INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS MONTERREY

PROGRAMA DE GRADUADOS EN ELECTRONICA, COMPUTACION, INFORMACION Y COMUNICACIONES

ESPECIFICACION DE UNA ARQUITECTURA PARA DBMS DISTRIBUIDO BASADA EN COMPONENTES

TESIS

PRESENTADA COMO REQUISITO PARCIAL PARA OBTENER EL GRADO ACADEMICO DE:

MAESTRIA EN CIENCIAS EN TECNOLOGIA INFORMATICA

POR

ROGER HUMBERTO CASTILLO ESPINOLA

MONTERREY, NUEVO LEON; DICIEMBRE DE 2002

INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS MONTERREY
PROGRAMA DE GRADUADOS EN ELECTRONICA, COMPUTACION, INFORMACION Y COMUNICACIONES

ESPECIFICACION DE UNA ARQUITECTURA PARA DBMS DISTRIBUIDO BASADA EN COMPONENTES

TESIS

PRESENTADA COMO REQUISITO PARCIAL PARA OBTENER EL GRADO ACADEMICO DE:

MAESTRIA EN CIENCIAS EN TECNOLOGIA INFORMATICA

POR

ROGER HUMBERTO CASTILLO ESPINOLA

MONTERREY, NUEVO LEON; DICIEMBRE DE 2002

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS MONTERREY

PROGRAMA DE GRADUADOS EN ELECTRÓNICA, COMPUTACIÓN, INFORMACIÓN Y COMUNICACIONES

ESPECIFICACION DE UNA ARQUITECTURA PARA DBMS DISTRIBUIDO BASADA EN COMPONENTES

TESIS

PRESENTADA COMO REQUISITO PARCIAL PARA OBTENER EL GRADO ACADEMICO DE:

MAESTRÍA EN CIENCIAS EN TECNOLOGÍA INFORMÁTICA

POR:

ROGER HUMBERTO CASTILLO ESPÍNOLA

MONTERREY, NUEVO LEÓN; NOVIEMBRE DE 2002

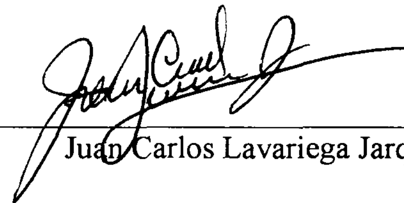
INSTITUTO TECNOLÓGICO DE ESTUDIOS SUPERIORES DE MONTERREY

DIVISIÓN DE ELECTRÓNICA, COMPUTACIÓN, INFORMACIÓN Y COMUNICACIONES

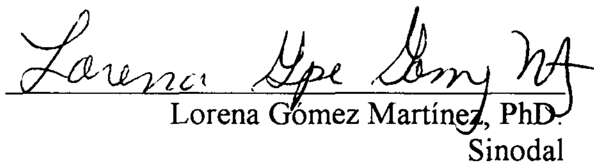
PROGRAMAS DE GRADUADOS EN ELECTRÓNICA, COMPUTACIÓN, INFORMACIÓN Y COMUNICACIONES

Los miembros del comité de tesis recomendamos que la presente tesis del Licenciado en Ciencias de la Computación Roger Humberto Castillo Espínola sea aceptada como requisito parcial para obtener el grado académico de Maestro en Ciencias en Tecnología Informática.

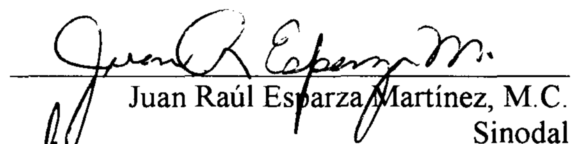
Comité de tesis:



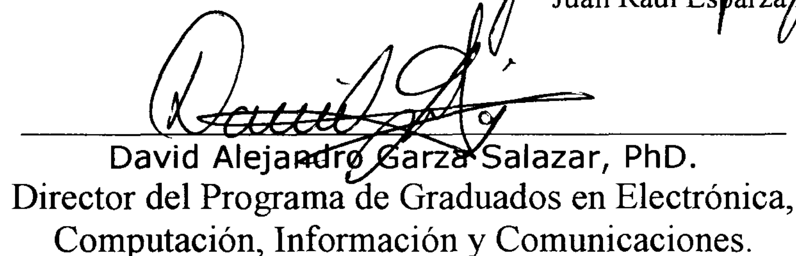
Juan Carlos Lavariega Jarquín, PhD.
Asesor



Lorena Gómez Martínez, PhD.
Sinodal



Juan Raúl Esparza Martínez, M.C.
Sinodal



David Alejandro Garza Salazar, PhD.
Director del Programa de Graduados en Electrónica, Computación, Información y Comunicaciones.

Noviembre de 2002

ESPECIFICACIÓN DE UNA ARQUITECTURA PARA DBMS DISTRIBUIDO BASADA EN COMPONENTES

POR:

ROGER HUMBERTO CASTILLO ESPÍNOLA

TESIS

Presentada al Programa de Graduados en Electrónica, Computación, Información y Comunicaciones.

Este trabajo es requisito parcial para obtener el grado de Maestro en Ciencias en Tecnología Informática

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

NOVIEMBRE DE 2002

Dedicatoria

A Dios, para ti Señor.

Dame Señor la fortaleza del viejo roble,
para que ningún triunfo me envanezca.
La alegría de la naturaleza,
para que ninguna soledad me abata.
La libertad del ave,
para elegir mi camino.
Y la voluntad de un expedicionario,
para seguir siempre adelante y servir.
Así sea.

Agradecimientos

A ti Señor, una vez más, por todo y por todos los de abajo, Gracias Señor!!.

A Mamá y Papá, que más les puedo decir, los quiero mucho, Gracias por todo, 1,2 y ..

A Esthela, mi pequeña, por todos esos momentos compartidos, por tu cálida sonrisa, por tu apoyo constante durante el desarrollo de este trabajo, y particularmente por el apoyo en esos momentos decisivos que marcaron la diferencia. Te amo.

A mi hermana Addy, por sus recetas integrales, a mi hermanita Heidi, por sus palabras de motivación, a mi sobrinita Vale, por sus sonrisas, a toda mi familia, por su apoyo y cariño a través de este camino.

A la familia Flores Suárez, por su confianza y cariño y por supuesto, por esas cenas tan ricas =).

A mis amigos Francisco, Gimer, Iván, Juan Carlos, Marina, Max, Sergio, Vladimir por la ayuda y convivencia en esta aventura que empezamos juntos.

A mis sinodales Dra. Lorena Gómez Martínez y Juan Raúl Esparza Martínez MC, por su paciencia, dedicación y apoyo durante la elaboración de esta tesis.

A mi asesor, Dr. Juan Carlos Lavariega Jarquín, por sus revisiones, acertados consejos y comentarios de gran valor a través del desarrollo de este trabajo.

Al Sistema Coopera por apoyarme y respaldarme a lo largo de este proyecto.

A todas las personas que de una forma u otra intervinieron en esta etapa de mi vida.

Gracias...

Resumen

Las arquitecturas de las bases de datos distribuidas han sido motivo de estudio e investigación aunque de poca utilización debido a que los DBMS se han enfocado a la arquitectura centralizada o manejan conceptos de distribución a su manera.

A lo largo de este trabajo se describen los distintos tipos de bases de datos distribuidas y se analizan diversas arquitecturas de manejadores de bases de datos con el fin de proponer una arquitectura para un manejador de base de datos distribuida basada en componentes proporcionando una descripción de los módulos de la misma, el flujo de la información entre ellos y diferentes escenarios prácticos de uso común en esta arquitectura.

Mediante esta arquitectura se pretende contribuir en el área de las bases de datos distribuidas proporcionando un modelo de DBMS distribuido configurable que permita la adaptación o eliminación de componentes, y sirva como marco de referencia para la implementación de sistemas de base de datos distribuidos.

La estructura de la arquitectura propuesta en esta tesis se encuentra basada en componentes de software, los cuales proporcionan las características de extensibilidad, adaptabilidad, reutilización, portabilidad y distribución de datos además de encapsular la funcionalidad del manejador. La definición del manejador mediante los componentes permite la configuración y adaptación de módulos según los requerimientos para la administración de la información. Se desarrollan ejemplos de estos componentes para comprender mejor el desempeño de la arquitectura del DBMS utilizando el lenguaje java.

Al final se describen algunos trabajos de investigación que pueden ser derivados de este trabajo y un ejemplo de la utilización de esta propuesta.

Contenido

DEDICATORIA .... ***IV***

AGRADECIMIENTOS .... ***V***

RESUMEN .... ***VI***

LISTA DE FIGURAS .... ***IX***

LISTA DE TABLAS. .... ***X***

CAPITULO 1 INTRODUCCIÓN ***1***

1.1 ANTECEDENTES 1
1.2 OBJETIVO DE LA TESIS 2
1.3 ALCANCE Y LIMITACIONES 2
1.4 ORGANIZACIÓN DE LA TESIS 2

CAPITULO 2 CONCEPTOS DE ARQUITECTURAS DE BASES DE DATOS DISTRIBUIDAS ***4***

2.1 CONCEPTOS BÁSICOS 4
2.2 SISTEMAS MANEJADORES DE BASES DE DATOS DISTRIBUIDOS HOMOGÉNEOS 5
2.3 SISTEMAS MANEJADORES DE BASES DE DATOS DISTRIBUIDOS HETEROGÉNEOS 7
2.4 EL CONCEPTO DE DBMS COMPONENTE: CDBMS 8
2.4.1 Fundamentos de CDBMS: ComponentWare 9
2.4.2 DBMS Orientado a Objetos (OODBMS) 9
2.4.3 DBMS Relacional (RDBMS) 9
2.5 MODELOS DE BASE DE DATOS COMPONENTES 10
2.6 OBJECT RELATIONAL DBMS (ORDBMS) COMO UN CDBMS 11
2.6.1 El Modelo de datos abstracto de ORDBMS *11*
2.7 ORDBM DISTRIBUIDO 12
2.7.1 Extensiones Distribuidas para SQL *13*

CAPITULO 3 ANÁLISIS DE ARQUITECTURAS ***15***

3.1 ARQUITECTURA ORACLE 15
3.2 ARQUITECTURA DB2 19
3.3 ARQUITECTURA MYSQL 23
3.4 ARQUITECTURA POSTGRESQL 25
3.5 ARQUITECTURA DE INNODB 29
3.6 CONCLUSIONES 33

CAPITULO 4 PROPUESTA DE ARQUITECTURA DISTRIBUIDA. ***34***

4.1 INTRODUCCIÓN 34
4.2 ARQUITECTURA DEL SISTEMA GLOBAL 34
4.3 ARQUITECTURA PROPUESTA 35
4.4 MÉTODOS GENERALES DE LA ARQUITECTURA PROPUESTA 39
4.5 TABLAS DE REFERENCIA PARA EL DISTRIBUTION CONTROL CENTER (DCC) 50
4.6 ADAPTABILIDAD Y PORTABILIDAD DE LOS MÓDULOS 52
4.6.1 Extensiones en DBMS´s *53*
4.7 DISTRIBUCIÓN DE LOS COMPONENTES 54
4.8 CONCLUSIONES 55

CAPITULO 5 VALIDACIÓN DE LA ARQUITECTURA MEDIANTE CASOS DE ESTUDIO. ***56***

5.1 ESCENARIO 1 CONSULTA DE UN REGISTRO A LA BASE DE DATOS (READ-ONLY) 56
5.2 ESCENARIO 2 ESCRITURA DE UN REGISTRO A LA BASE DE DATOS (INSERT) 60

5.3 Escenario 3 Modificación de un registro a la base de datos (Update)....63
5.4 Escenario 4 Eliminación de un registro en la base de datos (Delete)....67
5.5 Escenario 5 Respaldo y recuperación de una base de datos (Backup and Recovery)....71
5.6 Conclusiones....74

CAPITULO 6 CONCLUSIONES....75

6.1 Conclusiones globales....75
6.2 Trabajo para futuras investigaciones....76

APÉNDICE....77

REFERENCIAS BIBLIOGRÁFICAS....89

VITA....91

Lista de Figuras

Figura 2.1 Arquitectura global de un DDBMS homogéneo 6
Figura 2.2 Arquitectura del esquema de un DDBMS homogéneo 7
Figura 2.3 Dos componentes *plug-in* agregándole funcionalidad al DBMS 10
Figura 2.4 Un DBMS Configurable 11
Figura 2.5 Topología de un ORDBMS Distribuido 12
Figura 3.1 Arquitectura interna de Oracle 16
Figura 3.2 Instancia base de datos DB2 20
Figura 3.3 Vista general de la arquitectura del manejador MySQL 23
Figura 3.4 Arquitectura del ORDBMS postgreSQL. 25
Figura 3.5 Arquitectura del DBMS InnoDB 30
Figura 4.1 Arquitectura global del sistema MySQL 34
Figura 4.2 Distribution control center 35
Figura 4.3 Transaction control 35
Figura 4.4 Buffer manager 35
Figura 4.5 Arquitectura propuesta de un manejador de base de datos distribuidas 38
Figura 4.6 Estructura del componente Query Engine 39
Figura 4.7 Estructura del componente Distribution Control Center. 43
Figura 4.8 Estructura del componente Buffer Manager. 44
Figura 4.9 Estructura del componente Transaction Control. 46
Figura 4.10 Estructura del componente Recovery Manager 47
Figura 4.11 Estructura del componente Storage Manager Control. 49
Figura 4.12 Comunicación entre componentes en un ambiente distribuido 54
Figura 5.1 Escenario de consulta de un registro mediante un DDBMS 58
Figura 5.2 Escenario de inserción de un registro mediante un DDBMS 61
Figura 5.3 Escenario de modificación de un registro mediante un DDBMS 65
Figura 5.4 Escenario de eliminación de un registro mediante un DDBMS 69
Figura 5.5 Escenario de respaldo y recuperación de un registro mediante un DDBMS 72

Lista de tablas.

TABLA 4.1 ATRIBUTOS DE LA TABLA DE FRAGMENTACIÓN 50
TABLA 4.2 ATRIBUTOS DE LA TABLA DE LOCALIZACIÓN 51
TABLA 4.3 EJEMPLO DE DATOS EN LA TABLA DE FRAGMENTACIÓN 51
TABLA 4.4 EJEMPLO DE DATOS EN LA TABLA DE LOCALIZACIÓN 51

Capitulo 1 Introducción.

1.1 Antecedentes.

La tecnología de base de datos distribuidas agrupa dos conceptos divergentes, la integración a través del elemento base de datos y la distribución a través del elemento red.

Para manejar la información surgieron los sistemas manejadores de bases de datos al final de los años sesentas y ahora se encuentran suficientemente maduros, tanto que se puede decir que manejan todos los sistemas de información actuales.

El propósito de un Sistema Manejador de Bases de Datos (DBMS) es el de proveer poderosas herramientas para manejar una base de datos [DBELL].

Entre las funciones que debe contener se encuentran:

- Mantenimiento de estructuras físicas de datos.
- Lenguajes de recuperación y almacenamiento de datos.
- Facilidades para asegurar la integridad y seguridad de los datos.
- Soporte multiusuario.
- Métricas de tolerancia a fallas.

Una base de datos distribuida puede ser definida como una colección de datos compartidos lógicamente integrada, donde los datos se encuentran físicamente distribuidos a través de nodos de una red de computadoras [TOZS].

Y de aquí tenemos que un sistema manejador de bases de datos distribuidas es el software que maneja una base de datos distribuida de tal manera que los aspectos de distribución son transparentes para el usuario [DBELL].

Existen DBMS que han sido diseñados tanto para sistemas comerciales como open- source, entendiendo por open-source (fuentes abiertas), que el uso del sistema no requiere el pago de ningún tipo de licencia a ninguna casa desarrolladora de software, y que viene acompañado del *código fuente* para futuras mejoras o implementaciones por parte del usuario que lo utiliza [RSTAL], ambos enfocados mas bien a la centralización de las bases de datos o a ciertas formas de "acceso distribuido".

Actualmente no existe una arquitectura apropiada diseñada para manejadores de bases de datos distribuidas basada en componentes que se

encuentren soportadas tanto para sistemas libres (open source) como sistemas comerciales, por lo tanto, este proyecto tiene como objetivo final definir y especificar una arquitectura estándar para los manejadores de bases de datos distribuidas basada en componentes, que contenga las características de adaptabilidad, extensibilidad, dominio público y confiabilidad y sirva como marco de referencia para la implementación de bases de datos distribuidas.

Con el fin de facilitar el entendimiento de este documento, utilizaremos términos técnicos en inglés, dado que una traducción al español podría causar confusiones en el significado de los mismos.

1.2 Objetivo de la tesis.

El objetivo de esta tesis es definir y elaborar una arquitectura estándar para los manejadores de bases de datos distribuidas basada en componentes que se adapte congruentemente a las características propias de los sistemas libres, como son la adaptabilidad, extensibilidad, herramientas de dominio público y confiabilidad, proporcionando la funcionalidad de distribución de datos de forma transparente.

1.3 Alcance y limitaciones.

- Se utilizará el manejador de bases de datos MySQL como base experimental para la descripción de la arquitectura propuesta y realizaremos un estudio de diversos manejadores, tanto libres como comerciales.
- Se utilizarán bases de datos relacionales.
- Los resultados obtenidos serán aplicables tanto en ambientes abiertos como comerciales, utilizando los conceptos propuestos en este trabajo.

1.4 Organización de la tesis.

Para avanzar congruentemente en el desarrollo de este trabajo, el capítulo uno nos proporciona una introducción al tema de las bases de datos distribuidas, objetivo, alcances y algunos términos que utilizamos a lo largo de este trabajo. En el capítulo dos definimos conceptos importantes relacionados a las arquitecturas de base de datos distribuidas, también explicamos las clasificaciones actuales de las mismas y datos que permiten una mejor comprensión de los temas que se tratan en capítulos posteriores. El capítulo tres comprende los análisis de las arquitecturas de los manejadores de base de datos actuales, los cuales nos sirven de referencia para proponer una arquitectura propia para DDB. Ya con las bases proporcionadas en los capítulos uno al tres, en el capítulo cuatro se desarrolla una propuesta de arquitectura para soportar una base de datos distribuida, utilizando el manejador de base de datos MySQL, aprovechando las características funcionales implementadas en este manejador. En el capítulo cinco encontraremos las

validaciones a la propuesta del capitulo cuatro, realizando algunas rutinas y escenarios comunes para los manejadores de bases de datos y observaremos el comportamiento conceptual de la arquitectura distribuida. Las conclusiones finales vendrán en el capítulo seis, estableceremos también algunas áreas de investigación futura que se pueden derivar de nuestra investigación.

Capitulo 2 Conceptos de arquitecturas de bases de datos distribuidas.

2.1 Conceptos básicos

El tema de las bases de datos abarca muchos conceptos y términos técnicos, en esta sección se definen algunos de ellos, los cuales nos proporcionan una base de información necesaria para entender el contenido de esta tesis.

Base de datos distribuida (DDB): Colección de múltiples bases de datos lógicamente relacionadas y distribuidas en una red [TOZS].

Manejador de base de datos distribuidas (DDBMS): Software que permite la administración de DDB y oculta la distribución a los usuarios [TOZS].

Sistemas libres (open source): Son aquellos programas que proporcionan la libertad de acceso, modificación, estudio, adaptación, distribución, etc. No solamente se refiere a que no requieren pago de licencias, si no que proporcionan libertad de acción para los usuarios o desarrolladores que los utilizan [RSTAL].

Componentes de Software. Por componente de software se puede entender cualquier objeto de software con existencia independiente. Puede ser un procedimiento, un objeto, una aplicación completa, etc. Un componente puede ser una clase o un conjunto de clases interrelacionadas, y su propósito es servir como bloque de construcción de una aplicación.

Una característica deseable de un componente es que tenga una interfaz o un punto lógico de conexión entre el mismo y el sistema. Una interfaz debe ocultar los detalles de implementación del componente, pero debe proveer información necesaria sobre sus características al sistema. De esta manera un componente puede representar un elemento visual en una interfaz de usuario, una entidad en algún dominio, un manejador de base de datos, etc. [AMOL].

En [KDIT] se menciona que los componentes son cajas negras, lo cual significa que los usuarios pueden usar sus propiedades sin conocer su funcionamiento interno. La interfaz de un componente y su implementación deben ser generadas de forma separada, de tal modo que puedan existir múltiples implementaciones para una interfaz y las implementaciones puedan ser intercambiadas.

De acuerdo con [PVEL], se pueden identificar características en común:

- ***Orientación a la reutilización***, lo que nos hace pensar en algo que se puede tomar y utilizar más de una vez.
- ***Interoperabilidad***, el componente no puede solo usarse de forma aislada, sino que puede integrarse a otras y trabajar conjuntamente.
- ***Función significativa***, el componente tiene una función bien definida, identificable y diferenciable.
- ***Encapsulamiento***, generalmente los componentes ocultan detalles acerca de su implementación. Es importante la manera en que se definen los componentes. Para que estos puedan ser reutilizados, deben de proveer una función específica, así como tener un mecanismo mediante el cual se acceda su funcionalidad, pero que a la vez oculte los detalles de su implementación [AMOL].

Arquitectura: Es la definición de la estructura de un sistema. En ella se encuentran identificados todos los componentes del sistema, funciones de cada componente y las interrelaciones e interacciones entre ellos. La especificación de la arquitectura de un sistema de software requiere de la identificación de los diferentes módulos con sus interfaces e interrelaciones, en términos de los datos y el control de flujo a través del sistema [TOZS].

Veamos ahora la clasificación actual de los manejadores de bases de datos distribuidas.

2.2 Sistemas manejadores de bases de datos distribuidos homogéneos.

Este tipo de manejador tiene múltiples colecciones de datos; integra muchos recursos de datos. La mayoría de los sistemas distribuidos pertenece a esta clasificación. Según [DBELL] los sistemas homogéneos pueden ser divididos en dos partes:

- Autónomos
- No autónomos

El término autónomo indica un claro propósito de los diseñadores de sistemas de dar a los sistemas locales el control de sus propios recursos.

Un DDBMS homogéneo es parecido a un DBMS centralizado, pero en vez de almacenar la información en un mismo sitio, distribuye los datos en los diferentes nodos de una red utilizando el mismo manejador de base de datos, generando ambientes muy parecidos en cada sitio.

La función de distribución de datos es realizada por el módulo representado por la figura 2.1 en el cuadro de sistema distribuido de bases de datos. La figura 2.1 muestra el diagrama de la arquitectura de un DDBMS homogéneo [DBELL].

