muchas bases de datos al mismo tiempo, y adicionalmente, permite combinar transparentemente estos datos (datos de múltiples bases de datos relacionales y múltiples sistemas propietarios), como una asociación heterogénea (en inglés *heterogeneous join*), lo cual simplifica el acceso a las bases de datos en la red creando una única "fuente de datos virtual", que es lo único que ven los usuarios.

El dispositivo ODBC de Dharma/SQL ofrece interoperabilidad del tipo "conecte y use" (en inglés *plug and play*), abriendo los sistemas propietarios y una gran cantidad de aplicaciones para Windows y herramientas para Internet/intranet en una gran variedad de

plataformas. Soporta todos los niveles de las especificaciones ODBC y es compatible con la gramática extendida de SQL especificada por ODBC.

La utilidad Dharma/SQL Network ofrece apertura para sistemas cliente/servidor y de tres niveles (en inglés *three-tier -Internet/intranet-*), al manejar los paquetes de datos en la red y la conversión entre formatos. Ofrece también dispositivos de compuerta (*Dharma/SQL Gateways*) para que los ambientes Oracle (*Open Client*) y Sybase (*Sybase server*) puedan obtener acceso a bases de datos propietarias.

SQL Access utiliza buffers y caches extensivamente para maximizar el rendimiento en transacciones y consultas, incluyendo sentencias SQL precompiladas, información de metadata y de seguridad. Provee optimizaciones para la ejecución de consultas, como la construcción de índices para ambientes donde no es posible definirlos, con el objetivo de agilizar las consultas. Permite además desarrollar aplicaciones basadas en el servidor, en ambientes que incluyen SQL embebido, SQL interactivo y procedimientos almacenados, interfaces que permiten crear herramientas administrativas personalizadas y aplicaciones de tres niveles, así como utilidades para la carga y extracción de datos.

Dharma/SQL presenta características que facilitan la administración de los sistemas, como una interfaz de almacenamiento, que permite obtener propiedades de los índices que se utilizan, métodos de navegación en las bases de datos, costo de acceso a los datos, estadísticas de los datos manipulados por el sistema de almacenamiento (para el cálculo del plan de ejecución óptimo), entre otras.

Las interfaces de red de Dharma/SQL se fundamentan en la filosofía de RPC, en la que la solicitud de un cliente es enviada a un sistema remoto para ejecutar un procedimiento designado, utilizando los parámetros que se envían, y los resultados son enviados de vuelta a quien realizó la llamada. Una red Dharma/SQL consiste en una compilador de lenguaje de definición de interfaz de red, una librería de funciones de red en tiempo de corrida, y un conjunto de dispositivos para la capa de transporte (refiérase al modelo OSI de comunicación).

El compilador de red de Dharma/SQL recibe como entrada una defición de solicitudes API y produce como salida un conjunto de mensajes RPC para el cliente y el servidor, los que convierten los parámetros de llamada a los procedimientos en un paquete con un formato que puede ser enviado por la red por una librería de tiempo de ejecución, y viceversa.

La librería de tiempo de ejecución de Dharma/SQL Network es invocada por el código generado por el compilador, y contiene funciones para hacer llamadas a procedimientos remotos, recibir solicitudes de procedimientos remotos y para la conversión de formatos. Utiliza el formato estándar de representación externa de datos (XDR[23]) para la conversión de paquetes, lo cual le permite ser portable a diferentes plataformas.

---

[23] El protocolo XDR (en inglés representa *external data representation*) fue definido por Sun Microsystems, y consiste en definir una representación externa de los objetos (que sea estándar e independiente de la estructura física de los datos, y por consecuencia, intercambiable entre máquinas de diferentes tipos.

Esta librería permite el soporte para diferentes procesos de bases de datos y arquitecturas de enlaces. Las llamadas de RPC pueden ser ejecutadas en ambiente orientado a enlaces o a procesos.

Los dispositivos de transporte de Dharma/SQL Network se requieren para servir de interfaz a un protocolo específico de transporte de red y existe un dispositivo para cada protocolo soportado.

