UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CONTADURIA Y ADMINISTRACION

TAXONOMIA DE LAS BASES DE DATOS ORIENTADAS A OBJETOS

SEMINARIO DE INVESTIGACION INFORMATICA

QUE PARA OBTENER EL TITULO DE:

LICENCIADO EN INFORMATICA

P R E S E N T A :

AIDA ARACELI ESPINOZA SANCHEZ

ASESOR DEL SEMINARIO: M.A. LUIS EDUARDO LOPEZ CASTRO

MEXICO, D. F. 1998

Agradecimientos:

A Dios:

Por haberme permitido cumplir una más de mis metas al darme vida y salud para conseguirla.

A mis Padres:

Por haberme brindado apoyo, cariño y comprensión a lo largo de mi vida, porque gran parte de este logro se lo debo a ustedes

A mis hermanos:

Por su cariño y apoyo a lo largo de mi carrera.

A ti Miguel:

Por dedicarme tiempo, por ser como eres conmigo y por ayudarme a dar este paso tan importante en mi vida.

A mis amigos:

Porque directa o indirectamente me han apoyado para alcanzar esta meta

A mi asesor M.A. Luis Eduardo López Castro:

Porque gracias a su atención y conocimiento hemos realizado este trabajo que me ha dejado la satisfacción de haber alcanzado mi titulación

CONTENIDO

INTRODUCCIÓN.

Día con día nuestro trabajo requiere de un mayor esfuerzo para llegar a mayores niveles de calidad y confiabilidad a costos relativamente bajos y competitivos. Para ello es de vital importancia contar con información detallada, confiable, actualizada y en el momento preciso, información que puede ser generada en cada una de las actividades que conforman el proceso en el que participamos.

Dicha información constantemente se duplica, y las herramientas y formas que permiten obtenerla ayudan a que sea cada vez más detallada y abundante. Esa información se caracteriza por tener un alto grado de complejidad, por ello es necesario contar con medios que faciliten organizarla, procesarla e interpretarla correctamente. Los sistemas manejadores de bases de datos (SMBD) han y seguirán jugando un papel central en este proceso, de ahí el propósito de escribir en este momento del estado en que se encuentra la tecnología en la cual se sustentan.

Durante las pasadas tres décadas, la tecnología basada en software para soportar el desarrollo de aplicaciones intensivas de datos sufrió la evolución de cuatro generaciones:

1. Sistemas de archivos,
2 Sistemas de bases de datos jerárquicos,
3. Sistemas de bases de datos relacionales ,
4. Sistemas de bases de datos inteligentes o inductivas, sistemas de bases de datos distribuidas, y sistemas de bases de datos orientadas a objetos.

La transición desde la primera generación a la siguiente ha sido motivada por la necesidad de minimizar el costo de desarrollo (que escala rápidamente), así como el costo de mantenimiento y mejoramiento de programas de aplicación.

Los esfuerzos por desarrollar sistemas manejadores de bases de datos orientadas a objetos (SMBDOO) pueden mostrarse en dos grandes avenidas. La primera esta fuertemente influenciada por los avances en lenguajes de programación y puede verse como añadir o extender un lenguaje de programación orientado a objetos para convertirlo en un SMBDOO. Un ejemplo de esto es Gemstone, el cual es una extensión de Smalltalk. La segunda avenida consiste en añadir, extender o modificar los SMBDs existentes para integrarlos con los conceptos de orientación a objetos. Como ejemplo podemos mencionar el sistema Postgres de la Universidad de California en Berkley.

A pesar de que modificar un sistema manejador de bases de datos relacional (SMBDR) para hacerlo orientado a objetos (OO) es un enfoque poco apreciado en el ámbito de investigación para la obtención de un SMBDOO, dicho enfoque es altamente probable en el ámbito comercial actualmente dominado por modelos relacionales que paulatinamente incorporarán conceptos de OO en nuevas versiones

A diferencia de los sistemas manejadores de bases de datos relacionales, los orientados a objetos, carecen de un modelo formal estandarizado, así, cada fabricante ha tratado de sobresalir con su propia metodología, produciendo en el usuario final una resistencia natural a su uso, sin embargo, todos ellos parten de los mismos conceptos dando las mismas facilidades, prometiendo grandes ventajas con respecto a los SMBDR.