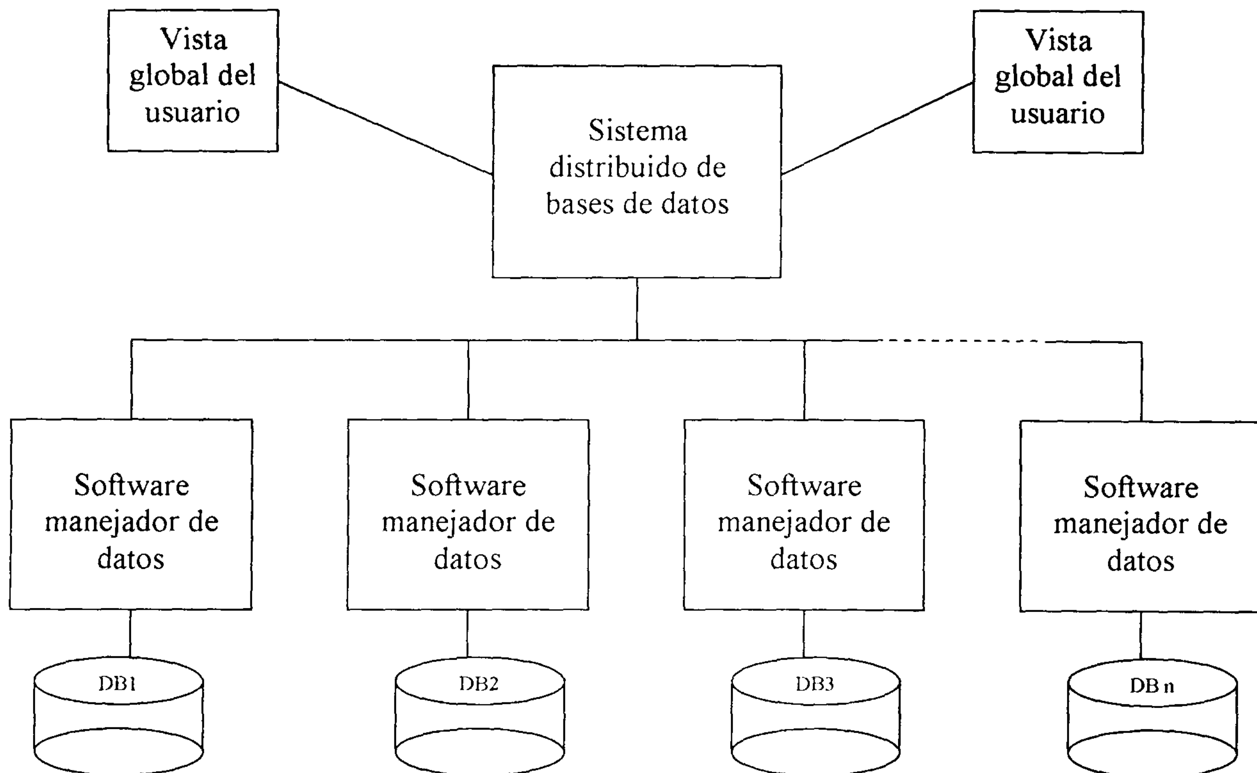


Figura 2.1 Arquitectura global de un DDBMS homogéneo

Para manejar los aspectos de distribución, debemos de agregar dos niveles más a la arquitectura ANSI / SPARC llamados esquemas de fragmentación y localización.

El esquema de fragmentación describe como se dividen las relaciones globales entre las bases de datos locales.

El esquema de localización especifica en que sitio de la red está almacenado cada fragmento de información. [DBELL]. La figura 2.2 muestra un ejemplo de esta arquitectura donde a partir de un esquema global, los usuarios realizan consultas a los diversos sitios mediante la interacción de esquemas de fragmentación y localización, los cuales proporcionan información de los datos almacenados en la BD [DBELL].

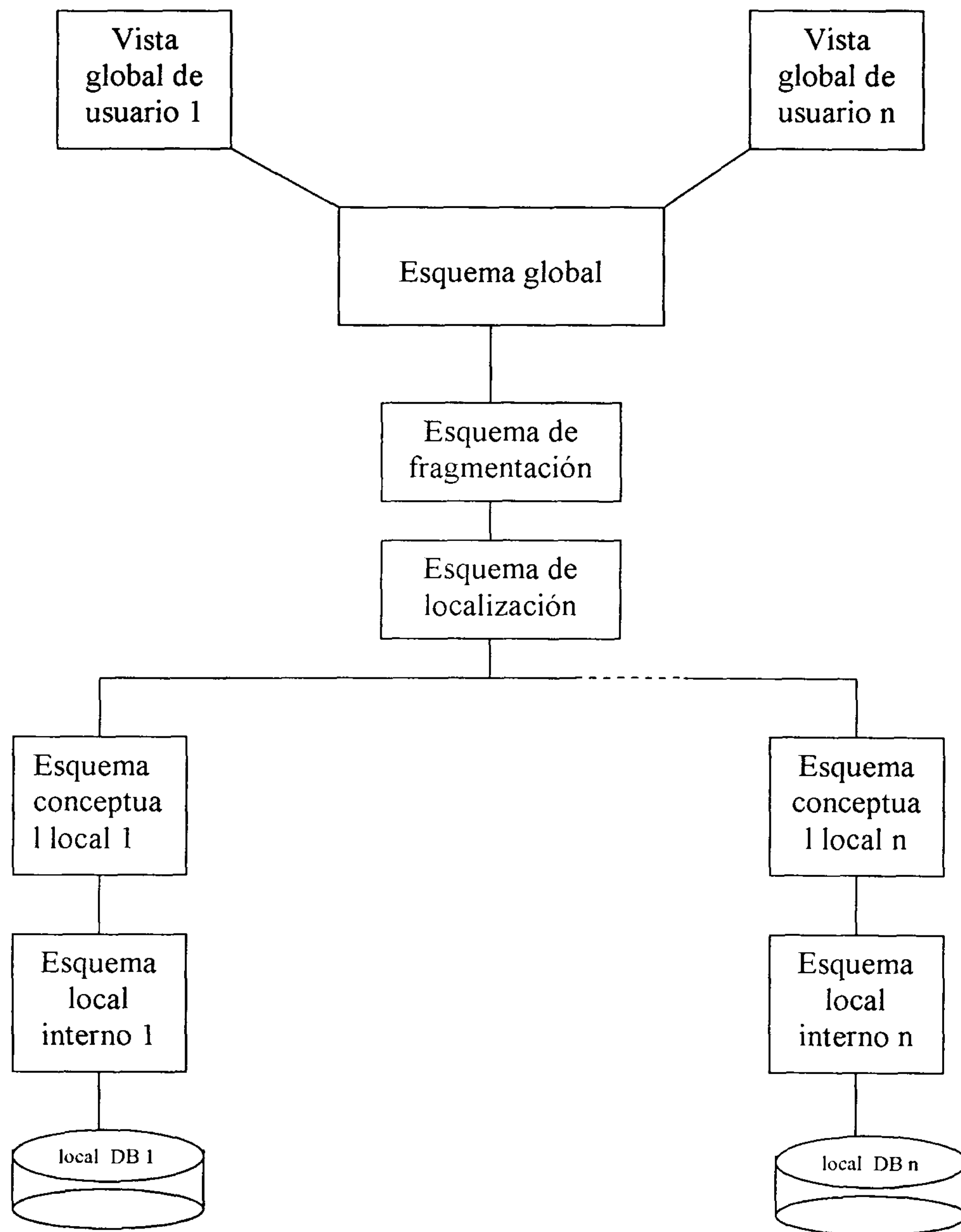


Figura 2.2 Arquitectura del esquema de un DDBMS homogéneo

2.3 Sistemas manejadores de bases de datos distribuidos heterogéneos.

La otra clase de sistemas distribuidos es la heterogénea, que se caracteriza por el uso de diferentes manejadores de bases de datos en los nodos locales.

Existen dos grandes subdivisiones para este tipo de DDBMS, los que realizan su integración completamente a través del sistema y los que realizan su integración vía puentes (gateways). El primer grupo de subdivide a su vez en dos clases, los DBMS con funcionalidad completa, los cuales *proveen un conjunto de*

todas las funciones que se esperan de un DBMS y los DBMS con funcionalidad parcial, que enfatizan los aspectos pragmáticos del manejo colectivo de datos, como conversiones entre sistemas y algunos aspectos de desempeño (Sistemas manejadores multidatabase).

Los sistemas manejadores de bases de datos múltiples (MDBMS) tienen múltiples DBMS´s posiblemente de diferentes tipos, y también múltiples DB´s pre-existentes. La integración es por lo tanto realizada por varios subsistemas. En la arquitectura de este tipo de DB´s existen tanto usuarios globales como locales.

Los MDBMS se dividen en dos clases, federadas y no federadas. En la clase no federada no existen los usuarios locales, es utilizada con relativamente poca frecuencia.

Las clases federadas se encuentran repartidas entre las que tienen un esquema global (fuertemente acopladas) y las que no lo tienen (débilmente acopladas). Las fuertemente acopladas manejan el esquema global, que es una vista lógica de todos los datos en la base de datos, es prácticamente la unión de todos los esquemas conceptuales locales de las DB´s, este esquema presenta una visión única y transparente para los usuarios de la base de datos, donde cada DBMS es libre para decidir que parte de su DB local contribuirá con el esquema global. A esta libertad de decisión se le conoce como autonomía local.

Las débilmente acopladas también llamadas, sistemas interoperables de base de datos, no poseen un esquema conceptual global. Esto es debido a que la construcción del esquema conceptual global es una tarea difícil y compleja que muchas veces requiere resolver diferencias semánticas y sintácticas entre sitios, y muchas veces estas diferencias son muy extensas que no garantizan la gran inversión que se aplica al desarrollo del esquema global [DBELL][TOZS].

2.4 El concepto de DBMS Componente: CDBMS

Con la incorporación del concepto de arquitectura de componentes en DBMS, es posible implantar las extensiones sin requerir que otras partes del sistema sean sobrescritas. Los componentes pueden ser suministradas por terceros y posiblemente por usuarios, incrementando la base desarrolladora de un DBMS. De ahí surge el concepto de DBMS´s Componentes ó CDBMS´s.

Aunque existen diferentes formas de CDBMS's, su base común es una arquitectura de componentes y la implementación de componentes con alguna clase de función de DB como soporte. Estos componentes pueden ser adicionados a un DBMS o usar de alguna manera para obtener el soporte a la DB. Por otra parte, una arquitectura de CDBMS también define lugares en el sistema (la parte variable) donde los componentes pueden ser adicionados [KDIT].

2.4.1 Fundamentos de CDBMS: ComponentWare

La arquitectura CDBMS toma la noción de ComponentWare. Esto es un paradigma propuesto recientemente para direccionar las cuestiones de reutilización (reusability), extensibilidad (extensibility) , apertura (openness), e interoperabilidad (interoperability) de Sistemas de Base de Datos. Esta es la noción que los sistemas de software son construidos en una manera disciplinada de construir bloques con propiedades específicas, llamados componentes [KDIT].

También se considera que un componente no debería tener un alto número de relaciones a otros componentes, pues esto podría restringir su potencial reutilización.

Los principios de ComponentWare se utilizan para entender mejor, abstraer, y clasificar los distintos enfoques para extender y personalizar DBMS´s. Además las características de ComponentWare son requerimientos cruciales para extensibilidad sistemática y bien definida, así como para integración. Por consiguiente las extensiones a un DBMS en este contexto son representadas como componentes, esto significa que deben cumplir las propiedades antes mencionadas de los componentes.

2.4.2 DBMS Orientado a Objetos (OODBMS)

Existen varias definiciones que han sido propuestas para un OODBMS. Una propuesta establece que un OODBMS proporciona al menos:

- Funcionalidad de base de datos.
- Debe de soportar objetos.
- Debe de proporcionar encapsulamiento.

La base de estos manejadores son los objetos, que son entidades únicas e identificables que contienen los atributos que describen el estado de un objeto en el mundo real y las acciones que se asocian con él [RDEC].

2.4.3 DBMS Relacional (RDBMS)

Una DBMS relacional es aquel en que la estructura de la base de datos se encuentra en forma de tablas y cuenta con las siguientes características [TOZS]:

- Las estructuras de datos son simples, relaciones que son tablas bidimensionales, cuyos elementos son los datos simples.
- El modelo relacional proporciona sólidos fundamentos para la consistencia de datos.
- Permite la manipulación de relaciones, mediante álgebra relacional.

2.5 Modelos de Base de Datos Componentes

En esta sección se presentan dos tipos de CDBMS's basados en el enfoque de componentes y su arquitectura: Componentes Plug-in y DBMS Configurables. Existen otros modelos como middleware y servicios en base de datos no considerados en este trabajo por su orientación al concepto de multidatabase. Detalles sobre estos modelos se muestra en [KDIT].

Componentes Plug-in. Este tipo de componentes permite "conectar" al DBMS características no estándar o funcionalidades que aún no son soportadas. De este modo, un sistema puede ser funcionalmente completo y cumplir requerimientos básicos, mientras las extensiones adicionan mas funcionalidad para necesidades específicas. Los componentes de esta clase de CDBMS son típicamente familias de tipos de datos abstractos o implementaciones de alguna función de DBMS tales como nuevas estructuras de índices. La figura 2.3 muestra el concepto de plug-in en DBMS, donde los componentes se integran al manejador agregándole funcionalidad ad-hoc.

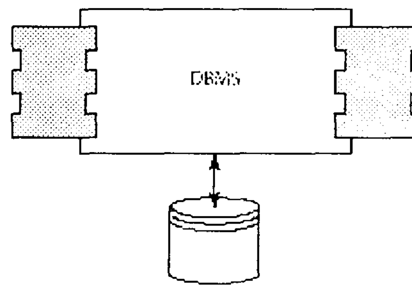


Figura 2.3 Dos componentes *plug-in* agregándole funcionalidad al DBMS

DBMS Configurables. Este enfoque permite desarrollar e integrar nuevas partes a un DBMS. A estos componentes se les puede considerar tareas desligadas del DBMS que pueden ser mezcladas y unidas posteriormente para dar soporte a la base de datos. Estos componentes son en realidad, subsistemas. Bajo este concepto, al desarrollar componentes selectos de un DBMS, se implementa la funcionalidad deseada y se obtiene un DBMS por configurar y ensamblar los componentes seleccionados. El DBMS es de ese modo, un DBMS completo. Este enfoque se muestra en la figura 2.4.

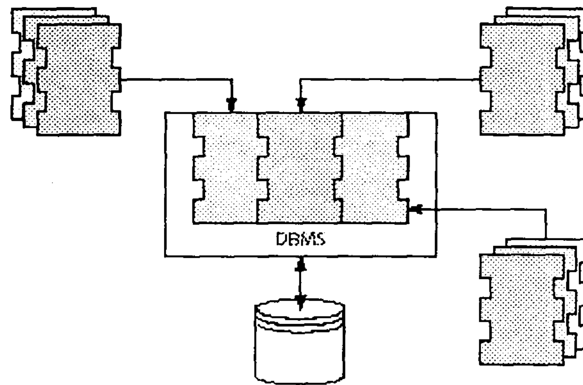


Figura 2.4 Un DBMS Configurable

2.6 Object Relational DBMS (ORDBMS) como un CDBMS

Un DBMS Componente (CDBMS) define un framework de sistemas de software para que los desarrolladores de aplicaciones puedan extender el DBMS al insertar módulos de programación lógica en éste. Un DBMS Objeto-Relacional (ORDBMS) es una clase de CDBMS.

Un ORDBMS puede soportar mecanismos de extensibilidad de componentes. Los usuarios finales enviando una consulta al sistema no tienen idea de cómo está implementada la extensión lógica que ellos invocan.

Lo que distingue un ORDBMS de otros frameworks componentes - tales como DBMS orientados a objetos, Middleware, y servidores de aplicaciones de propósito general [KDIT] -, es la manera en que, una vez integrados dentro del ORDBMS, los componentes son organizados y manipulados usando un lenguaje de Base de Datos declarativo, en vez de un lenguaje de programación procedural.

El sistema ORDBMS mas "joven" fue Post Ingres. Ahora, tres de los proveedores líderes –oracle, informix e IBM – han extendido sus sistemas para llegar a ser ORDBMS, aunque la funcionalidad dada por cada uno es ligeramente diferente. El concepto de ORDBMS, como un híbrido de RDBMS y OODBMS, es muy atractivo, preservando la riqueza del conocimiento y experiencia que ha sido adquirido con el RDBMS[RDEC].

2.6.1 El Modelo de datos abstracto de ORDBMS

El modelo de datos de ORDBMS está basado en el modelo de datos relacional. Los datos son organizados en relaciones, que corresponden a clases de hechos describiendo el estado del dominio del problema. Todas las características de integridad y consistencia del modelo de datos relacional, tales como llaves primarias y foráneas, pueden ser implementadas en un esquema

ORDBMS. La manipulación de datos, su recuperación y modificación es manejada a través de un lenguaje de programación declarativo. El más popular de estos es SQL.

2.7 ORDBM Distribuido

Un ORDBMS Distribuido difiere de un ORDBMS de simple sitio en que los objetos dentro del primero –los datos de objetos persistentes e implementación lógica de las extensiones – están físicamente distribuidos en un conjunto de sistemas de cómputo conectados en red (nodos). Como parte de sus operaciones, el sistema distribuido puede mover datos y lógica entre nodos para responder consultas de usuario tan eficientemente como sea posible (moviendo la implementación completa de algunos componentes lógicos entre nodos si es necesario). La topología de un ORDBMS Distribuido se muestra en la figura 2.5.

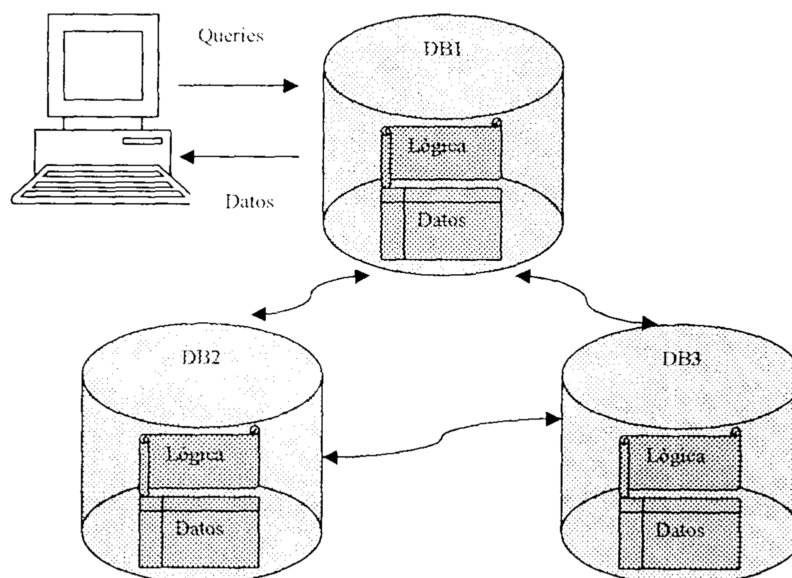


Figura 2.5 Topología de un ORDBMS Distribuido

Los DBMS componentes distribuidos tales como ORDBMS´s son útiles porque dirigen requerimientos que no son bien servidos por tecnologías de sistemas distribuidos tradicionales. Un ORDBMS Distribuido debe dirigir un conjunto de cambios que son la consecuencia de introducir extensiones definidas por el usuario en el modelo de datos (de un ORDBMS). Dependiendo de cómo es implementada, la extensión lógica puede ser móvil (lo que significa que puede ser copiada a través de la red para invocarlo dondequiera que se encuentre) ó ajustada (lo cual significa que puede correr solamente sobre un nodo particular de un sistema distribuido). Los productos RDBMS distribuidos pueden asumir que cada nodo en el sistema tiene funcionalidad idéntica o, en el caso de una

federación de productos RDBMS diferentes, tenga una funcionalidad similar (es decir que sea identificable). Pero con un DBMS componente distribuido, cada nodo podría contener conjuntos disjuntos de extensiones de componentes.

Muchos de los mecanismos desarrollados para administración de datos distribuidos en RDBMS´s pueden también ser aplicados en ORDBMS´s distribuidos. Por ejemplo, los modelos teóricos y técnicas de ingeniería usados para implementar que las transacciones distribuidas puedan ser reutilizadas [KDIT].

2.7.1 Extensiones Distribuidas para SQL

En RDBMS Distribuidos, el lenguaje SQL es modificado al adicionar nombres de sitio a los nombres de objetos esquema. Los objetos esquema en un sistema distribuido –tablas, vistas, y expresiones que hacen referencia a componentes lógicos- son completamente nombrados por una combinación de su nombre de sitio local y el nombre del sitio donde se originan. Por ejemplo la tabla *CallHistory* que se encuentra en el nodo *History* se denominará *CallHistory@History.*

En RDBMS Distribuidos, esta técnica fue aplicada para nombrar tablas y vistas. En ORDBMS Distribuidos esto también fue aplicado a lógica de componentes. Esto permite a una extensión integrada en un sitio ser referenciada de todos los otros sitios conectados. La llave para el procesamiento de consultas en ORDBMS son los metadatos (metadata). Cuando éste recibe una consulta, el ORDBMS local necesita crear una eficiente planificación de operaciones para calcular una respuesta. En ORDBMS Distribuidos, un sitio local podría necesitar acceder información sobre los datos y lógica residiendo en múltiples sitios remotos. Al menos, el nodo coordinador necesita verificar que ciertos objetos actualmente existen, y que este tiene permiso para accederlos.

Las propuestas para administrar catálogos distribuidos incluyen centralizar toda la información de ellos en un almacén, replicando éste entre todos los sitios participantes, o realizando chequeos remotos cada vez que las consultas son recibidas en el nodo coordinador. Estos enfoques son deficientes en alguna manera. La primera de las técnicas requiere que los cambios a los sitios locales sean propagados al almacén central o hacer un broadcast al conjunto entero de sitios interconectados. Esto desvía cada expresión DDL (Data Definition Language) en una transacción distribuida e implica una explosión en el número de mensajes cuando el número de sitios incrementa. Sin embargo leyendo desde catálogos remotos cada vez que una consulta es analizada y optimizada hace la compilación de la consulta más costosa.

En la práctica, diferentes sistemas relacionales distribuidos han adoptado diferentes estrategias. Muchos sistemas de investigación y sistemas comerciales recientes han optado por el esquema globalmente replicado porque se consideró

que los cambios de esquema eran relativamente eventos raros. Sin embargo, en ORDBMS Distribuidos hay una gran variedad y numero de objetos esquema, y en el desarrollo de aplicaciones se hacen actualizaciones y adiciones de nuevos componentes de manera regular. La frecuencia de cambios de esquema en ORDBMS Distribuidos será alta y hay argumentos para la estrategia de análisis de tiempo de chequeos remotos.

Por otra parte, el hecho que un nombre de sitio sea incluido en la especificación de la consulta no implica que ésta se encuentre en el sitio sobre el cual la lógica llamada esté actualmente ejecutándose. En primer lugar, una implementación equivalente de la función podría estar disponible en otro nodo (esta equivalencia necesita ser anotada en los catálogos del sistema). Mover los datos a ese nodo e invocar la lógica específica podría ser una mejor manera de procesar la consulta, y segundo, los ORDBMS´s Distribuidos pueden optar por tecnologías componentes para mover la implementación de lógica componente entre nodos del sistema.

El principio de transparencia de localización aplica no solo a donde los datos del sistema distribuido estén localizados, sino también a donde la lógica especificada en una consulta esté actualmente ejecutándose.

Capitulo 3 Análisis de arquitecturas.

En este capítulo analizaremos diferentes arquitecturas de DBMS, como Oracle y DB2 entre los comerciales así como MySQL, PostgreSQL, e InnoDB entre los open-source, y describiremos los elementos que las conforman.

3.1 Arquitectura Oracle

El servidor Oracle es un sistema manejador de base de datos relacionales que proporciona un entorno abierto, integrado y completo para el manejo de la información. Los componentes primarios de Oracle son la instancia y la base de datos.

La instancia es una combinación de procesos en segundo plano y estructuras de memoria. La instancia debe de ser iniciada para acceder a la información de la base de datos. Cada vez que inicia una instancia, un área de sistema global (SGA) es desplegada y son iniciados los procesos. La SGA es un área de memoria utilizada para guardar información compartida entre los procesos de la DB [MABB].

Los archivos de base de datos son archivos de sistema operativo, proporcionan el actual almacenamiento físico de la información de la base de datos. Son utilizados para asegurar la consistencia de los datos, y la recuperación de los mismos en caso de fallo de la instancia. En la figura 3.1 se observa el esquema general de la arquitectura de Oracle [BERN].