La opción distribuida del Dharma/SQL crea una base de datos virtual que contiene datos de bases de datos geográficamente distribuidas, que esconde las diferencias en la forma de acceso entre los datos, y permite a los usuarios (y las aplicaciones) conectarse a única base de datos para obtener acceso a cualquier datos en el sistema, haciendo inclusive combinaciones entre los datos de varias fuentes de datos, permitiendo crear vistas distribuidas para combinar o mezclar cualquier conjunto de bases de datos distribuidas, y facilita la transferencia de información entre sistemas, a través de operaciones de inserción, actualización y eliminación remotas.

El dispositivo ODBC de Dharma/SQL soporta la gramática extendida SQL, que incluyen queries anidados, vistas recursivas, expresiones "case", joins externos, operaciones de unión, intersección y diferencia de conjuntos, tipos de datos extendidos y llamadas a procedimientos, así como todas las funciones del API ODBC. Puede ser configurado para ambientes cliente/servidor y de Internet/intranet (de tres niveles) y puede ser ejecutado en plataformas UNIX y Windows.

6.3 OLE DB

El elemento de integración de datos de Microsoft, OLE DB es un conjunto de interfaces OLE que permite a las aplicaciones tener acceso uniforme a datos almacenados en diferentes fuentes (DBMS -bases de datos como DB2, Oracle, SQL Server, Access, Paradox-, y no DBMS - como información en sistemas de archivos de Windows NT y UNIX, archivos secuenciales indexados, archivos de correo electrónico, hojas electrónicas, herramientas de manejo de proyecto y otras-).

OLE DB está basado en un sistema de administración de bases de datos por Componentes, que tiene como ventajas el permitir construir sistemas utilizando únicamente la funcionalidad necesaria para el usuario final (sin la necesidad de adquirir un paquete completo de productos de administración de bases de datos -DBMS-), además, abre los sistemas de almacenamiento de datos que no se clasifican como DBMS para que puedan ser utilizados por API populares de acceso a datos como ODBC, sin la necesidad de construir interfaces SQL para los datos, pues el único requerimiento es construir una interfaz que permita ver los datos como tablas (en forma tabular).

OLE DB define un conjunto abierto y extensible de interfaces que factorizan y encapsulan porciones de funcionalidad de un DBMS, que a su vez definen los límites de los componentes del DBMS, que facilitan el acceso uniforme y transaccional a diferentes fuentes de datos. Así, un DBMS se convierte en un conglomerado de componentes que cooperan para consumir y producir datos a través de un conjunto uniforme de interfaces.

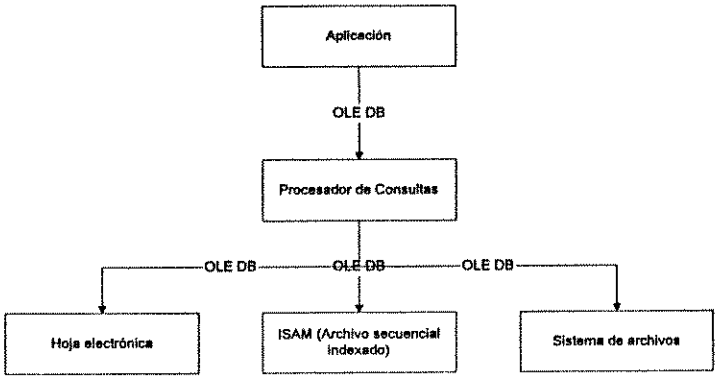


Figura 32: Arquitectura de un sistema que utiliza OLE DB

Las áreas funcionales de OLE DB incluyen acceso y actualización de datos, procesamiento de consultas, información de catálogo, avisos, transacciones, seguridad y acceso remoto a los datos.

OLE DB utiliza la infraestructura del modelo COM (objetos componentes), que reduce la duplicación innecesaria de servicios y provee un alto grado de interoperabilidad entre diversas fuentes de datos, ambientes de programación y herramientas.