Las bases de datos requieren un modelo propio de datos y, en ausencia de un estándar, ciertos conceptos concernientes al modelo de datos se han agrupado en un modelo central o modelo básico. Esta solución es suficientemente poderosa para satisfacer muchos de los requerimientos de las aplicaciones avanzadas, e identifica las principales diferencias respecto a los modelos convencionales.

Con base en lo anterior existe la necesidad de saber el avance que la industria ha tenido en estas nuevas tecnologías, cuáles son las direcciones futuras y sus últimos avances en la investigación.

Esperando que el siguiente trabajo contribuya proporcionando los más importantes conceptos del paradigma orientado a objetos, aplicados a las bases de datos, así como una visión general de los SMBDOO disponibles comercialmente, para que, tomándolos como base, aquel que lo requiera determine su conjunto de posibilidades, profundice en lo que considere y lo aplique.

El presente trabajo está organizado en 5 capítulos. En el capítulo 1 se mencionan los conceptos generales tanto de los sistemas de bases de datos relacionales como de los sistemas de bases de datos orientados a objetos, que nos ayudarán a comprender mejor las diferencias entre éstos.

El capítulo 2 contiene un análisis de algunos productos clasificándolos de acuerdo a su naturaleza, tomando en consideración algunas características como usos principales, recuperación, administración de transacciones, soporte de bases de datos distribuidas, manejo de datos multimedia, entre otros.

El capítulo 3 trata acerca de los estándares actuales que dan paso a la formalización de los conceptos del paradigma orientado a objetos, entre ellos encontramos: OMG, ODMG-93 y ANSI-SQL3.

En el capítulo 4 se da una breve descripción de algunos sistemas actualmente disponibles como productos comerciales, prototipos terminados o prototipos en desarrollo, entre ellos encontramos: GemStone, Iris, ObjectStore, Picquery, Sim, Vbase, entre otros.

Por último el capítulo 5 es un breve estudio de un SMBDOO comercial llamado ObjectStore.

1. SISTEMAS MANEJADORES DE BASES DE DATOS ACTUALES

Los sistemas convencionales (relacionales y prerrelacionales) han servido para mejorar las necesidades de aplicaciones del ambiente para el cuál fueron designadas, es decir, aplicaciones de procesamiento de datos en negocios, tales como control de inventario, nóminas, cuentas por cobrar, etc. Sin embargo, tan pronto como la tecnología de Bases de datos relacionales abandonó los laboratorios de investigación y registró su marca en el mercado serias limitaciones comenzaron a ser expuestas. En otras palabras, una variedad de aplicaciones comenzó a ser identificada como difícil para implantarse con el uso de sistemas de bases relacionales.

Estas aplicaciones incluyen diseño auxiliado por computadora, ingeniería, ingeniería de software y gestión y administración de procesos (CAD, CAE, CASE y CAM), sistemas basados en conocimiento (sistemas expertos y "shell" para sistemas expertos), sistemas de multimedia que manejan imágenes, gráficas, voz, y documentos textuales; modelos estadísticos y científicos y análisis de programas, sistemas de información geográfica, etc. Además presentan serias dificultades atribuibles al modelo de datos (intuitivamente, un modelo de datos es una representación lógica de datos, relacionales e interacción en los datos), de ahí la necesidad de un progreso en tecnología de bases de datos, que esta realizando una transición y además esta impulsando extensiones a sistemas administradores de datos actuales

El núcleo de transición radica en el paradigma orientado en los lenguajes de programación orientada a objetos después de la introducción de Simula-67 y Smalltalk-80. Hay dos razones principales por las cuales la metodología orientada a objetos es un sólido fundamento para la nueva generación de tecnología de bases de datos.

En primer lugar, la metodología orientada a objetos ofrece un modelo de datos que incluye los modelos de datos de un sistema de base de datos convencional. Un modelo de datos orientado a objetos puede representar no solamente los datos, las relaciones y la interacción de datos de modelos de datos convencionales, sino también permite encapsulamiento de datos y programas que operan los datos en un protocolo definido y proporcionan una estructura uniforme para el trato de tipo de datos arbitrarios definidos por el usuario Algunas relaciones en el modelo de datos, que son difíciles en Sistemas de bases de datos convencionales, son inherentes en un modelo de datos basado en objetos.

Otra razón es que la metodología orientada a objetos, a través de la noción de encapsulamiento y herencia, esta fundamentalmente diseñada para reducir la dificultad de desarrollo y evolución de sistemas complejos de software. Esta es, precisamente, la meta que motivó a la tecnología de administración de bases de datos a transformar de sistemas de archivos hacia sistemas de bases de datos relacionales. Un modelo de datos orientado a objetos

inherente satisface el objetivo de reducir la dificultad de diseñar y desarrollar bases de datos complejas, sofisticadas y muy grandes.