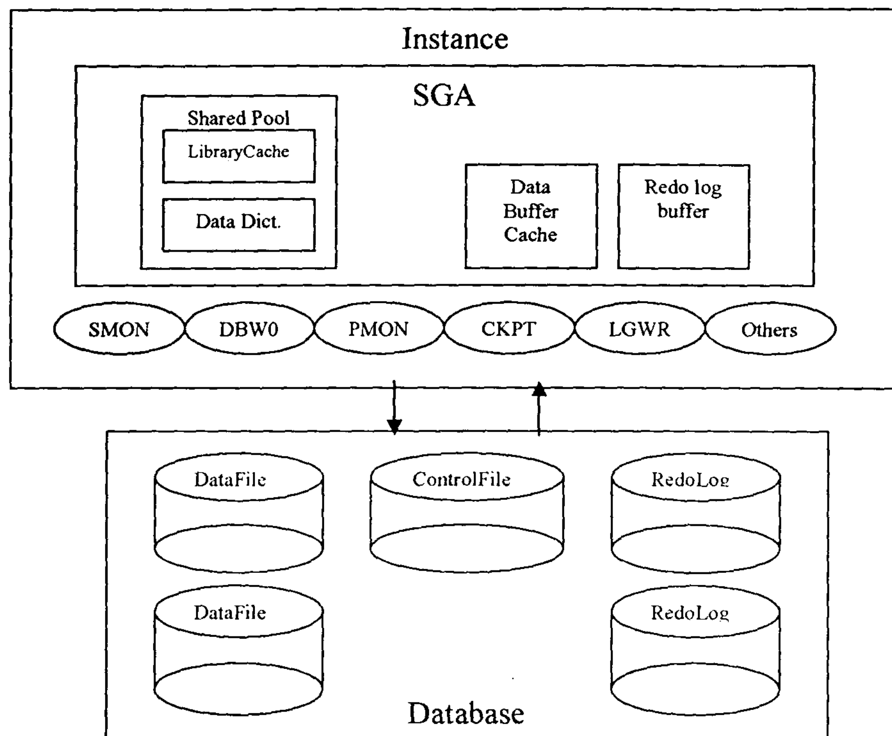


Figura 3.1 Arquitectura interna de Oracle

La base de datos contiene tres tipos de archivos [BERN][MABB]:

Archivos de datos (Data files). Contienen la información de la base de datos. Los datos son almacenados en tablas definidas por los usuarios, aunque también contienen el diccionario de datos, índices y otras estructuras propias del manejador. Una base de datos contiene al menos un archivo de datos.

Registro de rehacer (Redo log). Son archivos del sistema operativo en los que Oracle registra las modificaciones o transacciones que se registran en la base de datos. Permite la recuperación de datos en caso de fallo. Una DB Oracle tiene al menos dos registros de rehacer.

Archivos de control (Control file). Contienen información necesaria para mantener y verificar la integridad de la DB. Una DB necesita al menos un archivo de control, por ejemplo un archivo de control es utilizado para identificar los archivos de datos y registros de rehacer propios de una BD.

La instancia de Oracle contiene los siguientes componentes:

Área de sistema global (System Global Area). Es un área de memoria utilizada para almacenar información de la DB compartida por los procesos de

la misma. Contiene datos de control para el servidor oracle. Se encuentra localizada en la memoria virtual de la computadora donde reside el servidor oracle. Consiste de varias estructuras de memoria como son el caché de buffer de datos, el buffer del registro de rehacer y memoria compartida.

Caché de diccionario (Dictionary Cache). También conocido como caché de filas, es una colección de las definiciones utilizadas más recientemente de la base de datos. Incluye información de los archivos de la base de datos, tablas, índices, columnas, usuarios, y privilegios.

Caché de librería (Library Cache). Almacena información acerca de los enunciados sql mas recientemente utilizados. Utiliza una estructura de memoria llamada área compartida sql. Esta estructura de memoria contiene

- El texto de la consulta sql
- El arbol de compilación
- El plan de ejecución, los paso necesarios para ejecutar la consulta

Caché del buffer de datos (Data Buffer Cache). Es la memoria donde Oracle almacena los bloques de datos mas recientemente utilizados. Cuando una consulta es procesada, el proceso del servidor busca en el buffer los bloques que necesita, si el bloque no se encuentra, lee el data file y almacena una copia en el buffer de datos.

Buffer del registro de rehacer (Redo Log Buffer). Los servidor oracle registra los cambios realizados a los bloques de datos en el redo log, por lo que es utilizado para rastrear los cambios hechos a la DB por el servidor y los procesos. Los registros se realizan de manera secuencial por lo que los cambios realizados en una transacción pueden encontrarse intercalados con los cambios realizados a otra transacción.

Escritor del registro (LGWR). Realiza la escritura secuencial del buffer de registro de transacciones al archivo de transacciones en las siguientes situaciones:

- Cuando una transacción se completa
- Cuando el redo log buffer tiene un espacio vacío de un tercio de su capacidad.
- Cuando existe mas de un megabyte de cambios registrados en el redo log buffer.

Escritor de la BD (DBW0). Este proceso escribe los buffers utilizados del buffer de datos a los archivos de datos. Es uno de los dos únicos procesos que permiten escribir a los archivos de datos que constituyen la base de datos

Oracle. Conciertos sistemas operativos, por razones de rendimiento permite tener múltiples escritores de DB.

Monitor del sistema (SMON). Es un proceso obligatorio, lleva a cabo cualquier recuperación necesaria durante la inicialización. En el modo servidor paralelo puede recuperar una DB que haya fallado en otra computadora.

Monitor de proceso (PMON). Es un proceso obligatorio que lleva a cabo la tarea de recuperación cuando un usuario de la DB falla. Asume la identidad del usuario que falla, libera todos los recursos de la DB que tenía el usuario y deshace la transacción interrumpida.

Punto de comprobación (CKPT). Cuando los usuarios están trabajando en una DB Oracle realizan solicitudes para acceder a los datos. Dichos datos se leen de los archivos de la DB y se colocan en un área de memoria donde los usuarios pueden consultarlos. Finalmente algún usuario realiza un cambio en los datos, que debe registrase de nuevo en los archivos de datos originales. Cuando se produce la conmutación de los registros de rehacer, se produce un checkpoint o punto de comprobación. En ese momento, Oracle consulta la memoria y escribe la información de los bloques de datos modificados en el disco. También notifica al archivo de control que se ha producido una conmutación de los registros de rehacer. Estas son las tareas que realiza este proceso [MABB].

3.2 Arquitectura DB2

La arquitectura del manejador de DB2 consta de dos tipos de instancias, el servidor de administración y la instancia DB2. A continuación explicaremos el funcionamiento de las mismas [DB2].

Servidor de administración DB2. Es una instancia especial utilizada para manejar local y remotamente servidores DB2. Se debe de ejecutar el servidor para utilizar el asistente de configuración del cliente o el centro de control. El servidor influye en el centro de control y en el asistente de configuración de cliente al realizar las operaciones siguientes [DB2]:

- Habilitación de la administración remota de servidores DB2
- Detención y arranque de instancias DB2
- Proporciona medios para descubrir información acerca de la configuración de las instancias DB2, bases de datos y otros servidores de administración DB2 que se encuentren en la utilería Discovery DB2. Esta información es utilizada por el asistente de configuración de cliente y el centro de control para simplificar y automatizar la configuración de conexiones de clientes a las bases de datos.

Solamente se requiere un servidor de administración en cada servidor. Esta instancia es configurada durante la instalación para arrancar cuando el sistema operativo inicia.

La instancia DB2. La instancia DB2 es el ambiente lógico del servidor de base de datos donde se catalogan las bases de datos y se establecen los parámetros de configuración. Dependiendo de las necesidades de la empresa, se pueden crear más de una instancia. Algunas de las razones por las cuales utilizar múltiples instancias son [DB2]:

- Cuando se requiere utilizar una instancia para desarrollo y una para producción.
- Al adaptar una instancia a un ambiente particular.
- Para restringir el acceso a cierta información.
- Para instalar diferentes versiones de DB2 en la misma máquina.
- Controlar la asignación de autoridad en SYSADM, SYSCTRL y SYSMAINT para cada instancia.
- Optimizar la configuración del manejador de la base de datos para cada instancia
- Limitar el impacto del posible fallo de una instancia, las demás pueden funcionar independientemente.

Algunas desventajas podrían ser:

- Utilización adicional de recursos del sistema (memoria virtual, espacio de disco).
- Más administración, debido a la adición de las instancias.

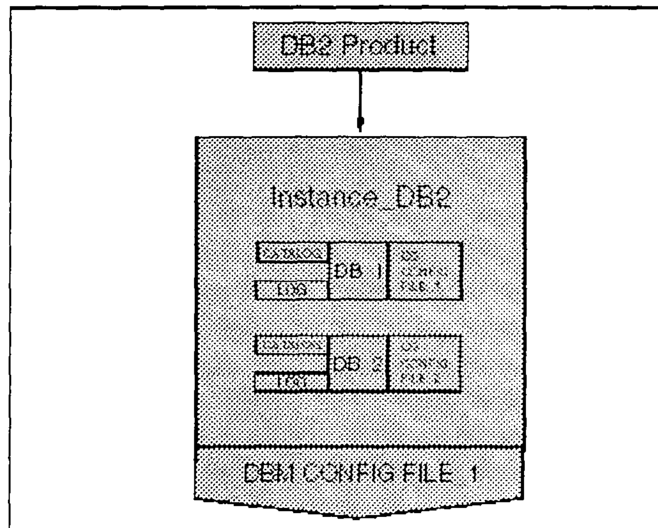


Figura 3.2 Instancia base de datos DB2

En la figura 3.2 observamos la arquitectura del manejador DB2 y describimos a continuación los módulos que la conforman [DB2].

Base de datos. La base de datos relacional DB2 presenta los datos como una colección de tablas. Las tablas consisten en un número definido de columnas y n número de filas. Cada base de datos incluye:

- Un conjunto de tablas del sistema que describen la estructura lógica y física de los datos.
- Un archivo de configuración que contiene los valores de los parámetros de la DB.
- Un archivo de registro (log) con el registro de las transacciones realizadas y las que se realizan al momento.

- **Espacio de tablas (Tablespaces).** Una base de datos esta organizada en tablespaces. Cada base de datos puede tener varios tablespaces asociados. Los tablespaces son creados dentro de la DB y las tablas son creadas dentro de los tablespaces. Se pueden utilizar diferentes tipos de tablespaces para almacenar diferentes tipos de información. Los tablespaces están compuestos de múltiples contenedores. Un contenedor es una localidad de almacenamiento físico (como un archivo o un dispositivo). Existe una relación de "uno a muchos" entre los tablespaces y los contenedores. Se pueden definir múltiples contenedores para un tablespace. Sin embargo, un contenedor solo puede ser asignado a un tablespace.

  DB2 soporta dos tipos de tablespaces:

  - Espacio de tablas administrado por el sistema (SMS).
  - Espacio de tablas administrado por la base de datos (DMS).

Para un SMS tablespace, cada contenedor es un directorio en el sistema de archivos del sistema operativo y es éste quien controla el espacio de almacenamiento conforme se requiera. En un DMS tablespace, el almacenamiento es predeterminado a un contenedor cuando el contenedor es creado. Los contenedores pueden ser añadidos dinámicamente a un DMS tablespace. En estos casos es el DBA el encargado de administrar el espacio de almacenamiento como se requiera [DB2].

Catálogos (Catalogs). Cada DB incluye un conjunto de tablas de sistema que describen la estructura física y lógica de la información. DB2 crea y mantiene un conjunto extenso de catálogos de tablas de sistema. Estas tablas contienen información acerca de las definiciones de objetos de la DB como tablas, índices, vistas, así como información de seguridad (permisos). Son creadas cuando se genera la DB y se actualizan durante el curso de operación normal.

Triggers. Un trigger es un conjunto definido de acciones que son ejecutadas cuando un evento específico definido para ese trigger ocurre. Los eventos trigger pueden ser de inserción, eliminación o modificación en contra de una tabla específica.

Tablas (Tables). Una tabla consiste en datos ordenados en columnas y filas. Los datos en las tablas están lógicamente relacionados y las relaciones pueden ser definidas entre tablas. La información puede ser manipulada basándonos en principios matemáticos y operaciones llamadas relaciones.

La información es accedida a través del lenguaje de consulta estructurado (SQL) que es un lenguaje estandarizado para definir y manipular datos en una DB relacional.

Vistas. Una vista es manera eficiente de representar información sin necesidad de mantenerla. Es decir es una tabla temporal que no requiere almacenamiento permanente. Es una tabla virtual creada utilizada y eliminada.

Buffer Pools. Es una cantidad de memoria principal diseñada para captar tablas e índices tan pronto se lean del disco o sean modificadas. El propósito del buffer pool es mejorar el rendimiento del sistema. La información puede ser accedida más rápido de la memoria que del disco. Por lo tanto, mientras menos acceda el manejador de la DB al disco, mejor será el rendimiento. Se pueden manejar más de un buffer pool, aunque la mayoría de las veces con uno es suficiente [DB2].

3.3 Arquitectura MySQL

MySQL es una base de datos open source diseñada para proporcionar velocidad, poder y precisión en proyectos de misión crítica y uso pesado desarrollada por MySQL AB [MYSQ]. La arquitectura de MySQL se subdivide en cinco grandes subsistemas, el procesador de queries (query engine), el manejador de buffer (buffer manager), el controlador de transacciones (transaction control), el manejador de almacenamiento (storage manager) el manejador de recuperación (recovery manager). También incluye dos componentes que son el manejador de procesos (process manager) y las utilerias (utilities) [GTAR] En la figura 3.3 observamos la arquitectura del manejador MySQL, que a.continuación describimos.

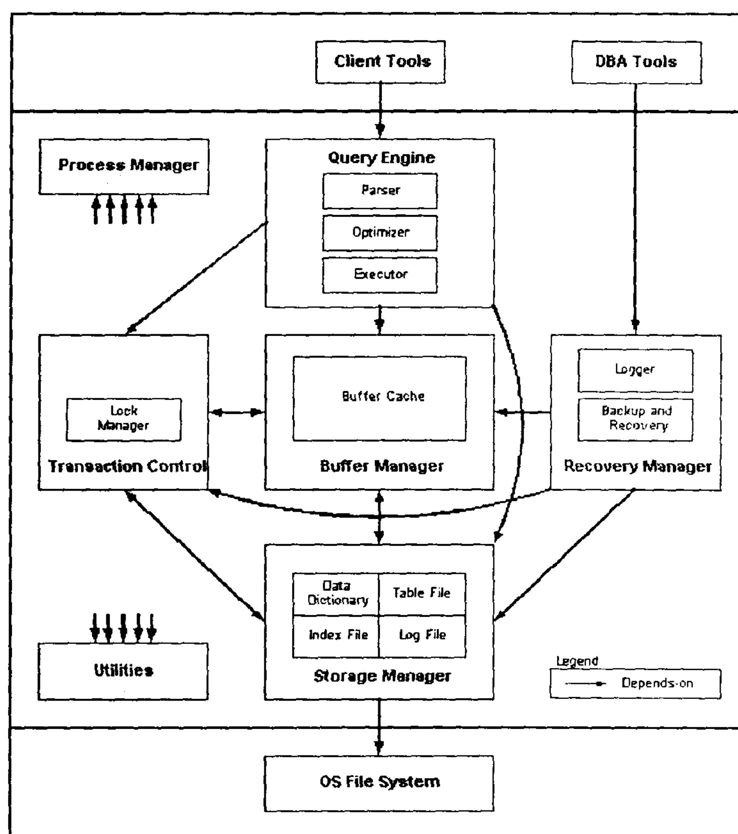


Figura 3.3 Vista general de la arquitectura del manejador MySQL

Procesador de queries (Query engine). Recibe los comandos SQL provenientes de los usuarios, realiza una optimización de los comandos, realiza un plan de ejecución, lo ejecuta y envía de regreso al usuario la respuesta a su solicitud. Contiene tres subsistemas, el analizador sintáctico, optimizador de queries y el ejecutor.

Manejador de buffer (Buffer manager). Reside entre el manejador de queries y el manejador de almacenamiento, Este componente es el responsable de la administración de la memoria. Realiza copias de los datos en el buffer para acceder con mayor rapidez a la información. Para proporcionar los datos solicitados por el procesador de queries, almacena la parte activa de los datos provenientes de los archivos de datos en la memoria principal y realiza los cambios necesarios en el manejador de almacenamiento.

Controlador de transacciones (Transaction control). Este subsistema realiza la función de facilitar la concurrencia en el acceso a los datos. Proporciona el bloqueo en las tablas para asegurar que los múltiples usuarios que accedan a la información, lo hagan de manera consistente. Este componente incluye el Manejador de bloqueo.

Manejador de almacenamiento (Storage manager). Este componente se encarga de mantener persistente y eficientemente el almacenamiento y recuperación de la información de los archivos de sistema del sistema operativo. Los tipos de archivos manejados por este módulo son archivos que almacenan la información de las tablas, índices, bitácoras, y el diccionario.

Manejador de recuperación (Recovery manager). Este subsistema tiene la responsabilidad de mantener copias de la información, con el fin de recuperarla en caso de pérdida de datos. También realiza una bitácora de los comandos que modificaron la información y otros eventos importantes dentro de la base de datos. La bitácora y el respaldo de recuperación son los componentes de este subsistema.

Manejador de procesos (Process manager). Realiza dos funciones en el sistema. Uno de ellos es administrar las conexiones de los usuarios. Esto involucra módulos para la administración de conexiones de red con clientes y eliminar conexiones de usuarios. La segunda es la sincronización entre las tareas y procesos. Estas funciones involucran módulos para multi.threading y thread locking, para llevar a cabo operaciones seguras de threads.

Utilerias (Utilities). Es un proveedor común para los demás subsistemas. Contiene rutinas de propósito general, las cuales son utilizadas por los diversos subsistemas del manejador. Incluye rutinas como son: manipulación de cadenas, operaciones de ordenamiento, funciones abstractas del sistema operativo como archivos de I/O y designación o liberación de memoria [GTAR].

En el siguiente capítulo propondremos una arquitectura que se adapte a los requerimientos de un manejador de base de datos distribuidas, utilizando como referencia el manejador MySQL, incluyendo en ella los componentes necesarios para realizar las funciones de distribución de datos.

3.4 Arquitectura PostgreSQL.

PostgreSQL es un sistema manejador de bases de datos objeto relacional (ORDBMS) basado en postgres, versión 4.2 desarrollado en la Universidad de California en Berkeley. Como un manejador de este tipo, ofrece la herencia y nuevos tipos de datos que son parte de lenguajes de programación orientados a objetos además de las características tradicionales de un DBMS relacional [POST]. Basándonos en la figura 3.4 que representa la arquitectura del manejador, definiremos las funciones que realiza cada uno de los principales componentes del mismo.

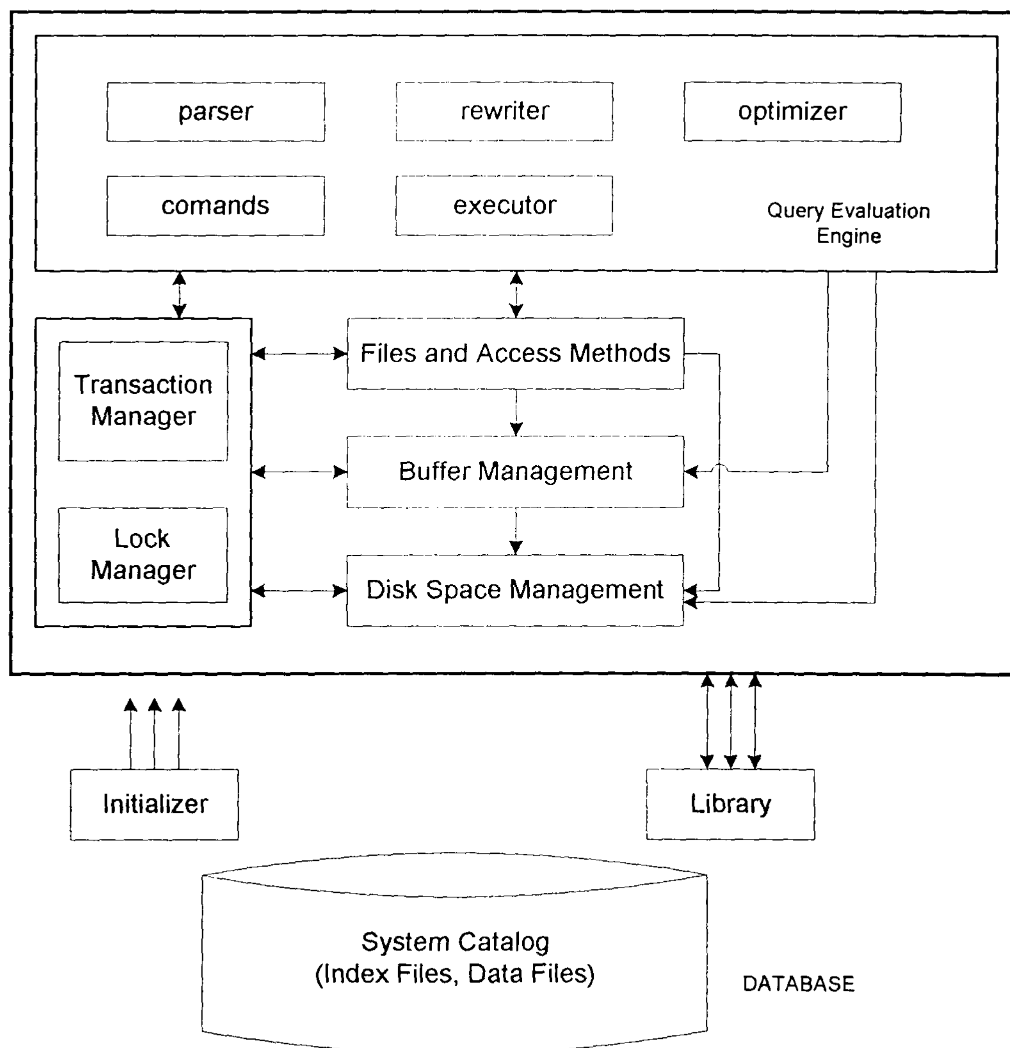


Figura 3.4 Arquitectura del ORDBMS postgreSQL.

Inicializador (Inicializer). Consiste en los módulos de Postmaster y el Bootstrap. EL postmaster es un demonio (proceso) y el Bootstrap es el creador de la instancia de la base de datos.

Subsistema de archivos y métodos de acceso. (File and access method subsystem). En este subsistema se encuentran los métodos que proporcionan el soporte para varios mecanismos de acceso a la información. La definición y manipulación del catálogo del sistema también se realiza en este módulo. Existen dependencias entre este subsistema y el modulo de control de concurrencia para las operaciones de acceso concurrente y al Manejador de buffer y librerías para las operaciones de paginación y varios tipos de utilidades. Otra dependencia va de este módulo al módulo de Evaluación de consultas, específicamente al ejecutor.

Manejador de buffers (Buffer management). El manejo del buffer contiene dos partes: el buffer y la paginación, las cuales se encargan del buffer de acceso al disco y de las operaciones de paginación respectivamente. El buffer proporciona dos mecanismos separados de control de acceso a disco para buffers compartidos. Contadores referenciales y buffers de bloqueo. Existen dos tipos de buffers de bloqueo, el compartido y el exclusivo.

La paginación implementa operaciones de paginación como inicializaciones, adicion de elementos a paginas, etc.

Manejador de espacio de disco (Disk space manager subsystem). Proporciona operaciones relacionadas, con la administración del espacio de disco. Algunas de las funciones que se realizan en este subsistema son, inicialización y eliminación de los manejadores de espacio de disco, calcula el número de bloques de postgres en la relación utilizada, crea la relación en el dispositivo y regresa su archivo de descripción, etc.

Control de concurrencia (Concurrency control). Este subsistema consiste en dos módulos, el manejador de transacciones y el manejador de bloqueos. El manejador de transacciones soporta la concepción de las transacciones. El manejador de bloqueos se encarga de coordinar la comunicación entre procesos y la administración de los bloqueos. Este subsistema proporciona servicios de bloqueo y comunicación entre procesos a otros subsistemas como el manejador de buffers, de espacio de disco y el procesador de consultas. Por otra parte utiliza el subsistema de archivos y métodos de acceso para las definiciones del catálogo de nombres.