En la terminología OLE DB, un consumidor es una pieza de un sistema (o código de aplicación) que consume una interfaz OLE DB, esto incluye los mismos componentes OLE DB. Un proveedor es cualquier componente de software que expone una interfaz OLE DB, y pueden clasificarse en proveedores de datos (cualquiera que posee datos y los expone en forma tabular: DMBS, administradores de almacenamiento, archivos secuenciales indexados) y proveedores de servicios (cualquier componentes OLE DB que no posee datos propios, pero encapsula ciertos servicios produciendo y consumiendo datos a través de interfaces OLE DB, como un procesador de queries heterogéneos).

OLE DB define los siguientes componentes, cada uno de los cuales es un objeto OLE COM:

1. **Enumeradores:** los enumeradores buscan fuentes de datos disponibles y otros enumeradores. Los consumidores que no tienen una forma natural de acceso a determinada fuente de datos utilizan los enumeradores para buscar una fuente de datos a utilizar. Un enumerador expone la interfaz *ISourcesRowset* que retorna un conjunto de filas que describen todas las fuentes de datos y los enumeradores visibles a él.
2. **Objetos Fuente de Datos:** contienen lo necesario para hacer conexiones a una fuente de datos (un archivo o un DBMS) y crean sesiones.
3. **Sesiones:** proveen un contexto para las transacciones y pueden ser implícitas o explícitas. Un objeto Fuente de Datos puede crear múltiples sesiones. Las sesiones crean transacciones, comandos y conjuntos de filas.
4. **Transacciones:** los objetos transacción son utilizados cuando se finalizan o se abortan transacciones (las cuales pueden ser locales o distribuidas).

5. **Comandos:** los comandos ejecutan una instrucción (en forma de texto), como una sentencia SQL. Si éste especifica un conjunto de filas (como un SELECT de SQL), este comando crea el conjunto de filas. Una sesión puede crear múltiples comandos.
6. **Conjunto de filas:** presentan datos en forma tabular. Un caso especial de un conjunto de filas es un índice. Pueden crearse desde una sesión o desde un comando.
7. **Errores:** pueden ser creados por cualquier interfaz en cualquier objeto OLE DB. Contienen información adicional acerca de un error.