Librería (Library). Proporciona las estructuras de datos y funciones compartidas por otros subsistemas. Algunas de las principales utilerías que maneja son: rutinas para el manejo de cadenas, funciones de caché, llamadas dinámicas a funciones precargadas, administración de memoria, rutinas de ordenamiento, etc.

Mecanismo de evaluación de queries (Query Evaluation Engine). Este subsistema es el más importante de la arquitectura del DBMS PostgreSQL. Es el encargado del procesamiento de las consultas generadas por los usuarios. Debido a la importancia de este subsistema vamos a analizar los módulos que lo conforman más detalladamente.

- **PT (TCOP).** El PT tiene una función similar a un policía de tráfico. Se encarga de administrar las peticiones provenientes de la aplicación enviándolas al modulo que le corresponda. Este módulo contiene el principal manejador de postgres, así como el código que realiza las llamadas al analizador, optimizador, ejecutor, y funciones de comando. Este módulo depende de todos los demás módulos del MEC. También envía la cadena de consulta relacionada con las transacciones al subsistema de control de concurrencia.
- **Analizador (Parser).** Este módulo tiene como propósito convertir las consultas SQL en estructuras de comandos específicos. Entre sus funciones se encuentran agrupar las consultas en palabras claves, identificadores y constantes, crear las estructuras de comandos específicos para soportar los elementos de la consulta, verificar estas estructuras, definir la estructura de datos de los resultados del analizador.
- **Escritor (Rewriter).** Realiza el procesamiento para las reglas del sistema. El escritor sobre escribe las consultas mediante el sistema de escritura de consultas. Es en este subsistema donde se definen las reglas de escritura, interfaces externas para la escritura de las consultas, subrutinas de manipulación de consultas, eliminación de reglas, etc. El escritor depende del analizador, optimizador y el módulo de comandos del subsistema de evaluación de consultas. Depende del analizador dado que sobre escribe los resultados que éste genera. También requiere soporte por parte del optimizador para interactuar con las reglas y del modulo de comandos para interactuar con las vistas.
- **Optimizador (Optimizer).** Se encarga de la estructura de la consulta que proporciona el analizador y genera un plan óptimo que utilizará el ejecutor. El optimizador se compone de los siguientes elementos:

    - **Ruta.** Toma como instancia la salida generada por el analizador y genera todas las posibles formas de ejecutar la petición, asignando un costo a cada una de ellas.
    - **OCGE.** Es un optimizador de consultas genético. Cuando el número de tablas es muy grande, la cantidad de posibles pruebas es muy grande también. Este método considera cada tabla por separado, y supone el orden óptimo para realizar un join. Para un número pequeño de tablas, este mecanismo es tardado, pero cuando se trata de un número considerable de tablas, es muy rápido. Existe una opción para activar este mecanismo en el manejador.

- **Plan.** Toma como instancia la salida que genera el método Ruta, escoge la ruta o camino con el menor costo, genera un plan para que realice el ejecutor.
- **Prep.** Realiza planes de procesamiento especiales, como convertir una expresión booleana en una forma normal conjuntiva o disyuntiva.
- **Util.** Contiene subrutinas utilizadas por otros módulos del optimizador.

- **Ejecutor (Executor).** El ejecutor es el encargado de manejar las sentencias select, insert, update, y delete. Las operaciones requeridas para manipular estas sentencias incluyen recorrido de heaps, recorrido de índices, ordenamientos, uniones de tablas, agrupaciones, agregados y unicidad.
- **Comandos (Commands).** Este módulo procesa las sentencias SQL que no requieren manipulación compleja, algunas de ellas pueden ser vaccum, copy, alter, create table, create type, etc.

3.5 Arquitectura de InnoDB

InnoDB es esencialmente un sistema manejador de base de datos stand-alone que actualmente está ganando mucha popularidad, de hecho, MySQL con el fin de ofrecer una funcionalidad superior, sin comprometer el desempeño, lo ha integrado a su sistema [RBAN].

Esta adición proporciona a los usuarios de MySQL, la capacidad de realizar transacciones robustas y recuperación de errores, bloqueo a nivel de registros, así como las restricciones de llaves foráneas.

El modelo transaccional de InnoDB está entre los mejores del mercado ya que combina lo mejor de las propiedades de una base de datos multiversión, así como el bloqueo tradicional de dos fases.

El bloqueo se efectúa a nivel registro y las consultas se generan por default como solo lectura consistente sin bloqueo, al estilo de Oracle. La tabla de bloqueo en el InnoDB se almacena de forma tan eficiente que el bloqueo escalonado no se requiere, típicamente a varios usuarios se les permite bloquear todos los registros en la base de datos, o cualquier subconjunto aleatorio de registros, sin que InnoDB sufra de algún desborde de memoria.

A continuación analizaremos los módulos de la figura 3.5 que componen la arquitectura de este manejador [RBAN].

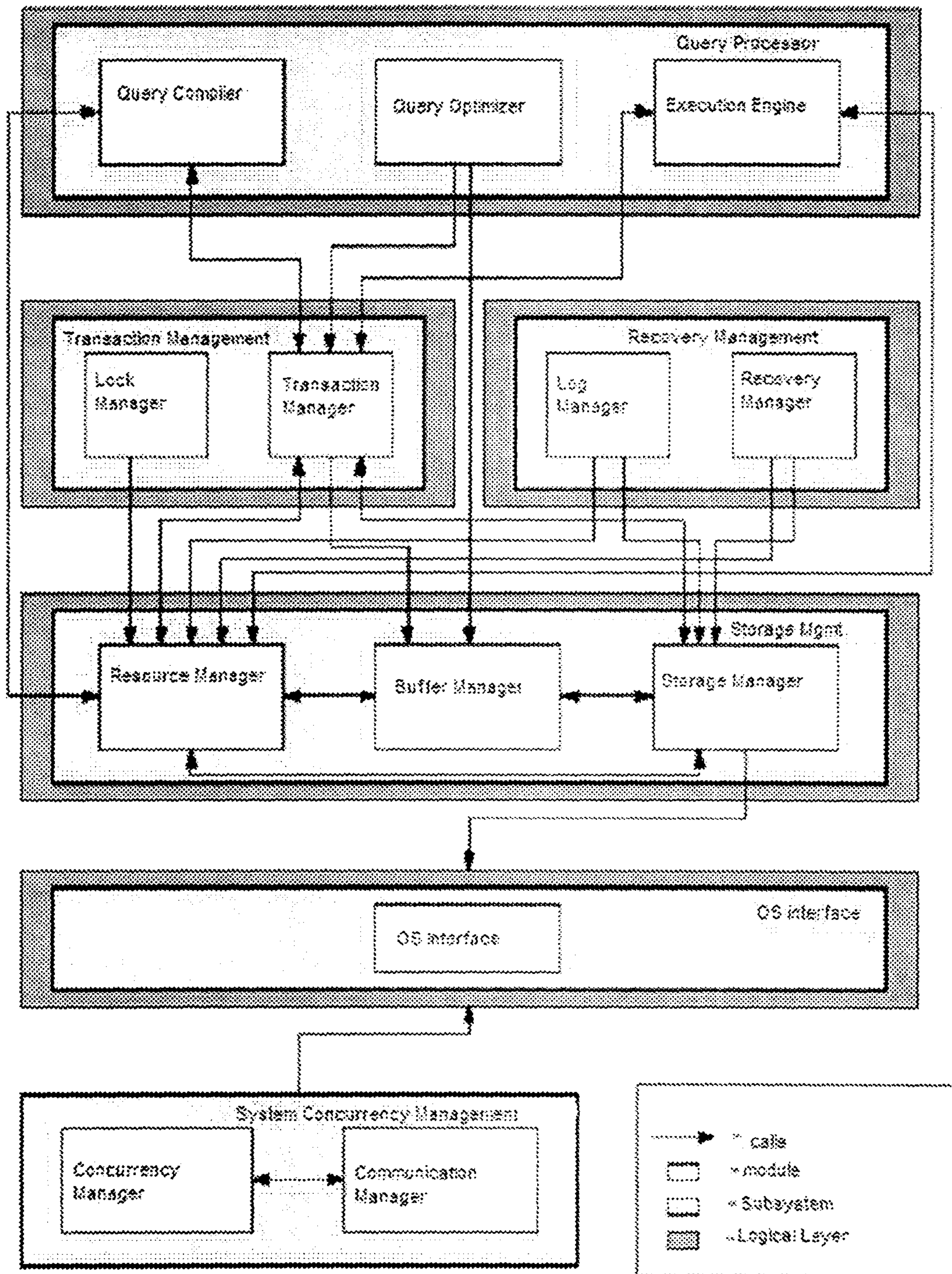


Figura 3.5 Arquitectura del DBMS InnoDB

Subsistema procesador de consultas (Query processor subsystem).

Compilador de consultas (Query compiler). Este módulo efectúa la compilación y el análisis léxico de de las consultas realizadas. Para compilar la consulta, el compilador debe obtener acceso a las tablas y registros en cuestión, así como a las transacciones para poder generar el árbol de compilación. Para tal función recibe los recursos (tablas y registros) del manejador de recursos. Los componentes basados en transacciones son obtenidos del manejador de transacciones.

Optmizador de consultas (Query optimizer). El propósito de este módulo es optimizar la consulta para que se ejecute eficiente y rápidamente. Cuando sucede un rollback en una recuperación de la base de datos, el optimizador de consultas optimiza las transacciones que se encuentran en la bitácora. El compilador de consultas genera un árbol de compilación, el cual se encuentra en memoria, y para optimizarlo, el optimizador debe ponerse en contacto con el manejador de buffers para poder recuperarlo exitosamente.

Ejecutor (Execution Engine). Una vez que la consulta se encuentra optimizada para su máximo desempeño, está lista para su ejecución. Los componentes de una consulta de una transacción específica se realizan mediante llamadas al manejador de transacciones. El ejecutor utiliza la utilería de comparación de registros del manejador de recursos.

Subsistema Manejador de transacciones (Transaction manager subsystem).

Manejador de bloqueos (Lock manager). En el modelo transaccional del InnoDB, el objetivo ha sido el combinar las mejores propiedades de las bases de datos multiversión y el bloqueo de dos fases tradicional. InnoDB realiza el bloqueo a nivel registro, con bloqueos de lectura y escritura. Varios usuarios de las tablas y registros en la base de datos utilizan una tabla de bloqueo para rastrear el estado de los bloqueos. El manejador de bloqueos utiliza la tabla de bloqueo que proviene del manejador de recursos.

Manejador de transacciones (Transaction manager). En InnoDB toda la actividad de los usuarios se realiza dentro de transacciones. Si el modo de auto-commit esta siendo utilizado en MySQL, entonces cada comando SQL formará una sola transacción. Las transacciones SQL o enunciados como el commit o rollback son compilados e identificados por el compilador de consultas como parte de la consulta realizada por el InnoDB. Las unidades atómicas de la transacción de la consulta del InnoDB son identificadas y planificadas por el manejador de transacciones. Es entonces cuando el ejecutor las realiza. El manejador de transacciones revisa la bitácora de registros del manejador de recursos para realizar las transacciones que deban de ser realizadas para propósitos de recuperación. Es el responsable de iniciar transacciones, realizar el commit de las transacciones, asignar segmentos de rollback para cada transacción, así como

funcionalidad para establecer las transacciones en varios estados y colas. El manejador de buffers localiza las colas y las transacciones, mientras el acceso a los archivos de bitácora se obtiene del manejador de almacenamiento.

Subsistema manejador de recuperación (Recovery Management Subsystem).

Manejador de bitácora (Log Manager). Es el responsable de mantener una bitácora de las operaciones que se realizan en la base de datos para fines de recuperación y respaldo. Cuando se realiza una operación de recuperación o respaldo, el InnoDB revisa los archivos de bitácora a partir de un punto de verificación aplicando las modificaciones registradas hechas a la base de datos.

Manejador de recuperación (Recovery Manager). InnoDB revisa automáticamente los archivos de bitácora (vía el storage manager) y realiza un roll-forward de la base de datos. InnoDB deshace las transacciones que no fueron completadas y estuvieron presentes en el momento de la caída del sistema. Al realizar esta operación el recovery manager accede a funciones, datos, y registros de bitácora (Manejador de recursos).

Subsistema de control de concurrencia (System Concurrency Subsystem)

Manejador de concurrencia (Concurrency Manager). El manejador de concurrencia contiene la implementación de los bloqueos de lectura y escritura que se requieren para la sincronización de operaciones. Utiliza la implementación de memoria compartida del manejador de comunicaciones.

Manejador de comunicación (Communication Manager). Implementa las primitivas de comunicación. En este módulo se establecen la interfaz de comunicación y la implementación de memoria compartida para la interface. Para realizar estas tareas, el manejador de comunicación debe realizar llamadas a las primitivas de memoria compartida del sistema operativo.

Subsistema manejador de almacenamiento (Storage Manager Subsystem).

Manejador de almacenamiento (Storage Manager). Es el responsable de proporcionar acceso eficiente a los archivos y bitácoras de la base de datos. Contiene mecanismos de manipulación de archivos a bajo nivel, incluyendo los métodos y estructuras para manipularlos. Recibe comandos del manejador de buffer e interactúa con el sistema operativo para regresar información de la base de datos.

Manejador de buffer (Buffer Manager). Es responsable del almacenamiento eficiente de datos en la memoria para optimizar la manipulación. Realiza decisiones acerca de que bloques de memoria deben permanecer y cuales deben de regresar a disco.

Manejador de recursos (Resource Manager). La responsabilidad del manejador de recursos es aceptar peticiones de niveles superiores, bloqueos, transacciones, o registros de bitácora y traducirlas en un formato apropiado de "tabla" que pueda ser entendido por el manejador de buffer. Ya que los datos hayan sido cargados en memoria, el manejador de recursos contiene la funcionalidad de manipulación de los datos. Entre sus funciones están la depuración de registros obsoletos, actualización de índices, reconstrucción de registros, etc.

Interface del sistema operativo (Operating System Interface). Permite el acceso al archivo de acceso de I/O, a la memoria compartida, sincronización, manejo de threads, y primitivas de control de proceso. Los módulos de concurrencia y comunicación, así como el manejador de almacenamiento acceden a este módulo de forma predominante.

3.6 Conclusiones

En este capitulo se analizaraon las arquitecturas de algunos DBMS tanto comerciales como el Oracle y el DB2, como los open-source como el MySQL, PostgreSQL e InnoDB. Todos tienen módulos en común pero también encontramos algunas diferencias en las arquitecturas.

En el caso de Oracle y DB2 se observa que una amplia cantidad de módulos forman parte de la arquitectura, cada uno con una función específica, y esto es en parte, debido a que son productos comerciales con amplia investigación, soporte técnico, publicidad, etcétera y, por lo mismo, requieren tener respaldo para las empresas que utilizan el producto. Sin embargo el aspecto de distribución se maneja con una lógica muy propia del manejador.

En el caso de los DBMS open source, se encontraron algunas diferencias entre las arquitecturas, por ejemplo se tiene que el DBMS de MySQL no posee un transaction manager propio, sino que se apoya de otro DBMS, el InnoDB, el cual le proporciona esa funcionalidad. El PostgreSQL es un manejador suficientemente completo, cuanta con los módulos muy interesantes que lo hacen uno de los mejores representantes de los DBMS libres.

De todos ellos se tomarán algunos conceptos que se consideran útiles para proponer una arquitectura estándar basada en componentes, adicionando los módulos necesarios para implantar el aspecto de distribución, de tal manera que se conserven las características de extensibilidad, adaptabilidad, y apertura, proporcionando mayor funcionalidad al sistema.

Capitulo 4 Propuesta de arquitectura distribuida.

4.1 Introducción.

En este capítulo se propondrá una arquitectura propia para DBMS basada en el manejador de base de datos MySQL, que permita la distribución de datos y mejore la estructura actual del MySQL. El objetivo es que esta arquitectura pueda servir como estándar para diseñar manejadores de base de datos distribuidos. Se toma como base la arquitectura actual de MySQL que se analizó en el capitulo anterior, en ella se adicionan algunos módulos e interrelaciones que se consideran pueden mejorar el desempeño del manejador de la base de datos y adicionar la funcionalidad distributiva para la base de datos.

4.2 Arquitectura del sistema global

En el capitulo tres se describen los principales componentes de la arquitectura de MySQL, veamos ahora una arquitectura global del sistema, a partir de ella se añadirán los módulos necesarios para complementar la estructura completa de un manejador de bases de datos distribuido. En esta arquitectura el DBMS se encuentra en una capa intermedia entre los usuarios de la base de datos que pueden ser clientes o administradores, y el sistema operativo. Los clientes dependen del DBMS para leer, escribir, o modificar datos, y los administradores dependen del DBMS para modificar, agregar o eliminar los metadatos y realizar cualquier tarea administrativa que se requiera. En la figura 4.1 se puede ver ejemplificada esta arquitectura.

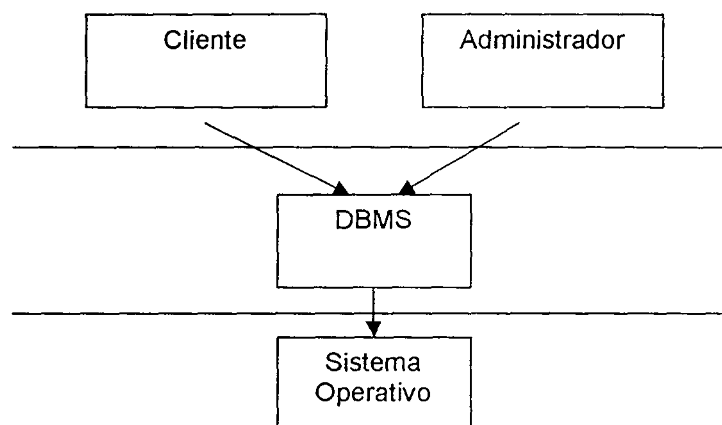


Figura 4.1 Arquitectura global del sistema MySQL

La parte principal de esta arquitectura global es la del DBMS, en ella se añadirán algunos módulos y relaciones que proporcionarán mejor desempeño y la funcionalidad distribuida que queremos demostrar en este trabajo.

4.3 Arquitectura propuesta.

Durante el desarrollo del capitulo anterior se analizaron diferentes manejadores de bases de datos como el Postgress, MySQL, InnoDB y algunos manejadores comerciales, se tomarán de ellos los módulos que se considera pueden implementar una mejor propuesta de arquitectura en el manejador MySQL para cubrir el aspecto de distribución en las bases de datos.

A continuación la figura 4.2, figura 4.3 y figura 4.4 describen algunos módulos adicionales que se consideran útiles para la mejorar el desempeño y la distribución de la información en el manejador MySQL, mismos que se incluyen en nuestra propuesta de arquitectura;

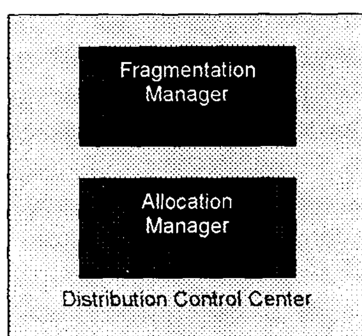


Figura 4.2 Distribution control center

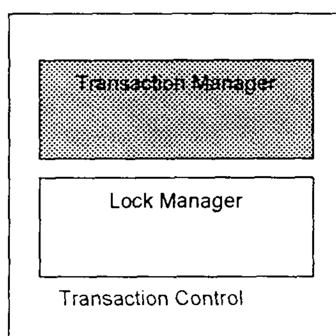


Figura 4.3 Transaction control

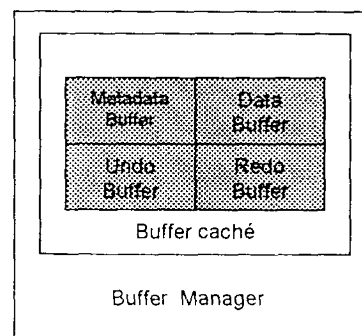


Figura 4.4 Buffer manager

El distribution control center se encarga controlar tanto la fragmentación de los datos como la localización de los mismos en la red. Es la parte medular de la distribución en la arquitectura propuesta en esta tesis. Este módulo contiene dos manejadores que son:

Fragmentation manager. Se encarga de mantener el esquema de fragmentación de las tablas mediante el establecimiento de sentencias SQL que definan la división de los datos. Establece comunicación con el Allocation manager para obtener la localización de los datos dado que ya conoce que se encuentran fragmentados.

Allocation manager. En este módulo se tiene la localización física de los fragmentos en los diferentes puntos de la red, se encarga de mantener un listado

de los sitios donde se puede acceder a la información requerida mediante las consultas distribuidas. Actúa en conjunto con el fragmentation manager del cual recibe solicitudes de la localización de los datos. Mediante la información obtenida de este módulo es posible generar el plan de ejecución para obtener la información solicitada en las consultas.

Transaction manager. El módulo de control de transacciones de MySQL solo cuenta con un lock manager encargado de administrar la concurrencia a los datos mediante un mecanismo de bloqueo, no incluye un subsistema que maneje la atomicidad y seriablidad en las transacciones, sugerimos para este fin, la adición de un transaction manager. La función de este subsistema radica en la manipulación atómica y serializada de las transacciones almacenando copias de la información modificada por las transacciones activas mediante el buffer manager y revertirlas en caso de ser necesario. Este subsistema optimizará el desempeño de la base de datos e incrementará la eficiencia del manejador en general, y liberará al programador de la responsabilidad de programar mecanismos de control de las transacciones.

El buffer manager actualmente cuenta con un solo subsistema, el caché de buffers, que nos sirve para la administración de la memoria, y proporciona una respuesta mas ágil del sistema. Sin embargo este módulo no realiza distinciones entre los diferentes tipos de buffers, como pueden ser los datos, metadatos, redo, y undo. Describiremos una subclasificación de buffers que sugerimos incluir en nuestra arquitectura para optimizar el manejo de la memoria en el manejador de base de datos distribuidas:

Metadata buffer. La función que realiza es manejar un buffer para los archivos de metadatos del diccionario de datos, al conservar los últimos datos consultados agiliza el flujo de la información en el manejador.

Data buffer. Este buffer se encarga de manejar un buffer de datos para las tablas e índices, y su función es la de conservar en memoria los datos consultados más recientemente, con el propósito de agilizar las consultas en el manejador.

Undo buffer. En beneficio del transaction manager, este buffer maneja copias de los datos que están siendo modificados al momento por las transacciones activas, así en caso de presentarse algún problema, simplemente se restaura la información almacenada en este buffer.

Redo buffer. Finalmente, la función que realiza este módulo es manejar un buffer de datos para los archivos de bitácora (log files).

El objetivo de tener varios buffers en el buffer manager es segregar los datos y agilizar las búsquedas de acuerdo a la naturaleza de los datos.

Estos módulos proporcionan funcionalidad y robustez extra al DBMS, pues cada uno ejecuta una función específica dentro de la arquitectura, en la sección 4.4 se describen los métodos que conforman la estructura de estos módulos y, que se utilizarán para generar los escenarios de trabajo en el capitulo cinco.

El diagrama de la figura 4.5 representa la arquitectura propuesta general del manejador de bases de datos distribuidas, incluyendo los módulos antes descritos y las dependencias que se consideran necesarias para el flujo de la información e interrelación de los módulos, en el capítulo siguiente se analizarán los escenarios que se podrían generar mediante las diversas operaciones transaccionales de una base de datos.

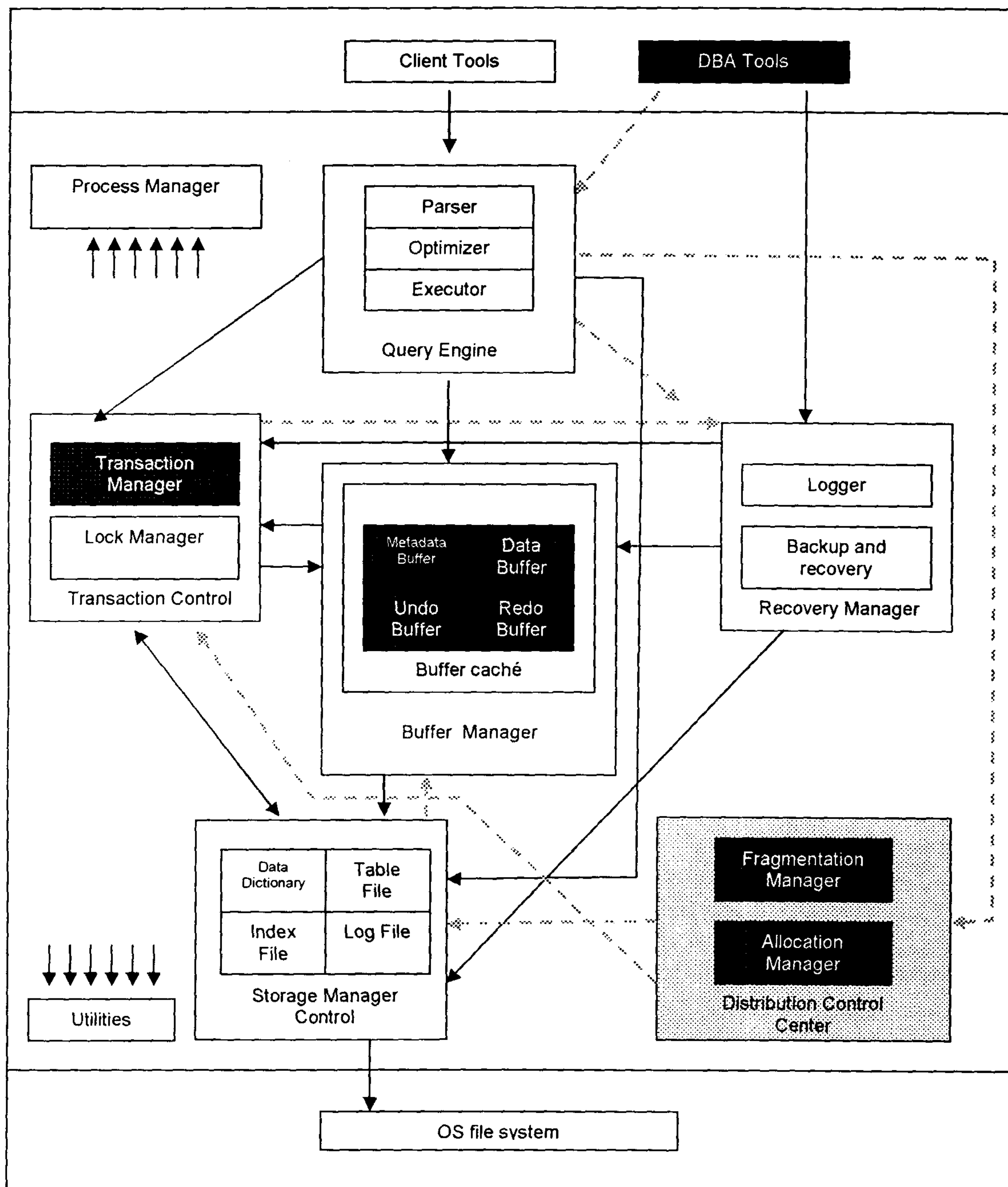


Figura 4.5 Arquitectura propuesta de un manejador de base de *datos distribuidas*

4.4 Métodos generales de la arquitectura propuesta.

Con el fin de ejemplificar el comportamiento y realizar un seguimiento del flujo de la información a través de la arquitectura distribuida propuesta, en las siguientes líneas se analizarán métodos importantes propios del DBMS incluyendo la funcionalidad distribuida que es la parte medular de este trabajo. Estos métodos serán utilizados más adelante en capitulo cinco al ejemplificar los principales escenarios operativos del manejador.

El DBMS consta de seis componentes principales, los cuales agrupan métodos que ejecutan tareas específicas y ofrecen funcionalidad a la arquitectura del DBMS, estos componentes son:

- Query Engine.
- Distribution Control Center.
- Buffer Manager.
- Transaction Control.
- Recovery Manager.
- Storage Manager Control.

A continuación se describe cada uno de ellos y los respectivos métodos que contienen.

Componente Query Engine. Este componente se encarga de recibir las peticiones de consulta que realizan los usuarios de la base de datos, analizar la consulta y verificar que tenga la sintaxis correcta, así como generar los planes de ejecución probables para optimizar el tiempo de respuesta del manejador utilizando el mejor plan de acción y ejecutarlo de acuerdo a la naturaleza de la transacción. Se encuentra relacionado con varios componentes, entre ellos el buffer manager, el transaction manager, y el distribution control center. La interacción entre estos componentes permite al query engine verificar la sintaxis de la consulta, generar los planes de acción y la ejecución de plan óptimo para la transacción. En la figura 4.6 se presenta la estructura del componente query engine, con los métodos que utiliza.

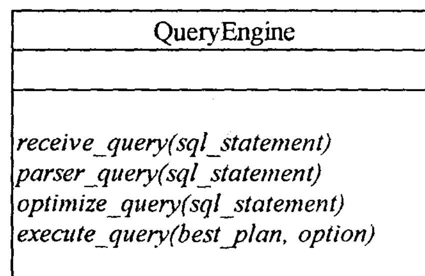


Figura 4.6 Estructura del componente Query Engine

Los métodos involucrados en este componente son los siguientes.

Método de recepción de la consulta. Este método se encarga de recibir la sentencia SQL y es el encargado de distribuirla al módulo correspondiente. Verifica que la sentencia sea válida, en caso de no serlo regresa el valor false. Interactuá directamente con el buffer manager y se encuentra localizada en el query engine.

```
public boolean receive_query(String sql_statement){
        if (sql_statement != nul)
                return true;
        else{
                System.out.print("Sentencia no válida");
                return false;
        }
} // end of the receive_query
```

Método de análisis sintáctico (parser). Este método realiza varias actividades de verificación, entre ellas están la verificación de la sintaxis de la sentencia, la correspondencia entre los metadatos, los índices y permisos de usuario. En caso de encontrar un error, regresa un valor false. Proporciona la seguridad de recibir una sentencia válida para realizar la operación requerida.

```
public boolean parser_query(String sql_statement){
CBufferManager BufMgr;
CStorageManager StoMgr;

        if (verify_sintaxis(sql_statement) &&
           BufMgr.verify_metadata(sql_statement) &&
           StoMgr.verify_index(sql_statement) &&
           StoMgr.verify_permisions(sql_statement))
                return true;
        else{
                System.out.print("Existe un error en la sentencia");
                return false;
        }
}// end of the parser_query
```

Método de optimización de consultas. Este método se encarga de realizar los planes de ejecución de la sentencia, previa verificación de los índices y la fragmentación de las tablas. Es fundamental para el buen desempeño de las consultas la generación de planes de ejecución óptimos. Mediante este método se generan planes de acción que optimizan el rendimiento del manejador, al realizar una transacción.

```
public String optimize_query(String sql_statement){
CStorageManager StoMgr;
CDistributionControl DistCtrl;
        if (StoMgr.index_verification(sql_statement) &&
           DistCtrl.verify_fragmentation(sql_statement)){
```

```
                String plan = generate_plan_execution(sql_statement);
                return plan;
        }
        else
                System.out.print("Existe un error en la sentencia");
}// end of the optimize_query
```

Método de ejecución de consultas. Este método se encarga de ejecutar el plan generado por el optimizador, independientemente de la operación solicitada en el query. Recibe como parámetros el plan de ejecución y la operación a realizar. Dependiendo de la operación realizará llamadas a diferentes funciones transaccionales, por ejemplo, lectura, inserción, modificación elminación, etc. Esencialmente es el que distribuye las tareas a los componentes de la arquitectura.

```
public boolean execute_query(String best_plan, int opcion){
CFunctionsQryEng FQryEng;
CTransactionControl TrCtrl;
CBufferManager BufMgr;
CRecoveryManager RecMgr;
CStorageManager StoMgr;

static int read_only = 1;
static int insert_query = 2;
static int update_query = 3;
static int delete_query = 4;
static int back_query = 5;
static int rec_query = 6;

        switch (opcion){
        read_only:
                if (StoMgr.dbuffer_mgr(sql_statement)){
                        FQryEng.execute_best_plan(best_plan);
                        return true;
                }
                else{
                        System.out.print("No se puede ejecutar la lectura");
                        return false;
                }
        break;

insert_query:
                if (TrCtrl.do_transaction(best_plan, opcion) &&
                   RecMgr.logger_mgr(opcion) &&
                   StoMgr.lf_mgr(best_plan,opcion) &&
                   StoMgr.tf_mgr(best_plan,,opcion))
                        return true;
                else{
                        System.out.print("No se puede ejecutar la inserción");
                        return false;
                }
        break;
```

```
        update_query:
                if (TrCtrl.do_transaction(best_plan,opcion) &&
                   RecMgr.logger_mgr(opcion) &&
                   StoMgr.lf_mgr(best_plan,opcion) &&
                   StoMgr.tf_mgr(best_plan,,opcion))
                        return true;
                else{
                        System.out.print("No se puede ejecutar la modificación");
                        return false;
                }
        break;
        delete_query:
                if (TrCtrl.do_transaction(best_plan,opcion) &&
                   RecMgr.logger_mgr(opcion) &&
                   StoMgr.lf_mgr(best_plan,opcion))
                        return true;
                else{
                        System.out.print("No se puede ejecutar la eliminación");
                        return false;
                }
        break;
        back_query:
                if (RecMgr.realize_backup(String sql_statement))
                        return true;
                else{
                        System.out.print("No se puede ejecutar el respaldo");
                        return false;
                }
        break;
        rec_query:
                if (RecMgr.realize_recovery(sql_statement))
                        return true;
                else{
                        System.out.print("No se puede ejecutar la recuperación");
                        return false;
                }
        break;
        } // end of the switch

}// end of the execute_query
```

Componente Distribution Control Center. Este componente se encarga controlar tanto la fragmentación de los datos como la localización de los mismos en la red, lo conforman dos métodos, uno que verifica la fragmentación de las tablas y uno que verifica la localización física de los fragmentos en la red. Interáctua directamente con el query engine, pues es el encargado de proporcionar la información necesaria para generar los planes de ejecución de las operaciones solicitadas al manejador. Este componente utiliza dos tablas de metadatos, las cuales contienen la información relevante para identificar y ubicar los fragmentos de tablas que se encuentran distribuidos en la red, mas adelante

se describirán estas tablas. En la figura 4.7 se describe la estructura del componente DCC.

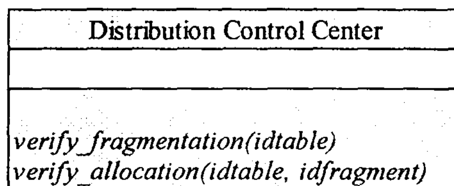


Figura 4.7 Estructura del componente Distribution Control Center.

A continuación describiremos los métodos del DCC.

Método de verificacion de la fragmentación de los datos. Este método tiene la responsabilidad de verificar si la tabla utilizada se encuentra fragmentada de alguna forma, recibe como parámetro el identificador de la tabla a consultar. Si encuentra que la tabla está fragmentada, debe de verificar la localización del fragmento, en caso contrario devuelve un valor nulo. Forma parte del la clase del DCC.

```
public String verify_fragmentation(String idtable){
statement SqlSt1;
Resultset ResultSQL1;

String sql_select = "select * from FragmentationTable where FragmentationTable.IDTable = " +
idtable;

ResultSQL1 = SqlSt1.executeQuery(sql_select);
        if (ResultSQL1.next()){
                if ResultSQL1.getString("Fragmented") = yes {
                        String local = verify_allocation(idtable,idfragment);
                        return local;
                }
                else{
                        System.out.print("No se encuentra fragmentada la tabla");
                        return null;
                }
        }
        else{
                System.out.print("No se encuentra la tabla");
                return null;
        }
}// end of the verify_fragmentation
```

Método de verificación de la localización de los datos. Este método se encarga de verificar la localización geográfica de los fragmentos, interactúa directamente con el método de verificación de la fragmentación, el cual le transfiere los parámetros identificadores de la tabla y el fragmento requerido. En caso de encontrarlo regresa la dirección ip del sitio donde se encuentra la

información, en caso contrario regresa un valor nulo. Forma parte de la clase del DCC.

```
public String verify_allocation(String idtable, int idfragment){
statement SqlSt1; Resultset ResultSQL1;
String sql_select = "select * from AllocationTable where AllocationTable.IDTable = " + idtable +
                    "and AllocationTable.IDFragment = " + idfragment;
ResultSQL1 = SqlSt1.executeQuery(sql_select);
        if (ResultSQL1.next()){
                String localizacion = ResultSQL1.getString("IDSitio");
                return localización;
        }
        else {
                System.out.print("No se encuentra la tabla");
                return null;
        }
}//end of the verify_allocation
```

Componente Buffer Manager. Este componente es el encargado de administrar la memoria del manejador, permite una recuperación agilizada de los datos al consultarlos directamente espacios de memoria específicos donde se almacenan los datos que se han consultado con anterioridad , evitando realizar peticiones innecesarias al storage manager. En caso de no contar con los datos en memoria, realiza una consulta al storage manager para acceder a los datos solicitados. Interactúa con todos los componentes de la arquitectura, es parte fundamental para el funcionamiento del DBMS. Mantiene 4 tipos de buffers, metadatos, datos, undo, redo. En la figura 4.8 se observa la estructura de este componente.

Buffer Manager
mdbuffer_mgr(information) *dbuffer_mgr(data)* *ubuffer_mgr(undodata)* *rbuffer_mgr(redodata)*

Figura 4.8 Estructura del componente Buffer Manager.

A continuación describiremos los métodos del Buffer Manager.

Método de administración del metadata buffer. Este método se encarga de verificar si la información de los metadatos a consultar se encuentra en el metadata buffer, en caso de estar disponible, la utiliza, en caso contrario, interactúa con el storage manager para obtenerla. Agiliza la obtención de metadatos, al consultar solamente el área de memoria reservada para este tipo de datos.

```
public boolean mdbuffer_mgr(String information){
CFunctionsBufferMgr FBufferMgr;
CStorageManager StoMgr;
	if (information){
		FBufferMgr.utilize_information(information);
	}
	else {
		StoMgr.find_storage_information(information);
	}
	return true;
} // end of the mdbuffer_mgr
```

Método de administracón del data buffer. Este método se encarga de verificar si la información de los datos a consultar se encuentra en el data buffer, en caso de estar disponible, la utiliza, en caso contrario, interactúa con el storage manager para obtenerla. Agiliza la obtención de datos, al consultar solamente el área de memoria reservada para este tipo de datos.

```
public boolean dbuffer_mgr(String data){
CFunctionsBufferMgr FBufferMgr;
CStorageManager StoMgr;
	if (data)
		FBufferMgr.utilize_data(data);
	else
		StoMgr.find_data_storage(data);
	return true;
}//end of the dbuffer_mgr
```

Método de administración del undo buffer. Este método se encarga de verificar si la información de los datos a consultar se encuentra en el undo buffer, en caso de estar disponible, la utiliza, en caso contrario, no existen transacciones previas que requieran de alguna recuperación. Se utiliza como memoria de respaldo para datos que están siendo modificados por alguna transacción.

```
public boolean ubuffer_mgr(String undodata){
CFunctionsBufferMgr FBufferMgr;
	if (undodata)
		FBufferMgr.undo_transaction(undodata);
	else
		System.out.print("No existen transacciones por deshacer");
	return true;
}// end of the ubuffer_mgr
```

Método de administración del redo buffer. Este método se encarga de verificar si la información de los datos a consultar se encuentra en el redo buffer, en caso de estar disponible, la utiliza, en caso contrario, no existen transacciones que requieran realizarse de nuevo. Facilita el rastreo de las transacciones que se han realizado en la base de datos, agilizando la recuperación en caso de requerirse.

```
public boolean rbuffer_mgr(String redodata){
CFunctionsBufferMgr FBufferMgr;
        if (redodata)
                FBufferMgr.redo_transaction(redodata);
        else
                System.out.print("No existen transacciones por rehacer");
        return true;
}// end of the rbuffer_mgr
```

Componente Transaction Control. Este componente se encarga de realizar las diversas transacciones que proporciona un DBMS y de realizar los bloqueos para garantizar la consistencia de los datos en a base de datos. Interactúa con el query engine, el storage manager, el buffer manager y el recovery manager. Es también de gran importancia para el DBMS ya que es el componente encargado de realizar las transacciones generadas por las consultas de los usuarios. A continuación se observa en la figura 4.9 la estructura de este componente y los métodos que lo conforman.

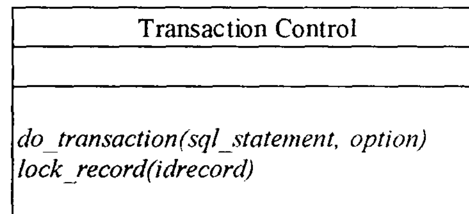


Figura 4.9 Estructura del componente Transaction Control.

A continuación se definen los métodos de este componente.

Método de realización de transacciones. Este método se encarga de ejecutar las diversas operaciones transaccionales de la base de datos. Antes de ejecutar la transacción, verifica si existe alguna violación a las restricciones de la base de datos, en caso de ocurrir, regresa un valor false, en caso contrario, efectúa la transacción solicitada.

```
public boolean do_transaction(String sql_statement, int opcion);{
CFunctionsStorageMgr FStoMgr;
CFunctionsTrCtrl FTrCtl;
        if (FStoMgr.verify_md_restrictions(sql_statement,opcion))
                FTrCtrl.realize_transaction(sql_statement, opcion);
        else{
                System.out.print("Existe una violación a las restricciones de la base de datos");
                boolean transaccion_status = false;
        }
        if (transaction_status){
                System.out.print("La transacción ha sido realizada con éxito");
                return true;
        }
```

```
        else {
                System.out.print("No se pudo realizar la transacción");
                return false;
        }
}//end of the do_transaction
```

Método de bloqueo de registros. Se encarga de establecer los bloqueos a los registros que se van a utilizar de la base de datos a solicitud del transaction manager. En caso de no existir el registro envía un mensaje de error. Previene la modificación simultánea de un mismo registro de la base de datos.

```
public boolean lock_record(String record){
        if (record){
                String record_status = lock;
                return true;
        }
        else{
                System.out.print("No existe el registro");
                return false;
        }
}//end of the lock_record
```

Componente Recovery Manager. Este componente se encarga de la recuperación y respaldo de la información que se encuentra en la base de datos así como de realizar una bitácora de todas las transacciones que se realizan en ella. Contiene métodos que ejecutan estas actividades de seguridad, de realizar el historial de las transacciones, la recuperación de archivos dañados y el respaldo de un archivo o toda la base de datos. Interactúa con los componentes query engine, storage manager transaction manager y el buffer manager. A continuación, en la figura 4.10 se observa la estructura de este componente y los métodos que lo conforman.

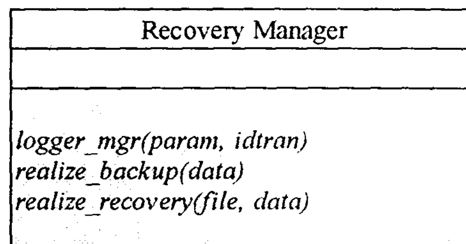


Figura 4.10 Estructura del componente Recovery Manager.

Los métodos que forman parte de este componente son los siguientes.

Método de administración de la bitácora. Se encarga de registrar y recuperar las transacciones en la bitácora del sistema. Es el histórico de las transacciones del sistema. Recibe un parámetro de escritura o lectura, en caso de ser escritura,

registra los movimientos en la bitácora, en caso de ser lectura, recupera la información solicitada. Almacena el historial de las transacciones de la base de datos y asegura la recuperación del sistema en caso de una caída del mismo.

```
public boolean logger_mgr(String param,String idtran){
CFunctionsRecMgr FRecMgr;

        if (param=="write")
                FRecMgr.write_transaction(param,idtran);
        else
                if (param=="read")
                        FRecMgr.read_transaction(param,idtran);
        return true;
}//end of the logger_mgr
```

Método de backup de la base de datos. Realiza un respaldo de la información solicitada y la guarda en un sitio específico de almacenamiento. Interactúa con el storage manager para realizar copias de la información solicitada.

```
public boolean realize_backup(String data){
CFunctionsRecMgr FRecMgr;
        FRecMgr.backup_process(data);
        return true;
}//end of the realize_backup
```

Método de recuperación de la base de datos. Método que realiza la recuperación de información en la base de datos, partiendo de un archivo de datos previo. Mediante este método se reconstruye información de la base de datos para llevarla de un estado inconsistente a uno consistente. Realza consultas a las bitácoras del sistema para restaurar la información.

```
public boolean realize_recovery(String file,String data){
CFunctionsRecMgr FRecMgr;
        FRecMgr.get_previous_datafile(file);
        FrecMgr.do_recovery(data);
        return true;
}// end of the realize_recovery
```

Componente Storage Manager Control. Este componente se encarga de almacenar y administrar físicamente los datos de la base da datos. Contiene archivos del diccionario de datos, datos operativos, índices y bitácora. Mantiene relación con el buffer manager, el transaction manager, el recovery manager y el distribution control center. Proporciona información y datos contenidos en la base de datos. Es la memoria permanente del sistema. En la figura 4.11 se observa la estructura de este componente y los métodos que lo conforman.

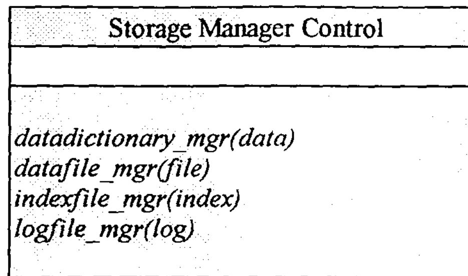


Figura 4.11 Estructura del componente Storage Manager Control.

A continuación se describen los métodos de este componente.

Método de administración del data dictionary. Se encarga de administrar el diccionario de datos del sistema. Incluye todas las operaciones posibles realizables a esta información. Es el manejador de la información de las tablas del sistema.

```
public boolean datadictionary_mgr(String data){
CFunctionsStoMgr FStoMgr;
        FStoMgr.manage_data(data);
        return true;
}//end of the datadictionary_mgr
```

Método de administración del data file. Se encarga de administrar los archivos de datos del sistema. Es el manejador de las tablas que contienen la información útil para la organización.

```
public boolean datafile_mgr(String file){
CFunctionsStoMgr FStoMgr;
        StoMgr.manage_datafiles(file);  //Rutina que administra el index file
        return true;
}//end of the datafile_mgr
```

Método que administra el index file. Se encarga de administrar los índices del sistema. Mantiene estrecha relación con el manejador de metadatos y en general con los métodos del storage manager.

```
public boolean indexfile_mgr(Strinf index){
CFunctionsStoMgr FStoMgr;
        FStoMgr.manage_indexfiles(index);
        return true;
}//end of the indexfile_mgr
```

Método que administra el log file. Se encarga de administrar la bitácora del storage manager. Registra los cambios realizados en las archivos del sistema.

```
public boolean logfile_mgr(String log){
CFunctionsStoMgr FStoMgr;
        FStoMgr.manage_logfile(log);
        return true;
}//end of the logfile_mgr
```

4.5 Tablas de referencia para el Distribution Control Center (DCC).

El DCC utiliza información propia de la base de datos para realizar las verificaciones tanto de fragmentación como de localización. La tabla de fragmentación permite almacenar los datos de las tablas fragmentadas como son el identificador de la tabla, el identificador del fragmento, el tipo de fragmentación, ya sea horizontal, vertical o híbrida, la sentencia sql responsable de la fragmentación y un campo de verificación de replicación del fragmento. A su vez la tabla de localización permite conocer la ubicación exacta del fragmento en la red al contener el identificador del fragmento, identificador de la localización, el identificador de la tabla, el nodo que contiene el fragmento (nombre del host), su dirección lógica (ip) y su dirección física (localización geográfica). Estas tablas se encuentran relacionadas mediante la llave primaria, la cual se encuentra conformada en la tabla de fragmentación por el identificador el fragmento y el identificador de la tabla, y en la tabla de localización por los mismos mas el identificador de localización.