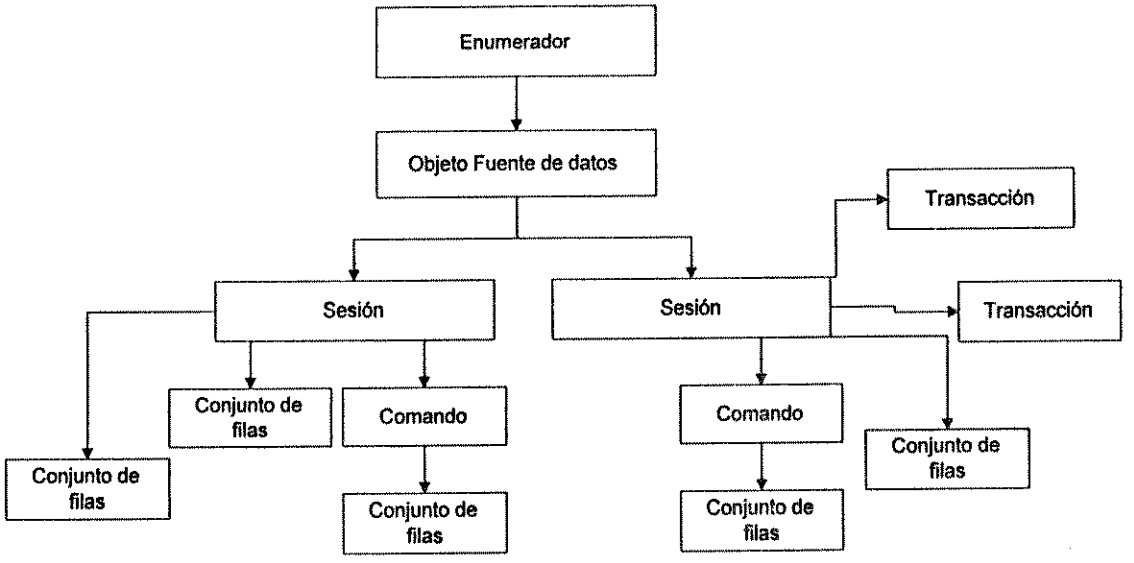


Figura 33: Modelo de conexión OLE DB

El modelo de conexión de OLE DB define la forma cómo los proveedores de datos y servicios se localizan y se activan, y está representado en los objetos Fuente de Datos y Sesión. Para poder tener acceso a un proveedor OLE DB, un consumidor debe instanciar el objeto fuente de datos, cada proveedor de datos se identifica por un identificador único de clase (CLSID) en el registro del sistema operativo y es instanciado. El objeto fuente de datos expone las propiedades que el consumidor utiliza para proveer información básica de autenticación (como usuario y clave de acceso) así como el nombre de la fuente de datos (archivo o base de datos).

Cuando el objeto fuente de datos ha sido inicializado, el consumidor puede hacer llamadas a métodos para consultar las capacidades de proveedor, como interfaces, propiedades de conjuntos de filas y los dialectos de SQL que el proveedor soporta. Además, puede crear sesiones. Las sesiones funcionan como un constructor de conjuntos de filas, comandos y transacciones.

Los conjuntos de filas son el objeto central que permite a todos los proveedores de datos OLE DB presentar los datos en forma tabular. Conceptualmente, un conjunto de filas es un conjunto de tuplas de datos, en la que cada una tiene columnas de datos. El objeto de conjunto de filas más simple tiene cuatro interfaces: *IRowset*, que contiene métodos para

buscar filas en forma secuencial; *IAccessor*, que permite la definición de enlaces de grupos de columnas que describen la forma en que los datos tabulares pueden ligarse a variables de un programa consumidor; *IColumnsInfo*, que provee información sobre las columnas del conjunto de filas; y *IRowsetInfo*, que provee información acerca del conjunto de filas mismo. Otras interfaces de conjuntos de filas ofrecen capacidades adicionales, como inserción, eliminación y actualización de filas, acceso directo, etc. Los conjuntos de filas pueden ser creados de dos formas: como resultado de un query, o directamente como resultado de una llamada directa a una interfaz de apertura.

En OLE DB, las sentencias del lenguaje de definición de datos (DDL) y del lenguaje de manipulación del los datos (DML) se conocen como texto de comando (típicamente expresado en ANSI SQL 92). Un objeto comando contiene un texto de comando y encapsula los servicios de procesamiento de consultas disponibles en un DBMS, y tienen varias interfaces que representan áreas de funcionalidad de un procesador de consultas: formulación, preparación y ejecución de la consulta. El propósito principal de un objeto Comando es ejecutar un texto de comandos, lo que puede requerir la creación de un conjunto de filas.

Un objeto consumidor OLE DB que desea utilizar un comando, debe seguir los siguientes pasos:

1. Obtener una interfaz en el objeto Comando.
2. Construir una cadena de caracteres que representa el texto del comando.
3. Pasar la cadena de caracteres al comando.
4. Indicar las propiedades del conjunto de filas resultante (y las interfaces que tendrá).
5. Ejecutar el comando, y si la instrucción lo especificaba, crear el conjunto de filas resultante.

Dado que los proveedores pueden consumir y producir conjuntos de filas, es posible crear procesadores de consultas para ejecutar queries distribuidos, heterogéneos o paralelos, así como procesadores especializados como procesadores de SQL, de búsqueda de texto, procesadores de consultas de imágenes y/o geográficos.

Los consumidores OLE DB pueden obtener información de catálogo a través de la interfaz *IDBSchemaRowset*, que permite solicitar información de tipos de datos, tablas, columnas, índices, vistas, restricciones, estadísticas, conjuntos de caracteres y dominios. Toda la información de catálogo se retorna al consumidor en forma de conjuntos de filas, cuyos esquemas están basados en el estándar ANSI SQL 92.

OLE DB utiliza un modelo de control de notificaciones OLE para el paso de mensajes entre componentes OLE DB y consumidores. Las notificaciones definen un mecanismo básico en el cual se implementa el comportamiento de la fuente de datos que está siendo utilizada.