A continuación de definen los valores que deben de representar claramente la configuración distribuida de las tablas de la base de datos. Los principales valores identificados son:

Atributos de la tabla de fragmentación (Fragmentation table).

Atributo	Descripción	Tipo
IDFragment	Identificador del fragmento	Numérico
TableName	Identificador de la tabla fragmentada	Caracter
Fragmented	Valor lógico que indica si esta fragmentada o no	Lógico
FragmentationType	IDentificador del tipo de fragmentación	Caracter
SQLFragment	Sentencia SQL de fragmentación	Caracter
Replicated	Indicador de existencia de replicación del fragmento	Lógico

Tabla 4.1 Atributos de la tabla de fragmentación

Atributos de la tabla de localización (Allocation table).

Atributo	Descripción	Tipo
IDFragment	Identificador del fragmento	Numérico
TableName	Identificador de la tabla fragmentada	Caracter
IDAllocation	Identificador de la localización del fragmento	Numérico
IDSite	Identificador del sitio (nombre del nodo)	Caracter
IPAddress	Dirección lógica del sitio donde se encuentra el fragmento	Caracter
PhysicalLocation	Nombre de la localización física del sitio donde se encuentra el fragmento	Caracter

Tabla 4.2 Atributos de la tabla de localización

A continuación se observan ejemplos de valores posibles para los atributos de las tablas de fragmentación y localización.

Supongamos que el administrador de la base de datos desea establecer en la oficina número uno y dos la información relacionada con los clientes de sexo masculino, y en la oficina número 4 la información de los clientes femeninos. Para ello debe de representar esa información de la siguiente manera, tomando como base los atributos explicados anteriormente:

IDFragment	Table Name	Fragmented	Fragment Type	SQL Fragment	Replica
01	Customer	Yes	H	Select * from customer where customer.sex="M"	Yes
02	Customer	Yes	H	Select * from customer where customer.sex<>"M"	Yes

Tabla 4.3 Ejemplo de datos en la tabla de fragmentación

IDFragment	TableName	IDAllocation	IDSite	IPAdress	PhysicalLocation
01	Customer	01	Site1	192.168.10.2	Office1
01	Customer	02	Site2	192.168.10.5	Office2
02	Customer	03	Site3	192.168.10.7	Office4

Tabla 4.4 Ejemplo de datos en la tabla de localización

De esta manera, un usuario una realizar una consulta a la tabla customer, con información relacionada con algún cliente femenino, se conoce que esa tabla se encuentra fragmentada y por supuesto el sitio a donde el manejador debe de dirigirse para obtener la información requerida.

4.6 Adaptabilidad y portabilidad de los módulos.

Debido a la naturaleza intrínseca actual de los manejadores de bases de datos, la tarea de implementar nuevos módulos que interactúen integralmente con los módulos existentes se torna muy complicada y difícil. Debido a esta causa, resulta atractivo considerar la alternativa de extender un DBMS, de tal forma que, permita adicionar o reemplazar funcionalidad modular tanto como sea demandada. Para ello se requiere de una arquitectura claramente definida representada en el DBMS.

Esta arquitectura define los puntos en el sistema donde las implementaciones pueden ser adicionadas. En general estas adecuaciones, deben ser específicas para ciertos puntos del sistema, los efectos y modificaciones en otras partes del mismo deben de ser evitadas o al menos minimizadas, siempre que sea posible.

Los componentes del manejador deben de ser realizados de tal forma que permitan una adaptación flexible al sistema. Los componentes podrían ser entonces proporcionados por fuentes externas o ingenieros locales, incrementado de esta forma el desarrollo base de los DBMS.

Es difícil que un solo DBMS agrupe todas las aplicaciones emergentes y demandantes del mercado, por lo tanto, una solución factible es la de extender el DBMS en base a la demanda, esto es, proporcionar lo estándar, la funcionalidad comúnmente utilizada y añadir, poco a poco, características más avanzadas. En otras palabras, en vez de obtener un DBMS monolítico, podemos utilizar el concepto de un DBMS a la carta [KDIT].

Cuando hablamos de adaptaciones a un DBMS nos viene a la mente la reutilización, extensibilidad, libertad de código, e interoperatividad. La utilización de componentes es una forma disciplinada de de construir bloques con propiedades específicas. Definiendo más formalmente, un componente es un artefacto de software que modela e implementa un conjunto coherente y bien definido de funciones. Cada componente adiciona características extras al DBMS proporcionando mayor alcance y funcionalidad. La portabilidad de los componentes permite su utilización independientemente del ambiente en el cual se implementen, en nuestro caso, los componentes se encuentran modelados en java, lenguaje que les otorga esta característica.

4.6.1 Extensiones en DBMS´s

Un problema general en escenarios de base de datos es la estructura monolítica del DBMS tradicional. La estructura monolítica o arquitectura monolítica, se refiere a que el DBMS es una unidad simple cuyas partes son conectadas a otra y es dependiente de otra a tal grado que las modificaciones y extensiones no son posibles llevarlas a cabo fácilmente.

Los DBMS monolíticos en ocasiones deben cumplir nuevos requerimientos, aparentemente tienen que ser extendidos para incluir nueva funcionalidad, sin embargo, mejorar un sistema simple con módulos, implementando todas las funciones nuevas, no es un enfoque viable por varias razones:

- Los DBMS podrían llegar a ser mas grandes y, en consecuencia, mas compleja que no podrían ser mantenidos a un costo razonable.
- Los usuarios tendrían que pagar un alto precio para adicionar funcionalidad, aun si ellos no necesitan cada parte de esto.
- Un proveedor de DBMS podría no tener la experiencia para realizar tales extensiones y podrían no tener los recursos para comprometer todas las extensiones en un periodo razonable.

Una opción para esto es considerar la alternativa de extender el DBMS de manera fragmentada, es decir, adicionando funcionalidad (o reemplazando) de una forma modular, tanto como sea necesario.

Dado que las extensiones modulares a un DBMS afectan su arquitectura, requieren que ciertos prerrequisitos sean cumplidos, en otras palabras se requiere que una arquitectura de software bien definida sea impuesta al DBMS. Esta arquitectura debe definir claramente los lugares en el sistema donde las extensiones sean posibles. En general, las extensiones deberían ser colocadas en algunos lugares bien definidos en el sistema, y los efectos y modificaciones sobre otras partes deberían ser evitadas o al menos minimizadas.

Algo deseable en las arquitecturas de los DBMS´s es definirlas en fragmentos (componentes) de tal manera que nuevos componentes puedan ser adicionados ó componentes existentes puedan ser intercambiados de una manera flexible. Así, la arquitectura de componentes especifica y restringe las maneras en las que un DBMS debe ser personalizado. A fin de cuentas, la arquitectura también define la noción de componente, así como el dominio de extensiones válidas a un DBMS [KDIT].

4.7 Distribución de los componentes.

Dado que entre las finalidades que se persiguen mediante la implementación de las bases de datos distribuidas se encuentran la redundancia de funcionalidad y disponibilidad de datos, el establecimiento de mecanismos de comunicación entre los componentes que conforman la arquitectura del DBMS, permite establecer las relaciones entre ellos, de tal forma que, al implementar varios sitios con un DBMS con la arquitectura propuesta, se mantenga una coordinación entre los componentes de los mismos, o distribuir los componentes de un manejador a través de varios sitios de la red.

Utilizando mensajes entre componentes se puede interactuar y optimizar el desempeño del manejador al realizar las operaciones en forma distribuida. Existen diversas maneras de comunicar los componentes. Entre las más comunes se encuentra el paso de objetos o instancias las cuales se simulan la ejecución local de un componente independientemente de su localización. La figura 4.12 demuestra la coordinación entre dos componentes del manejador que se encuentran en sitios geográficamente distintos. La información que intercambian permite ejecutar una transacción en un entorno distribuido.

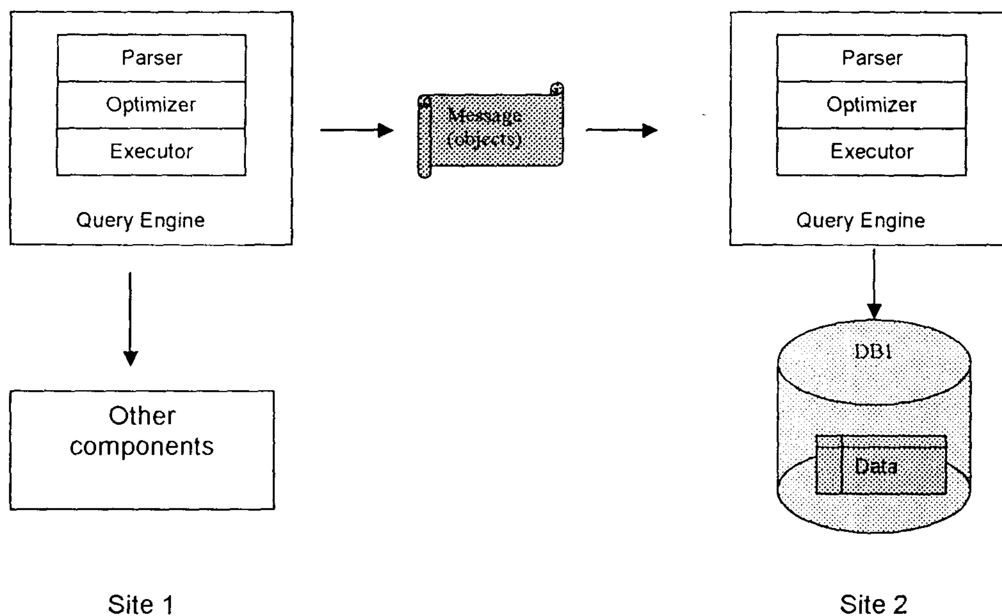


Figura 4.12 Comunicación entre componentes en un ambiente distribuido

La definición de la arquitectura del DBMS basada en componentes de software permite establecer diferentes configuraciones del manejador dependiendo de los requerimientos del usuario u organización, pudiendo manejarse tanto de forma centralizada como de manera distribuida.

4.8 Conclusiones.

La arquitectura propuesta en este capítulo contiene algunos componentes que mejoran el desempeño de un DBMS, tomando como referencias los DBMS´s del capítulo tres. Los principales componentes son el Distribution Control Center, el Transaction Manager, y el Buffer Manager. Mediante métodos se describe la funcionalidad de cada uno de ellos utilizando el lenguaje de programación java. Se hace énfasis en el DCC y se refieren algunos ejemplos de datos que demuestran la lógica del componente, así como la descripción de las tablas que utiliza.

La utilización de componentes utilizando el lenguaje java, permite conservar las características de un software libre de portabilidad, adaptabilidad y encapsulamiento, proporciona la capacidad de ir incrementando funcionalidad a un DBMS según las necesidades de la organización que lo utiliza y permite mejorar el costo del producto. En el capitulo cinco se observa como se desempeñan los componentes aquí descritos.

Capitulo 5 Validación de la arquitectura mediante casos de estudio.

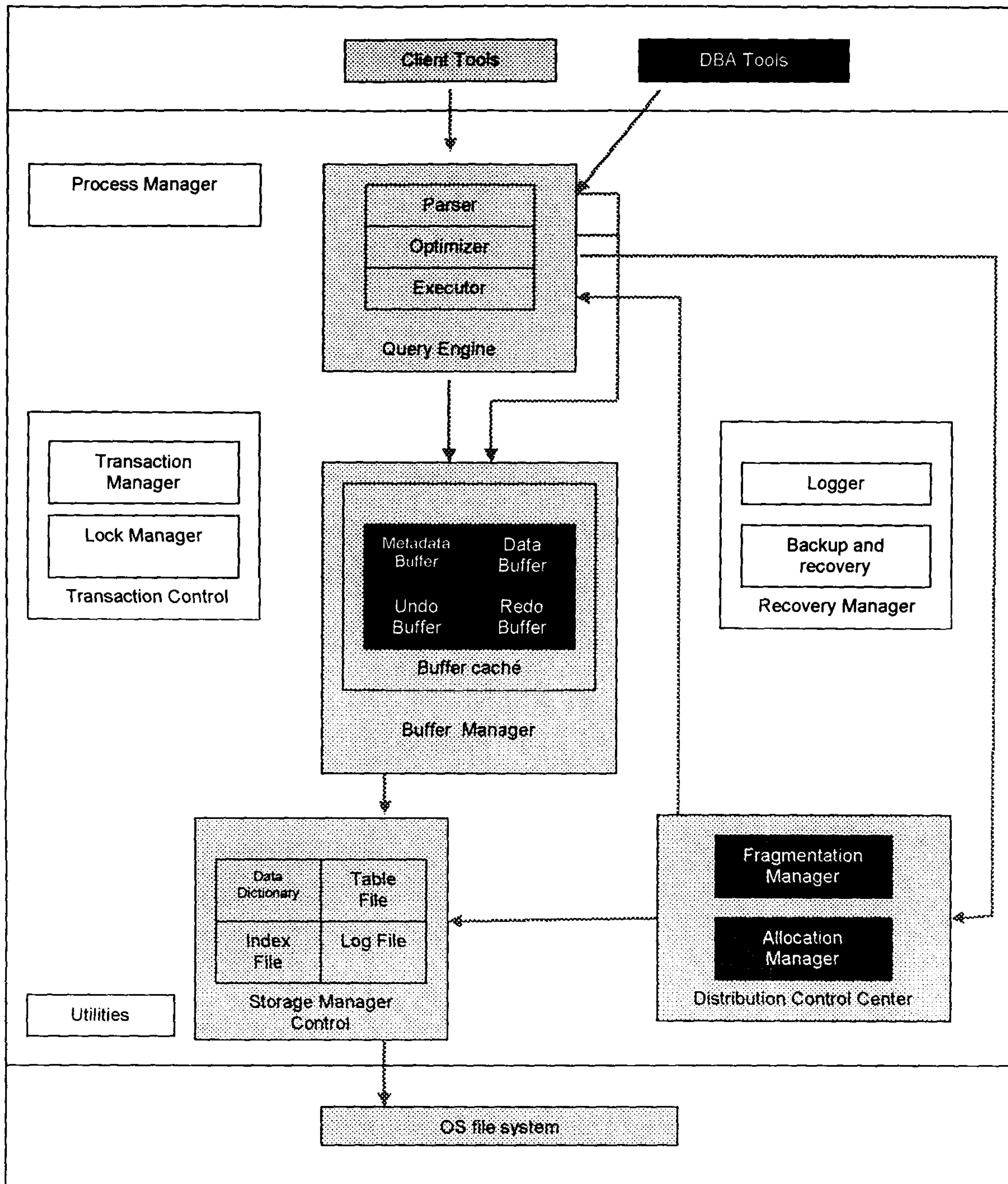
En este capítulo se analizan los casos de estudio con las transacciones más representativas de un DBMS distribuido. Estos escenarios se modelan mediante diagramas de componentes donde se observa el flujo de la información y se describe también, mediante código java, el comportamiento y operaciones lógicas de cada uno de ellos tomando como base los métodos descritos en el capítulo cuatro. Se consideran los componentes propuestos del capitulo cuatro como son el Distribution Control Center, el Lock Manager y el Buffer Manager. Los casos de estudio que se analizan son la consulta, inserción, modificación eliminación, respaldo y recuperación de un registro en la base de datos distribuida. A continuación se analizan los casos de estudio.

5.1 Escenario 1 Consulta de un registro a la base de datos (read-only).

Veamos que sucede cuando tenemos una consulta a un registro de la base de datos distribuida, en la figura 5.1 se identifican los módulos del manejador que intervienen en el proceso, desde la petición del usuario hasta la recuperación del registro solicitado, así como el flujo de la información a través de cada módulo del sistema.

- Se identifican dos posibles fuentes generadoras de consulta que son el usuario y el DBA
- La consulta se le transfiere al parser que se encarga de verificar la sintaxis y asegura junto con el buffer manager la existencia de los valores tabla-columna en los metadatos, así como los permisos de acceso correspondientes.
- Una vez construido el árbol de compilación a partir de la consulta, éste es transferido al optimizer, el cual verifica la disponibilidad y correspondencia de índices. Este módulo tiene interacción directa con el distribution control center, pues verifica mediante el fragmentation manager si los datos a consultar se encuentran en fragmentos de tabla así como la localización de ellos interactuando con el allocation manager.
- El allocation manager transfiere el flujo de nuevo hacia el optimizer, el cual, con la información obtenida, formula planes de ejecución probables eligiendo al final, el más apropiado.
- El executor se encarga de ejecutar el plan, lee la información que satisface la consulta en el data buffer y realiza las operaciones relacionales algebraicas apropiadas.

- Si el buffer manager no encuentra los datos requeridos en sus buffers, se verifica el data file respectivo mediante el storage manager, se obtienen los datos y se almacenan en el buffer.
- Si de alguna manera la información esta siendo modificada por algún otro usuario, se verifica el undo buffer para obtener la versión del dato previa a la modificación.



Componentes involucrados
Componentes existentes
Componentes adicionales
Flujo de la información

Figura 5.1 Escenario de consulta de un registro mediante un DDBMS

Método de consulta de un registro a la base de datos (read-only)

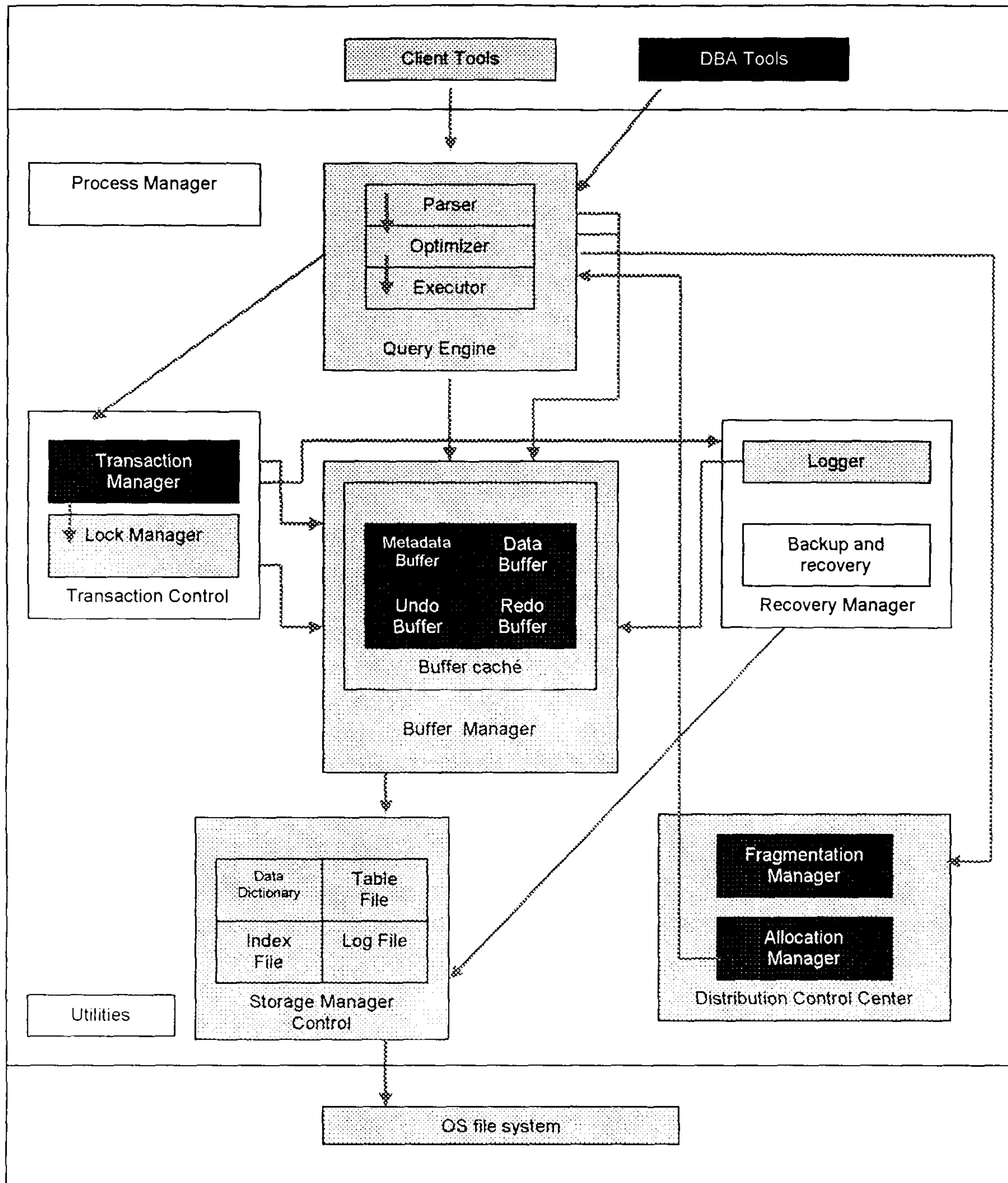
Realiza las operaciones necesarias para acceder a la base de datos distribuida y devolver los resultados obtenidos al usuario

```
public boolean read_only_request_scenario(String sql_statement, int opcion_read_only){

CVisualFunctions FVisual;
boolean query_string, evalpq, evalop, evalex;
query_string = receive_query(sql_statement);

if (query_string){
	avalpq = parser_query(sql_statement);
	if (evalpq){
		evalop=optimize_query(sql_statement);
		if (evalop){
			evalex = execute_query(best_plan, opcion_read_only);
			if (evalex){
				FVisual.show_user_results(results);
				System.out.print("La consulta ha sido realizado con éxito");
			}
			else
				System.out.print("No se pudo ejecutar la consulta");
	}
		else
			System.out.print("No se pudo optimizar la consulta");
	}
	else
		System.out.print("No se pudo analizar la consulta");
}
else
	System.out.print("La consulta no es correcta");
if (evalex)			//Regresa un valor verdadero o falso
	return true;			//dependiendo del resultado de la evaluación
else
	return false;
}//end of the read_only_request_scenario
```

5.2 Escenario 2 Escritura de un registro a la base de datos (Insert).

Un escenario muy ilustrativo en los manejadores de bases de datos es la inserción de registros, en nuestro caso se observa como interactúa el distribution control center con los módulos del manejador para realizar la transacción en un ambiente distribuido. La figura 5.2 representa la escritura de un registro en la base de datos.

- Se observan dos fuentes generadoras de la transacción, el usuario y el DBA.
- El registro a insertar se le transfiere al parser que se encarga de verificar la sintaxis y asegura junto con el buffer manager la existencia de los valores tabla-columna en los metadatos donde se va a insertar el registro, así como los permisos de acceso correspondientes.
- Una vez construido el árbol de compilación a partir de la petición de escritura, éste es transferido al optimizer, el cual verifica la disponibilidad y correspondencia de índices.
- El optimizer se comunica con el distribution control center para verificar si el registro se va a insertar en una tabla fragmentada así como la localización del fragmento en la red.
- El allocation manager devuelve la información al optimizer que se encarga de generar planes de ejecución probables para la transacción, eligiendo al final el más adecuado.
- El executor se encarga de ejecutar el plan, solicitando al transaction manager la realización de la transacción de inserción.
- Antes de insertar el registro, el transaction manager realiza ciertas operaciones para garantizar la realización de la transacción. Verifica si la transacción viola alguna restricción de la base de datos definida en los metadatos, el redo buffer se mantiene nulo, inserta la información en el data buffer, informa al logger de la transacción que se lleva a cabo para que se registre, el logger a su vez almacena esta información en el redo buffer, y finalmente se registra la transacción en el log file del storage manager.
- Se realiza la inserción del registro en el storage manager, pasando la información del data buffer al table file.



Componentes involucrados
Componentes existentes
Componentes adicionales
Flujo de la información

Figura 5.2 Escenario de inserción de un registro mediante un DDBMS

Método de inserción de un registro a la base de datos

```
public boolean insert_request_scenario(sql_statement, opcion_insert){
CVisualFunctions FVisual;
boolean query_string, evalpq, evalop, evalex;

query_string = receive_query(sql_statement);
if (query_string){
	evalpq = parser_query(sql_statement);
	if (evalpq){
		evalop=optimize_query(sql_statement);
		if (evalop){
			evalex=execute_query(best_plan, opcion_insert);
			if (evalex){
				FVisual.show_user_results(results);
				System.out.print("La inserción ha sido realizada con éxito");
			}
			else
				System.out.print("No se pudo insertar el registro");
		}
		else
			System.out.print("No se pudo optimizar la sentencia");
	}
	else
		System.out.print("No se pudo analizar la sentencia");
}
else
	System.out.print("La sentencia no es correcta");

if (evalex)			//Regresa un valor verdadero o falso
	return true;			//dependiendo del resultado de la evaluación
else
	return false;

}// end of the insert_request_scenario
```

5.3 Escenario 3 Modificación de un registro a la base de datos (Update).

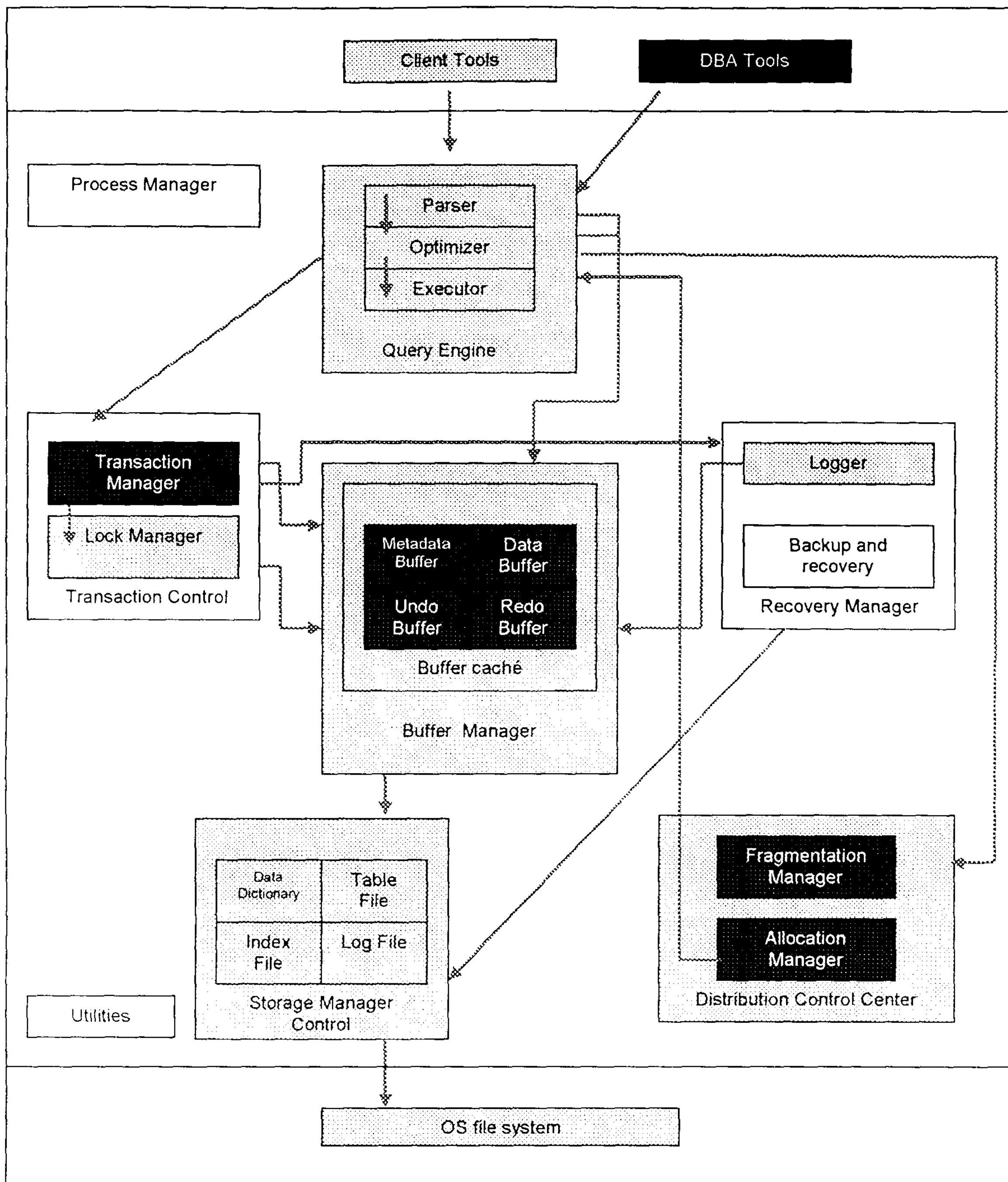
Así como la inserción de nuevos registros, se necesita también la modificación o actualización de los datos. El escenario que se analiza a continuación es la modificación (update) de registros en la base de datos distribuida, se observa como interactúa el distribution control center con los módulos del manejador, para realizar la transacción en un ambiente distribuido. La figura 5.3 representa todo el proceso de modificación o actualización paso a paso.

- Se tienen dos posible fuentes generadoras de la transacción, el usuario y el DBA, uno de ellos genera una sentencia solicitando la actualización de datos.
- La sentencia se le transfiere al parser que se encarga de verificar la sintaxis y asegura junto con el buffer manager la existencia de los valores tabla-columna en los metadatos donde se va a insertar el registro, así como los permisos de acceso correspondientes.
- Una vez construido el árbol de compilación a partir de la petición de escritura, éste es transferido al optimizer, el cual verifica la disponibilidad y correspondencia de índices.
- El optimizer se comunica con el distribution control center para verificar si el registro a modificar se encuentra en una tabla fragmentada así como la localización del fragmento en la red.
- El allocation manager devuelve la información al optimizer que se encarga de generar planes de ejecución probables para la transacción, eligiendo al final el más adecuado.
- El executor se encarga de ejecutar el plan, solicitando al transaction manager la realización de la transacción de actualización.

Antes de modificar el registro, el transaction manager realiza ciertas operaciones para garantizar la realización de la transacción.

- Verifica si la transacción viola alguna restricción de la base de datos definida en los metadatos.
- Solicita al lock manager candados para los registros que serán modificados.
- Mantiene una copia de los datos en el undo buffer y se modifica la información en el data buffer.
- Informa al logger de la transacción que se lleva a cabo para que se registre.
- El logger a su vez almacena esta información en el redo buffer, y finalmente se registra la transacción en el log file del storage manager.

    - El inicio y el final de la transacción se registran finalmente en el log file.
- Si los datos a ser modificados no se encuentran en el data buffer, el buffer manager obtiene los datos del storage manager, los coloca en el buffer y realiza la modificación. Sin embargo, si los datos a ser modificados están siendo utilizados por otra transacción de algún otro usuario, la transacción de modificación debe de abortar o en su defecto esperar a que la transacción en curso finalice, para empezar a ejecutarse.



Componentes involucrados
Componentes existentes
Componentes adicionales
Flujo de la información

Figura 5.3 Escenario de modificación de un registro mediante un DDBMS

Método de modificación de un registro a la base de datos

```
public boolean update_request_scenario(sql_statement, opcion_update){
CVisualFunctions FVisual;
boolean query_string, evalpq, evalop, evalex;

query_string = receive_query(sql_statement);
if (query_string){
	evalpq = parser_query(sql_statement);
	if (evalpq){
		evalop=optimize_query(sql_statement);
		if (evalop){
			evalex=execute_query(best_plan, opcion_update);
			if (evalex){
				FVisual.show_user_results(results);
				System.out.print("La modificación ha sido realizada con éxito");
			end.
			else
				System.out.print("No se pudo modificar el registro");
		}
		else
			System.out.print("No se pudo optimizar la sentencia");
	}
	else
		System.out.print("No se pudo analizar la sentencia");
}
else
	System.out.print("La sentencia no es correcta");

if (evalex)			//Regresa un valor verdadero o falso
	return true;			//dependiendo del resultado de la evaluación
else
	return false;

}//end of the update_request_scenario
```

5.4 Escenario 4 Eliminación de un registro en la base de datos (Delete).

La eliminación de un registro es parte de las operaciones comunes de un manejador de base de datos, en nuestro caso, la distribución involucra algunas nuevas dependencias que permiten ubicar, y dar de baja del sistema la información en cuestión que se encuentra en alguna parte de la red. En la figura 5.4 se tiene el escenario que ejemplifica la secuencia de pasos que desempeña el manejador para realizar la transacción de eliminación.

- Como el escenario anterior, se tienen dos posible fuentes generadoras de la transacción, el usuario y el DBA, uno de ellos genera una sentencia solicitando la eliminación de datos.
- La sentencia se le transfiere al parser que se encarga de verificar la sintaxis y asegura junto con el buffer manager la existencia de los valores tabla-columna en los metadatos donde se va a eliminar el registro, así como los permisos de acceso correspondientes.
- Una vez construido el árbol de compilación a partir de la petición de eliminación, éste es transferido al optimizer, el cual verifica la disponibilidad y correspondencia de índices.
- El optimizer se comunica con el distribution control center para verificar si el registro a eliminar se encuentra en una tabla fragmentada así como la localización del fragmento en la red.
- El allocation manager devuelve la información al optimizer que se encarga de generar planes de ejecución probables para la transacción, eligiendo al final el más adecuado.
- El executor se encarga de ejecutar el plan, solicitando al transaction manager la realización de la transacción de eliminación.

Para eliminar el registro, el transaction manager realiza ciertas operaciones para garantizar la realización de la transacción.

- Verifica si la transacción viola alguna restricción de la base de datos definida en los metadatos.
- Solicita al lock manager candados para los registros que serán eliminados.
- Mantiene una copia de los datos en el undo buffer y se modifica la información en el data buffer.
- Informa al logger de la transacción que se lleva a cabo para que se registre.
- El logger a su vez almacena esta información en el redo buffer, y finalmente se registra la transacción en el log file del storage manager.
- El inicio y el final de la transacción se registran finalmente en el log file.
- Si los datos a ser eliminados no se encuentran en el data buffer, el buffer manager obtiene los datos del storage manager, los coloca en el buffer y

realiza la eliminación. Sin embargo, si los datos a ser eliminados están siendo utilizados por otra transacción de algún otro usuario, la transacción de eliminación debe de abortar o en su defecto esperar a que la transacción en curso finalice, para empezar a ejecutarse.

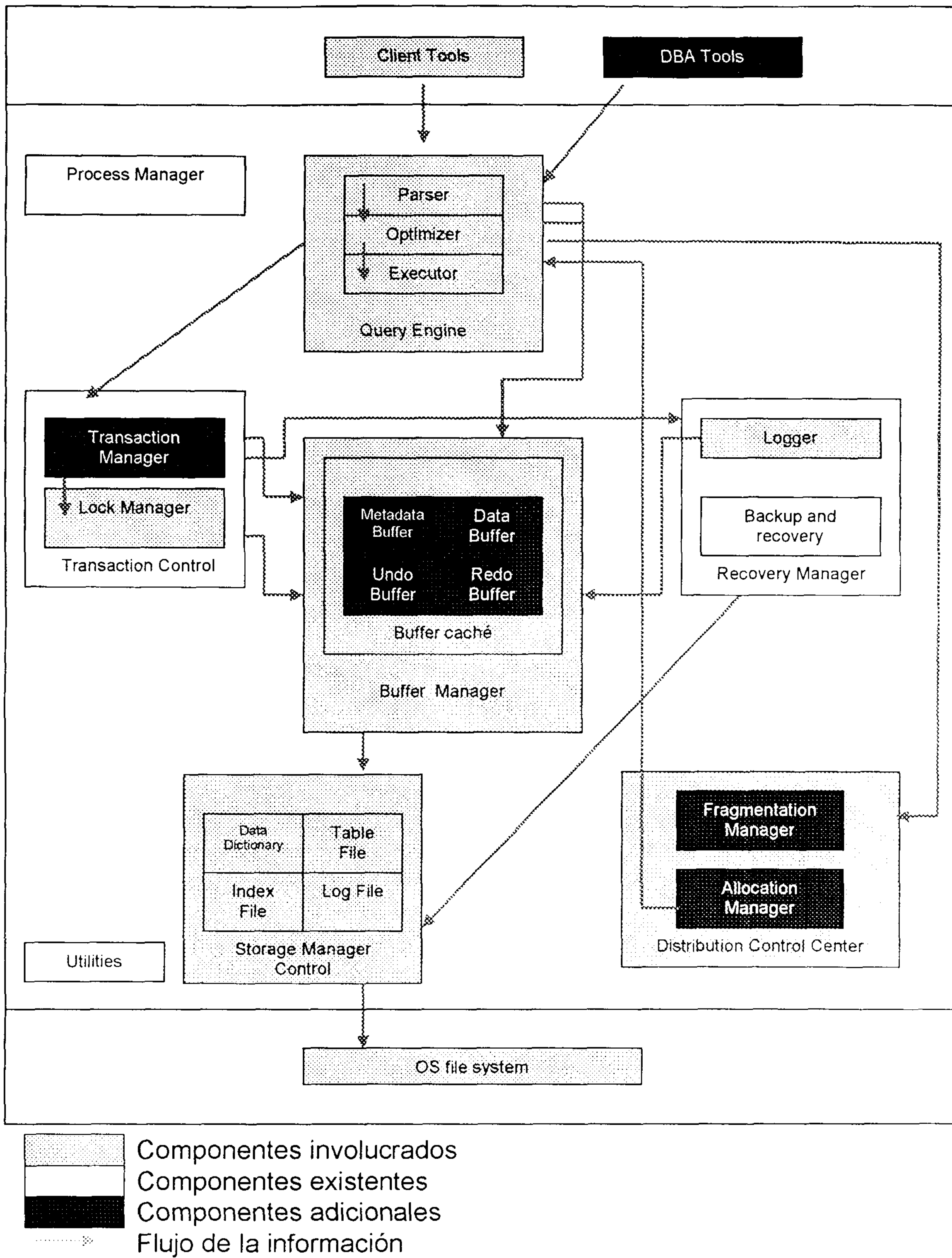


Figura 5.4 Escenario de eliminación de un registro mediante un DDBMS

Método de eliminación de un registro a la base de datos

```
public boolean delete_request_scenario(sql_statement, opcion_delete){
CVisualFunctions FVisual;
boolean query_string, evalpq, evalop, evalex;

query_string = receive_query(sql_statement);
if (query_string){
	evalpq = parser_query(sql_statement);
	if (evalpq){
		evalop=optimize_query(sql_statement);
		if (evalop){
			evalex=execute_query(best_plan, opcion_delete);
			if (evalex){
				FVisual.show_user_results(results);
				System.out.print("La eliminación ha sido realizada con éxito");
			}
			else
				System.out.print("No se pudo eliminar el registro");
		}
		else
			System.out.print("No se pudo optimizar la sentencia");
	}
	else
		System.out.print("No se pudo analizar la sentencia");
}
else
	System.out.print("La sentencia no es correcta");

if (evalex)			//Regresa un valor verdadero o falso
	return true;			//dependiendo del resultado de la evaluación
else
	return false;

}//end of the delete_request_scenario
```

5.5 Escenario 5 Respaldo y recuperación de una base de datos (Backup and Recovery).

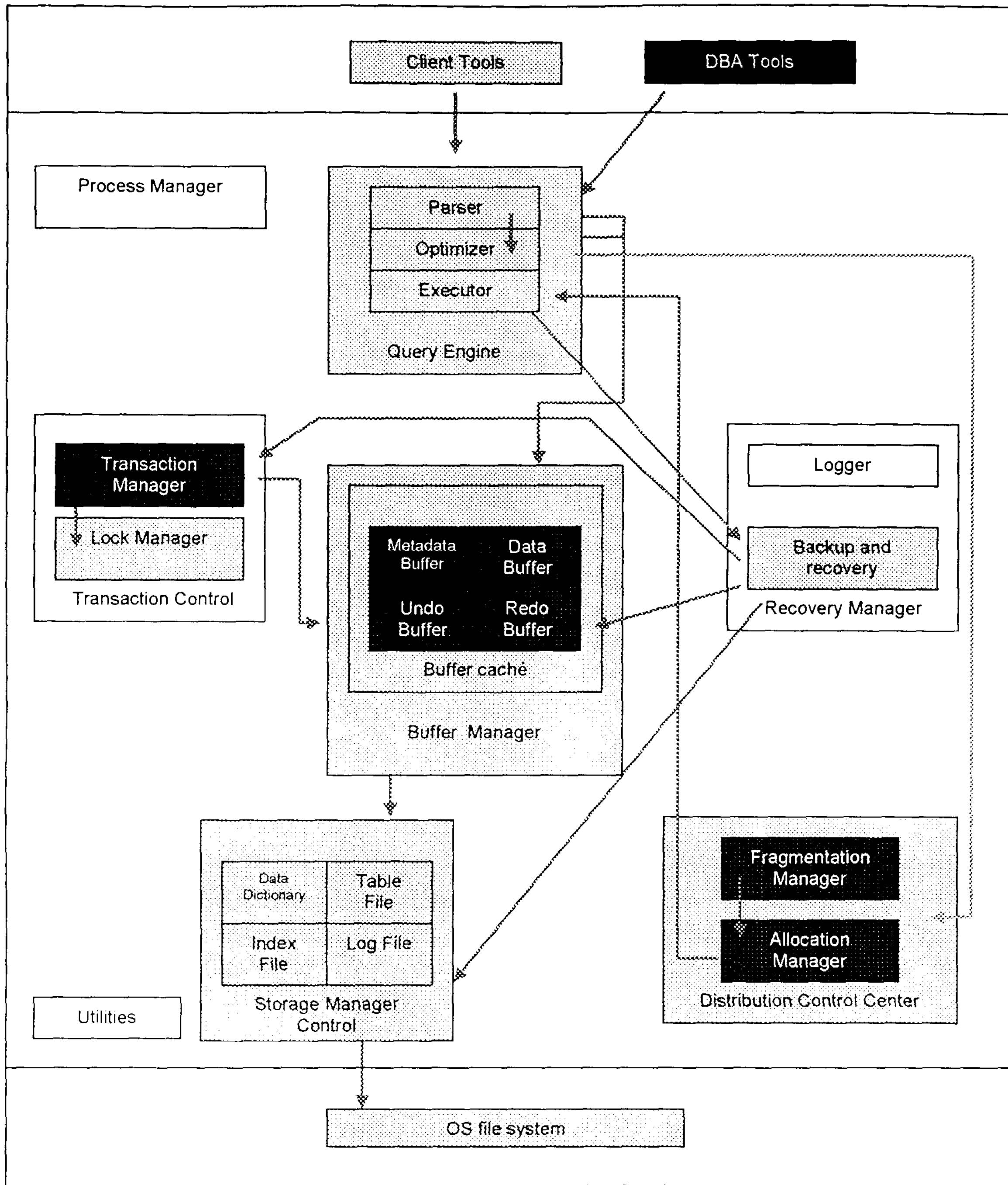
Una de las operaciones que realiza un administrador de base de datos es el respaldo y la recuperación de la misma. Ambas operaciones se analizarán en este escenario, la figura 5.5 de la página siguiente ejemplifica ambas situaciones.

Backup.

- Después de haber sido analizada la petición, el optimizer consulta con el distribution control center la fragmentación y localización de los datos para realizar sus planes de ejecución y elegir el mejor de ellos.
- El allocation manager devuelve al optimizador los resultados de la petición permitiéndole generar sus planes de alternativas de ejecución eligiendo la mejor para el executor.
- El executor canaliza la mejor ejecución del comando de backup al recovery manager, el cual, mediante el módulo de backup and recovery realiza una petición de respaldo al storage manager.
- El storage manager realiza una copia de la totalidad o una parte de los data files y los respalda seguros en un lugar específico.

Recovery.

- En el caso de que algún data file resulte dañado, es necesaria la recuperación del mismo. El comando de recuperación es requerido y el data file se obtiene a partir de un respaldo existente.
- El optimizador se encarga de revisar con ayuda del distribution control center la fragmentación y localización de la tabla a recuperar.
- El distribution control center devuelve al optimizador el control para que realice los mejores planes de ejecución de la recuperación.
- El ejecutor realiza el mejor plan de recuperación que le encarga el optimizador interactuando con el recovery manager.
- El módulo de backup and recovery se encarga junto con el log file del storage manager y el redo buffer del buffer manager de la reconstrucción de los datos.
- El log file es utilizado para rehacer todos cambios registrados en la base de datos desde el ultimo backup realizado.
- El recovery manager realiza una petición al transaction manager para deshacer los cambios realizados por las transacciones que aun no realizaban un commit o rollback al momento del problema.



Componentes involucrados
Componentes existentes
Componentes adicionales
Flujo de la información

Figura 5.5 Escenario de respaldo y recuperación de un registro mediante un DDBMS

Método de recuperación y respaldo de la base de datos

```
public boolean backup_recovery_request_scenario(String sql_statement, int opcion_recback){
static int rec = 1;
CVisualFunctions FVisual;
boolean query_string, evalpq, evalop, evalex;
query_string = receive_query(sql_statement);
if (query_string){
        evalpq = parser_query(sql_statement);
        if (evalpq){
                evalop=optimize_query(sql_statement);
                if (evalop){
                        evalex=execute_query(best_plan, opcion_recback);
                        if (evalex){
                                if opcion_recback = rec then
                                        System.out.print("La recuperación ha sido realizada con
                                                        éxito");
                                else
                                        System.out.print("El backup ha sido realizado con éxito");
                        FVisual.show_user_results();
                        }
                        else
                                if (opcion_recback = rec)
                                        System.out.print("No se pudo ejecutar la recuperación
                                                        solicitada");
                                else
                                        System.out.print("No se pudo ejecutar el backup
                                                        solicitado");
                }
                else
                        System.out.print("No se pudo optimizar la sentencia");
        }
        else
                System.out.print("No se pudo analizar la sentencia");
}
else
        System.out.print("La sentencia no es correcta");

if (evalex)             //Regresa un valor verdadero o falso
        return true;            //dependiendo del resultado de la evaluación
else
        return false;

}//end of the backup_recovery_request_scenario
```

5.6 Conclusiones.

En este capítulo se demuestra la validez de la arquitectura propuesta en el capítulo cuatro mediante escenarios de operaciones comunes de un DBMS. Se ha descrito el flujo de la información internamente y la interacción entre los componentes de la arquitectura propuesta. Se debe mencionar que una validación más completa del manejador queda fuera del alcance de esta tesis debido a la cantidad de componentes que ésta implica. Sin embargo, se sugieren componentes prototipo que representan las diversas piezas que conforman la arquitectura propuesta. Los componentes propuestos en esta tesis pueden ser modificados y adaptados según las necesidades de funcionalidad que se requiera, e incluso se pueden realizar nuevos componentes que interactúen con la arquitectura, y que proporcionen nueva funcionalidad.