El acceso remoto a los datos no está totalmente soportado por la versión 1.0 (disponible en 1997) del OLE DB.

CONCLUSIONES

1. Un sistema de bases de datos heterogéneas es un tipo especial de un sistema de múltiples bases de datos, y la heterogeneidad del mismo puede variar desde el nivel de conceptual de la aplicación (con qué objetivo fue diseñada), nivel de abstracción específico, nivel del esquema conceptual (especificación de la metadata), formato de los datos y herramienta de manipulación (DBMS).

2. El objetivo principal de la integración de sistemas de bases de datos es brindar al usuario final un ambiente totalmente transparente, ocultando aspectos de distribución, localidad, replicación, herramientas utilizadas, sin sacrificar la seguridad, la eficiencia y la integridad de la información, asegurando la interoperabilidad entre los componentes.

3. El problema de integración de sistemas de bases de datos heterogéneos es parte del problema general que nació con la evolución de la tecnología de información hacia los sistemas abiertos, y tiene un alto grado de relación con los sistemas operativos, las redes y los protocolos de comunicación.

4. El diseño de la integración de sistemas de bases de datos, y en especial, la selección de las herramientas de integración, depende principalmente de las características de la aplicación que la requiere: puede ser una aplicación de consulta que no necesita control de transacciones fuera de línea (en el caso de un *datawarehouse*), una aplicación de consulta en línea, o bien un sistema de actualización a múltiples sistemas totalmente transaccional y en línea, por lo que al iniciar un proyecto de integración es necesario definir el nivel de integración deseado, analizar cada componente a integrar y evaluar los requerimientos respecto a tiempo de respuesta, infraestructura disponible y disponibilidad de los datos.

5. Un mediador es un componente de software que forma una capa de abstracción entre dos componentes de un sistema y se encarga de interpretar la semántica individual de cada uno. La necesidad los mediadores se deriva de que es prácticamente imposible uniformar todos los sistemas de bases de datos bajo un modelo semántico único (por razones de mercadeo y por que diferentes dominios necesitan diferentes modelos) y que lo más usual es que todos los dominios cambiarán con el tiempo.

6. La integración de sistemas de bases de datos requiere de componentes tanto a nivel de hardware (debe existir un medio de comunicación, sea éste un cable de red local, una conexión vía telefónica o vía satélite) como de software.

7. El concepto de mediación se pone en práctica en el Middleware, que es una infraestructura de aplicaciones distribuidas, constituido por conjuntos de software distribuido que permite a elementos de las aplicaciones interoperar a través de enlaces de red y se refiere un conjunto de productos que incluyen software de red (como drivers TCP/IP), software de red específico de las bases de datos y drivers de ODBC.

RECOMENDACIONES

1. La integración de sistemas heterogéneos es un tema con bastante empuje en la actualidad y se ha desarrollado en múltiples áreas y el ingeniero en ciencias y sistemas debe, en su labor de integrador de sistemas, debe conocer los problemas fundamentales con los que se enfrentará en su labor, por lo que se recomienda introducir en los cursos de sistemas operativos, telecomunicaciones y redes y bases de datos, los temas de integración de sistemas (enfatizando en el área que corresponde a cada curso).
2. Como pudo observarse durante el desarrollo de esta tesis, la integración de bases de datos heterogéneas puede realizarse de varias formas, cada una con sus ventajas y desventajas, por lo que se recomienda desarrollar nuevas investigaciones para ampliar la información acerca de cada uno de estos métodos de integración y el desarrollo de una metodología para la evaluación de integración de sistemas de bases de datos heterogéneas.
3. A los profesionales en sistemas de bases de datos, se recomienda que al evaluar sistemas manejadores de bases de datos para implantación de soluciones, seleccionen aquellas herramientas que soporten los estándares propuestos por ANSI y OSI, pues éstas son las más aptas para pertenecer a un sistema integrado posteriormente. Una herramienta poco estándar será más difícil de integrar. Asimismo, que al diseñar sistemas de bases de datos, se analice la organización como un todo, y no solamente un área funcional, para evitar al máximo la heterogeneidad semántica.
4. A las empresas guatemaltecas, y en general, a todos los profesionales de la informática, se recomienda considerar como una posible solución a sus necesidades en sistemas de bases de datos, la integración de sus sistemas actuales (sean éstos homogéneos o heterogéneos), pues ésto favorece el abaratamiento de las soluciones y la agilización de los resultados.