La utilización de componentes permite dejar atrás en enfoque monolítico que se tenía de los DBMS´s, mediante ellos podemos adaptar, reducir, ampliar y configurar un DBMS, de acuerdo a las necesidades de procesamiento de información que se requiera, tal como se menciona en [KDIT], algo deseable en las arquitecturas de los DBMS´s es definirlas en fragmentos (componentes) de tal manera que nuevos componentes puedan ser adicionados ó componentes existentes puedan ser intercambiados de una manera flexible. Así, la arquitectura de componentes especifica y restringe las maneras en las que un DBMS debe ser personalizado. A fin de cuentas, la arquitectura también define la noción de componente, así como el dominio de extensiones válidas a un DBMS.

Actualmente se encuentran en desarrollo algunos trabajos de tesis relacionados con la fragmentación de datos [FALV] y el control de concurrencia mediante componentes de software [MCAN], utilizando herramientas de dominio público como el lenguaje de programación java y, MySQL como DBMS.

Capitulo 6 Conclusiones.

6.1 Conclusiones globales.

El tema de las bases de datos distribuidas ha sido ampliamente investigada en el transcurso de los últimos años, sin embargo, dado el vertiginoso avance de los dispositivos computacionales, el poder de procesamiento y las comunicaciones, el crecimiento de las bases de datos se ha enfocado hacia la centralización, donde una gran cantidad de usuarios acceden desde diferentes localizaciones a un repositorio de información, y por lo tanto, la distribución se ha quedado en el nivel de investigación, y no ha sido explotado al ciento por ciento el gran potencial de utilidad que tiene esta rama de la computación.

Como aportación particular de esta tesis, se proporciona una arquitectura que puede ser marco de referencia para mejorar el desempeño de un manejador de bases de datos adaptándole la funcionalidad de la distribución de datos, demostrando el flujo de la información, la estructura de los componentes de localización y fragmentación, y los escenarios más significativos con las operaciones comunes de un DBMS. También se describe el funcionamiento de varios DBMS como el MySQL, Postgress, InnoDB, del área de software libre, así como DB2 y Oracle, entre los comerciales.

De las arquitecturas descritas se toman las ideas principales para generar una arquitectura completa, segura, confiable y funcional, proporcionando métodos que ejemplifican el flujo de la comunicación entre los componentes del DBMS.

Un tema importante es el concepto de componentes, con lo cual se pretende encapsular diversas funcionalidades y adaptarlas según sean requeridas al DBMS. El establecimiento de esta metodología permite flexibilidad, adaptabilidad y escalamiento al DBMS otorgando al usuario, justo las herramientas que utiliza para las necesidades de control de información que requiera.

El estudio de las bases de datos ha destacado mucho en el área informática ya que el almacenamiento de la información representa la historia presente, pasada y en ocasiones, incluso futura de las organizaciones, y por lo tanto, su propagación ha sido bien aceptada por el medio comercial. La variante de las bases de datos distribuidas poco a poco se irá implementando en nuestro medio, donde cada vez es más necesario tener información a la mano, en dispositivos más pequeños y con el mejor rendimiento posible, buscando optimizar el procesamiento, tiempos de respuesta, seguridad y por supuesto, costos.

6.2 Trabajo para futuras investigaciones.

Dada la naturaleza distribuida de las bases de datos, un aspecto importante a investigar es la replicación de los fragmentos de la base de datos, y su continua actualización para garantizar la seguridad de los datos y una redundancia en la disponibilidad de los datos. La arquitectura propuesta puede servir como marco de referencia para mejorar este aspecto del manejador.

La utilización de dispositivos hand-held con restricciones en la administración, conexión y almacenamiento de datos es un área de oportunidad para la aplicación de las bases de datos distribuidas ya que éstas podrían ser utilizadas para suplir esas limitantes con un procesamiento y almacenamiento mínimo, y sería interesante analizar los problemas que conllevaría esta tipo de implementación, arquitecturas, seguridad, conexiones, etc.

Para finalizar se desea dar un ejemplo de la utilización de las bases de datos distribuidas, las cuales podrían ser muy útiles en el campo de la medicina, imaginemos una gran base de datos estandarizada que almacene el historial médico de cada persona, y que la información de cada una de ellas se encuentre almacenada en un dispositivo digital mismo el cual proporcione al momento información vital en casos de emergencia con solo acceder a él mediante un lector, y que al recibir la persona algún tratamiento , se actualice la información, tanto en el dispositivo como en la base de datos, convirtiéndonos de esta manera en un sitio móvil con una pequeña base de datos interconectada a una gran red de datos. Este es solo un ejemplo de aplicación de una base de datos distribuida, pero así como ésta, existen muchos más que podrían adaptarse a esta herramienta, por ello espero que este trabajo sirva como referencia para futuras investigaciones y ayude a comprender mejor el funcionamiento de los DDBMS.

Clases del manejador de base de datos distribuidas

```
//Librerias principales
import QryEngFunctions.*
import TrCtrlFunctions.*
import BufMgrFunctions.*
import RecMgrFunctions.*
import StoMgrFunctions.*
import DistCtrlFunctions.*
import VisualizationFunctions.*

//Instancias a las clases principales
//CQueryEngine QryEng;
//CTransactionControl TrCtrl;
//CBufferManager BufMgr;
//CRecoveryManager RecMgr;
//CStorageManager StoMgr;
//CDistributionControl DistCtrl;
/*********************************************************************************/
//Clase del Query Engine
public class CQueryEngine{

// Método para recibir el query
public boolean receive_query(String sql_statement){
        if (sql_statement != nul)
                return true;
        else{
                System.out.print("Sentencia no válida");
                return false;
        }
} // end of the receive_query
/*********************************************************************************/
//Método para analizar el query
public boolean parser_query(String sql_statement){
CBufferManager BufMgr;
CStorageManager StoMgr;

        if (verify_sintaxis(sql_statement) &&
           BufMgr.verify_metadata(sql_statement) &&
           StoMgr.verify_index(sql_statement) &&
           StoMgr.verify_permisions(sql_statement))
                return true;
        else{
                System.out.print("Existe un error en la sentencia");
                return false;
        }
}// end of the parser_query
```

```
//Método de optimización de consultas
public String optimize_query(String sql_statement){
CStorageManager StoMgr;
CDistributionControl DistCtrl;
        if (StoMgr.index_verification(sql_statement) &&
           DistCtrl.verify_fragmentation(sql_statement)){
                String plan = generate_plan_execution(sql_statement);
                return plan;
        }
        else
                System.out.print("Existe un error en la sentencia");
}// end of the optimize_query
/*****************************************************************************/
//Método de ejecucion de consultas
public boolean execute_query(String best_plan, int opcion){
CFunctionsQryEng FQryEng;
CTransactionControl TrCtrl;
CBufferManager BufMgr;
CRecoveryManager RecMgr;
CStorageManager StoMgr;

static int read_only = 1;
static int insert_query = 2;
static int update_query = 3;
static int delete_query = 4;
static int back_query = 5;
static int rec_query = 6;

        switch (opcion){
        read_only:
                if (StoMgr.dbuffer_mgr(sql_statement)){
                        FQryEng.execute_best_plan(best_plan);
                        return true;
                }
                else{
                        System.out.print("No se puede ejecutar la lectura");
                        return false;
                }
        break;
        insert_query:
                if (TrCtrl.do_transaction(best_plan, opcion) &&
                   RecMgr.logger_mgr(opcion) &&
                   StoMgr.lf_mgr(best_plan,opcion) &&
                   StoMgr.tf_mgr(best_plan,,opcion))
                        return true;
                else{
                        System.out.print("No se puede ejecutar la inserción");
                        return false;
                }
        break;
```

```
        update_query:
                if (TrCtrl.do_transaction(best_plan,opcion) &&
                   RecMgr.logger_mgr(opcion) &&
                   StoMgr.lf_mgr(best_plan,opcion) &&
                   StoMgr.tf_mgr(best_plan,,opcion))
                        return true;
                else{
                        System.out.print("No se puede ejecutar la modificación");
                        return false;
                }
        break;
        delete_query:
                if (TrCtrl.do_transaction(best_plan,opcion) &&
                   RecMgr.logger_mgr(opcion) &&
                   StoMgr.lf_mgr(best_plan,opcion))
                        return true;
                else{
                        System.out.print("No se puede ejecutar la eliminación");
                        return false;
                }
        break;
        back_query:
                if (RecMgr.realize_backup(String sql_statement))
                        return true;
                else{
                        System.out.print("No se puede ejecutar el respaldo");
                        return false;
                }
        break;
        rec_query:
                if (RecMgr.realize_recovery(sql_statement))
                        return true;
                else{
                        System.out.print("No se puede ejecutar la recuperación");
                        return false;
                }
        break;
        } // end of the switch

}// end of the execute_query

} // end of the class CQueryEngine
/*************************************************************************************/
```

```
//Clase del Distribution Control Center
public class CDistributionControl{

//Método de verificacion de la fragmentación de los datos
public String verify_fragmentation(String idtable);
{
statement SqlSt1;
Resultset ResultSQL1;

String sql_select = "select * from FragmentationTable where FragmentationTable.IDTable = " +
idtable;
ResultSQL1 = SqlSt1.executeQuery(sql_select);
	if (ResultSQL1.next()){
		if ResultSQL1.getString("Fragmented") = yes {
			String local = verify_allocation(idtable,idfragment);
			return local;
		}
		else{
			System.out.print("No se encuentra fragmentada la tabla");
			return null;
		}
	}
	else{
		System.out.print("No se encuentra la tabla");
		return null;
	}
}// end of the verify_fragmentation
/**********************************************************************************/
//Método de verificación de la localización de los datos
public String verify_allocation(String idtable, int idfragment)
{
statement SqlSt1;
Resultset ResultSQL1;

String sql_select = "select * from AllocationTable where AllocationTable.IDTable = " + idtable +
			"and AllocationTable.IDFragment = " + idfragment;
ResultSQL1 = SqlSt1.executeQuery(sql_select);
	if (ResultSQL1.next()){
		String localizacion = ResultSQL1.getString("IDSitio");
		return localización;
	}
	else {
		System.out.print("No se encuentra la tabla");
		return null;
	}
}//end of the verify_allocation

}//end of the class CDistributionControl
/**********************************************************************************/
```

```
//Clase del Buffer Manager
public class CBufferManager{
//Método que administra el metadata buffer
public boolean mdbuffer_mgr(String information)
{
CFunctionsBufferMgr FBufferMgr;
CStorageManager StoMgr;

        if (information){
                FBufferMgr.utilize_information(information);
        }
        else {
                StoMgr.find_storage_information(information);
        }
        return true;
} // end of the mdbuffer_mgr
/*****************************************************************************/
//Método que administra el data buffer
public boolean dbuffer_mgr(String data)
{
CFunctionsBufferMgr FBufferMgr;
CStorageManager StoMgr;
        if (data)
                FBufferMgr.utilize_data(data);
        else
                StoMgr.find_data_storage(data);
        return true;
}//end of the dbuffer_mgr
/*****************************************************************************/
//Método que administra el undo buffer
public boolean ubuffer_mgr(String undodata)
{
CFunctionsBufferMgr FBufferMgr;
        if (undodata)
                FBufferMgr.undo_transaction(undodata);
        else
                System.out.print("No existen transacciones por deshacer");
        return true;
}// end of the ubuffer_mgr
/*****************************************************************************/
//Método que administra el redo buffer
public boolean rbuffer_mgr(String redodata)
{
CFunctionsBufferMgr FBufferMgr;
        if (redodata)
                FBufferMgr.redo_transaction(redodata);
        else
                System.out.print("No existen transacciones por rehacer");
        return true;
}// end of the rbuffer_mgr

}//end of the class CBufferManager

/*****************************************************************************/
```

```
//Clase del Transaction Control
public class CTransactionControl{

//Método que realiza las transacciones
public boolean do_transaction(String sql_statement, int opcion);
{
CFunctionsStorageMgr FStoMgr;
CFunctionsTrCtrl FTrCtl;
        if (FStoMgr.verify_md_restrictions(sql_statement,opcion))
                FTrCtrl.realize_transaction(sql_statement, opcion);
        else{
                System.out.print("Existe una violación a las restricciones de la base de datos");
                boolean transaccion_status = false;
        }
        if (transaction_status){
                System.out.print("La transacción ha sido realizada con éxito");
                return true;
        }
        else {
                System.out.print("No se pudo realizar la transacción");
                return false;
        }
}//end of the do_transaction
/*********************************************************************************/
//Método que realiza el bloqueo de los registros
public boolean lock_record(String record)
{
        if (record){
                String record_status = lock;
                return true;
        }
        else{
                System.out.print("No existe el registro");
                return false;
        }
}//end of the lock_record

}//end of the class CTransactionControl
/*********************************************************************************/
//Clase del Recovery Manager
public class CRecoveryManager{

//Método que administra la bitácora
public boolean logger_mgr(String param,String idtran)
{
CFunctionsRecMgr FRecMgr;

        if (param=="write")
                FRecMgr.write_transaction(param,idtran);
        else
                if (param=="read")
                        FRecMgr.read_transaction(param,idtran);
        return true;
}//end of the logger_mgr
/*********************************************************************************/
```

```
//Método que realiza el backup de la base de datos;
public boolean realize_backup(String data)
{
CFunctionsRecMgr FRecMgr;
	FRecMgr.backup_process(data);
	return true;
}//end of the realize_backup
/*******************************************************************************/
//Método que realiza la recuperación de la base de datos
public boolean realize_recovery(String file,String data)
{
CFunctionsRecMgr FRecMgr;
	FRecMgr.get_previous_datafile(file);
	FrecMgr.do_recovery(data);
	return true;
}// end of the realize_recovery

}//end of the class CRecoveryManager
/*******************************************************************************/
//Clase del Storage Manager
public class CStorageManager{

//Método que administra el data dictionary
public boolean datadictionary_mgr(String data)
{
CFunctionsStoMgr FStoMgr;
	FStoMgr.manage_data(data);
	return true;
}//end of the datadictionary_mgr
/*******************************************************************************/
//Método que administra el data file
public boolean datafile_mgr(String file)
{
CFunctionsStoMgr FStoMgr;
	StoMgr.manage_datafiles(file);  //Rutina que administra el index file
	return true;
}//end of the datafile_mgr
/*******************************************************************************/
//Método que administra el index file
public boolean indexfile_mgr(Strinf index)
{
CFunctionsStoMgr FStoMgr;
	FStoMgr.manage_indexfiles(index);
	return true;
}//end of the indexfile_mgr
/*******************************************************************************/
//Método que administra el log file
public boolean logfile_mgr(String log);
{
CFunctionsStoMgr FStoMgr;
	FStoMgr.manage_logfile(log);
	return true;
}//end of the logfile_mgr

}//end of the class CStorageManager
```

```
//Clase de los escenarios de trabajo
public class CWorking_Scenarios{

//Método de consulta de un registro a la base de datos (read-only)
//Realiza las operaciones necesarias para acceder a la base de datos
//distribuida y devolver los resultados obtenidos al usuario

public boolean read_only_request_scenario(String sql_statement, int opcion_read_only)
{
CVisualFunctions FVisual;
boolean query_string, evalpq, evalop, evalex;

query_string = receive_query(sql_statement);
if (query_string){
	avalpq = parser_query(sql_statement);
	if (evalpq){
		evalop=optimize_query(sql_statement);
		if (evalop){
			evalex = execute_query(best_plan, opcion_read_only);
			if (evalex){
				FVisual.show_user_results(results);
				System.out.print("La consulta ha sido realizado con éxito");
			}
			else
				System.out.print("No se pudo ejecutar la consulta");
		}
		else
			System.out.print("No se pudo optimizar la consulta");
	}
	else
		System.out.print("No se pudo analizar la consulta");
}
else
	System.out.print("La consulta no es correcta");

if (evalex)			//Regresa un valor verdadero o falso
	return true;			//dependiendo del resultado de la evaluación
else
	return false;
}//end of the read_only_request_scenario

/*****************************************************************************************/
```

```
//Método de escritura de un registro a la base de datos
public boolean insert_request_scenario(sql_statement, opcion_insert)
{
CVisualFunctions FVisual;
boolean query_string, evalpq, evalop, evalex;

query_string = receive_query(sql_statement);
if (query_string){
        evalpq = parser_query(sql_statement);
        if (evalpq){
                evalop=optimize_query(sql_statement);
                if (evalop){
                        evalex=execute_query(best_plan, opcion_insert);
                        if (evalex){
                                FVisual.show_user_results(results);
                                System.out.print("La inserción ha sido realizada con éxito");
                        }
                        else
                                System.out.print("No se pudo insertar el registro");
                }
                else
                        System.out.print("No se pudo optimizar la sentencia");
        }
        else
                System.out.print("No se pudo analizar la sentencia");
}
else
        System.out.print("La sentencia no es correcta");

if (evalex)             //Regresa un valor verdadero o falso
        return true;            //dependiendo del resultado de la evaluación
else
        return false;

}// end of the insert_request_scenario

/*********************************************************************************/
```

```
//Método de modificación de un registro a la base de datos
public boolean update_request_scenario(sql_statement, opcion_update)
{
CVisualFunctions FVisual;
boolean query_string, evalpq, evalop, evalex;

query_string = receive_query(sql_statement);
if (query_string){
	evalpq = parser_query(sql_statement);
	if (evalpq){
		evalop=optimize_query(sql_statement);
		if (evalop){
			evalex=execute_query(best_plan, opcion_update);
			if (evalex){
				FVisual.show_user_results(results);
				System.out.print("La modificación ha sido realizada con éxito");
			end.
			else
				System.out.print("No se pudo modificar el registro");
		}
		else
			System.out.print("No se pudo optimizar la sentencia");
	}
	else
		System.out.print("No se pudo analizar la sentencia");
}
else
	System.out.print("La sentencia no es correcta");

if (evalex)		//Regresa un valor verdadero o falso
	return true;		//dependiendo del resultado de la evaluación
else
	return false;

}//end of the update_request_scenario
/*********************************************************************************/
```

```
//Método de eliminación de un registro a la base de datos
public boolean delete_request_scenario(sql_statement, opcion_delete)
{
CVisualFunctions FVisual;
boolean query_string, evalpq, evalop, evalex;

query_string = receive_query(sql_statement);
if (query_string){
	evalpq = parser_query(sql_statement);
	if (evalpq){
		evalop=optimize_query(sql_statement);
		if (evalop){
			evalex=execute_query(best_plan, opcion_delete);
			if (evalex){
				FVisual.show_user_results(results);
				System.out.print("La eliminación ha sido realizada con éxito");
			}
			else
				System.out.print("No se pudo eliminar el registro");
		}
		else
			System.out.print("No se pudo optimizar la sentencia");
	}
	else
		System.out.print("No se pudo analizar la sentencia");
}
else
	System.out.print("La sentencia no es correcta");

if (evalex)			//Regresa un valor verdadero o falso
	return true;			//dependiendo del resultado de la evaluación
else
	return false;

}//end of the delete_request_scenario
/*********************************************************************************/
```

```
//Método de recuperación y respaldo de la base de datos
public boolean backup_recovery_request_scenario(String sql_statement, int opcion_recback)
{
static int rec = 1;
CVisualFunctions FVisual;
boolean query_string, evalpq, evalop, evalex;

query_string = receive_query(sql_statement);
if (query_string){
	evalpq = parser_query(sql_statement);
	if (evalpq){
		evalop=optimize_query(sql_statement);
		if (evalop){
			evalex=execute_query(best_plan, opcion_recback);
			if (evalex){
				if opcion_recback = rec then
					System.out.print("La recuperación ha sido realizada con éxito");
				else
					System.out.print("El backup ha sido realizado con éxito");
			FVisual.show_user_results();
			}
			else
				if (opcion_recback = rec)
					System.out.print("No se pudo ejecutar la recuperación solicitada");
				else
					System.out.print("No se pudo ejecutar el backup solicitado");
		}
		else
			System.out.print("No se pudo optimizar la sentencia");
	}
	else
		System.out.print("No se pudo analizar la sentencia");
}
else
	System.out.print("La sentencia no es correcta");

if (evalex)			//Regresa un valor verdadero o falso
	return true;		//dependiendo del resultado de la evaluación
else
	return false;

}//end of the backup_recovery_request_scenario

}//end of the class CWorking_Scenarios
```

Referencias bibliográficas.

[MABB] Michael Abbey,Michael J. Corey, Ian Abramoson, Oracle 8i Guia de aprendizaje, Osborne McGrawHill, 2000.

[FALV] Francisco Álvarez Cavazos, *Distribución de Datos basada en Componentes para Bases de Datos Distribuidas*, Tesis de Maestría, ITESM, 2002.

[RBAN] Ryan Bannon, Alvin Chin, Faryaaz Kassam, Andrew Roszko, *InnoDB Concrete Architecture*, Department of Computer Science, University of Waterloo, 2002

[DBELL] David Bell, Jane Grimson, *Distributed DataBase Systems*, Addison Wesley Publishing Company 1992.

[DB2] DB2 Connect User's Guide, *Distributed Relational Database Architecture*, http://nscpcw.physics.upenn.edu/db2_docs/db2c0/db2c013.htm#HDRDRDAC

[MCAN] Maximiliano Canché Euán, *Control de Concurrencia basado en Componentes para Bases de Datos Distribuidas*, Tesis de Maestría, ITESM 2002.

[RDEC] Rick Decker, Stuart Hirshfield *The Object Concept, an introduction to computer programming* PWS Publishing Company International Thomson Publishing Company, 1995

[KDIT] Klaus R. Dittrich & Andreas Geppert, *Component Database Systems*, The Morgan Kaufmann Publishers, 2000.

[XDON] Xinji Dong, Lijie Zou, Yuan Lin,, *Concrete Arquitecture of PostgreSQL*, Department of Computer Science, University of Waterloo ON N2L 3G1, 2002.

[BERN] Bruce Ernst, *Oracle University Enterprise DBA Part 1A: Architecture and Administration*, Volume 1, Oracle Corporation, Julio 2001.

[DKRO] David M. Kroenke, *Procesamiento de bases de datos, Fundamentos, diseño e instrumentación*, Prentice Hall, 1995

[AMOL]. Andreas Molina, *Propuesta de una arquitectura y un componente genérico para acceso a Bases de donde Datos en aplicaciones Orientadas a Objetos*. Tesis de Maestría, ITESM Mty,NL., 2000.

[SORT] Sandra Maria Ortiz Ramos, *Análisis y diseño de una herramienta computacional que ayude al diseño lógico de bases de datos distribuidas*, Tesis Maestría en Ciencias, Especialidad en Ciencias Computacionales, ITESM,1989.

[POST] *PostgreSQL*,http://www.postgresql.org

[DOWE] David A. Owens and Frederick T. Sheldon; *Tool-based approach to distributed database design: includes Web-based forms design for access to academic affairs data*; *Proceedings of the 1999 ACM symposium on Applied computing*, 1999, Pages 227 – 231.

[TOZS] Tamer Ozsu, Valduriez, *Principles of Distributed database systems*, Second Edition, Prentice Hall, 1999

[ORAC] Oracle9i Database Administrator's Guide, Release 1 (9.0.1), Part Number A90117-01 http://download-ast.oracle.com/otndoc/oracle9i/901_doc/server.901/a90117/toc.htm

[RSTAL] Richard Stallman, *El proyecto GNU, Free software foundation*, 2000. http://www.gnu.org/home.es.html

[GTAR] Gerald Tarcisus, Rahian Al-Ekram, Yu Ping, *Concrete Arquitecture of MySQL RDBMS*, Department of Computer Science, University of Waterloo, 2002.

[PVEL]. Perla Velasco, *Prueba de Componentes de Software basadas en el modelo de Javabeans*. Tesis de Maestría, Unidad de Estudios de Posgrado, Universidad Autónoma de Tlaxcala, 2001..

[MYSQ] *The MySQL AB Company* http:www.mysql.com

Centro de Información-Biblio
30002000624350