BIBLIOGRAFÍA

Breitbart, Y. "Multidatabase Interoperability", **Sigmod Record**, Vol. 19, Núm. 3, 1990, pp. 53-59.

Bright, M.W. et. al. "A Taxonomy and Current Issues in Multidatabase Systems", **Computer**, Vol. 25, Núm. 3, 1992, pp. 50-59.

Bright, M.W. et. al. "Automated Resolution of Semantic Heterogeneity in Multidatabases", **ACM Transactions on Database Systems**, Vol. 19. Núm. 2, 1994. Pág. 212-252.

Brockschmidt, Kraig. **What OLE Is Really About**. Estados Unidos: Microsoft Corporation, 1996, 40 pp.

Calton Pu, et. al. "Heterogeneuos and Autonomous Transaction Processing", **Computer**, Vol. 24, Núm. 12, 1991, pp. 64-72.

Database Architecture Framework Task Group for the ANSI/X3/Sparc Database System Study Group. "Reference Model for DBMS Standarization", **Sigmod Record**, Vol. 15, Núm. 1, 1986, pp. 19-58.

Date, C.J. **An Introduction to Database Systems, Volume I**. Estados Unidos: Addison-Wesley Publishing Company, 1990, 860 pp.

Dodge, Marc. "Open Systems? Not Yet!", **Open Computing**, Vol. 11, Núm. 1, 1994, p. 35.

Hamilton, Dave, et.al. **Programming Windows NT 4.** Estados Unidos: SAMS Publishing, 1996, pp. 673-675.

Igras, Eugene. "A Framework for query processing in a federated database system: A Case Study." **http://wwwsgi.usrus.maine.edu/gisweb/spatdb/urisa/ur94016.html.**

Kador, John. "The Internet and the World Wide Web may be all the middleware you need", **DBMS,** Diciembre 1996, pp. 30-32.

Keuffel, Warren. "Middleware on the Web", **Internet Systems,** octubre 1996, p. 25-27.

Kim, Won. et. al. "Classifying Schematic and Data Heterogeneity in Multidatabase Systems", **COMPUTER**. Vol. 24, Núm. 12, 1991, pp. 12-17.

Middleware Working Party. **Delivering Data to the Desktop. A guide to ODBC Drivers.** Estados Unidos: Universities and Colleges Software Group. 1995.

Mullender, Sape. **Distributed Systems.** Estados Unidos: ACM Press Addison -Wesley Publishing Company, 1989, pp. 7 - 8.

Plederender, Richard. et. al. "DB Integrator: Open Middleware for Data Access." **http://www.digital.com**

Ram, Sudha. "Heterogeneous Distributed Database Systems", **COMPUTER,** Enero 1991, pp. 7-9.

Rifflet, Jean-Marie. **Comunicaciones en Unix.** España: McGraw-Hill, 1992. pp. 279-281.

Sciore, Edward. et.al. "Using Semantic Values to Facilitate Interoperability among Heterogeneus Information Systems". **ACM Transactions on Database Systems.** Vol. 19, Núm. 2, 1994, pp. 254-290.

Signore, Robert, et.al. **The ODBC Solution. Open database conectivity in distributed environments.** Estados Unidos: McGraw Hill Series on Computers, 1995. 650 pp.

Silberschatz, A. et.al. **Sistemas Operativos, Conceptos fundamentales.** Estados Unidos: Editorial Addison Wesley, 1994. 727 pp.

Tanembaum, Andrew S. **Redes de Ordenadores.** México: Editorial Prentice Hall, 1991, pp. 14-16.

Wiederhold, Gio. "Mediators in the Architecture of Future Information Systems", **COMPUTER.** Marzo, 1992. pp. 38-49