Podemos decir que el modelo orientado a objetos es hoy día una de las vías más prometedoras en el desarrollo de software y que podemos prever razonablemente que utilizando un enfoque similar para el manejo de bases de datos y para el desarrollo de aplicaciones de uso intensivo de datos dispondríamos de todos los beneficios existentes en el campo de la ingeniería de software. Es por eso que la programación orientada a objetos es otro paso evolutivo en el diseño y construcción de software. Por ello en este capitulo se describirán todos estos conceptos básicos tanto de los sistemas de bases de datos relaciones como de los sistemas de bases de datos orientados a objetos.

1.1 Conceptos generales del modelo relacional.

En primer lugar se enunciarán las características generales de un modelo de datos, tomando en cuenta diferentes propuestas de varios autores, para posteriormente entrar de lleno a las características del modelo relacional.

Modelo de datos.

Un modelo de datos en general es un formalismo matemático que permite una abstracción de la realidad contemplando los siguientes componentes mínimos:

- Una colección de objetos que representan entidades de la realidad [Date].
- Una notación para describir dichos objetos [Ullman].
- Un conjunto de operaciones para manipular tales objetos [Ullman 88 y Date 95].
- Una colección de reglas que dan un estado válido a los objetos [Date 95].
- Un conjunto de estructuras de datos asociadas a los datos del modelo. Tales estructuras son llamadas estructuras algebraicas que facilitan la representación y transformación de los datos en la implementación [Abellanas 91].

A continuación se analizarán las características generales del modelo relacional, al ser éste un modelo de datos muy utilizado comercialmente y cuyo fundamento radica en el concepto matemático de relación entre conjuntos, por lo que resulta muy sólido desde el punto de vista formal. De esta manera, se podrán comparar las características de este modelo con las del modelo orientado a objetos, analizando en qué puntos convergen y en qué puntos son distintos, así como en qué aspectos se contemplan uno con otro.

Descripción.

El modelo relacional de bases de datos fue propuesto por E.F. Codd en 1969 – 70 como una solución a los problemas de diseño de bancos de datos grandes. Está fundamentado principalmente en la teoría de conjuntos y el concepto matemático de relación y pretende hacer independiente la estructura interna de los datos de la semántica (significado) de éstos.

Para lo anterior, es importante considerar que existen tres niveles en los que se visualiza o se abstrae la información en los sistemas computarizados:

Nivel interno: se refiere a la manera en que la información es almacenada físicamente en los dispositivos correspondientes. Constituye la implementación dependiente del manejador de base de datos y arquitectura de la máquina.

Nivel conceptual: es la manera en que están estructurados los datos en la realidad. Es una visualización de los datos desde un enfoque estructural. La manera en que son relacionados unos datos con otros.

Nivel externo: es el nivel más cercano al punto de vista del usuario final. Es la forma en que este tipo de usuario percibe la información.

Lo anterior implica que existen diferentes visualizaciones de la información así como diferentes tipos de usuario. Estos tipos de usuario son, en general:

- Administrador de la base de datos DBA: (nivel físico y conceptual)
- Programador de aplicaciones (nivel conceptual y externo)
- Usuario final (nivel externo· consultas y reportes)

1.2 Conceptos generales del modelo orientado a objetos.

Existen varios paradigmas (modelos) de programación cuyo enfoque para atacar un problema dado, se basa en elementos diferentes:

- Procedural: Algoritmos
- Lógicos: Metas, expresadas con cálculo de predicados
- Reglas: Reglas If - Then
- Restricciones: Asociaciones invariantes
- Objetos: Clases y objetos.

Como se puede observar, el paradigma de orientación a objetos se basa en el modelado de la realidad considerando que en ella todo es un objeto con características y comportamiento particulares. Surge a finales de los 60's con un lenguaje de programación llamado SIMULA. En éste se incorporaron por primera vez los conceptos de clase, herencia, sobrecarga; etc. Sin embargo, es hasta mediados de los 80's cuando comenzó a popularizarse comercialmente.

Es importante considerar que este paradigma interviene en diferentes etapas del desarrollo de sistemas, y que no necesariamente todas estarán basadas en él para un proyecto completo[1]. Existe el análisis orientado a objetos (AOO), el diseño orientado a objetos (DOO) y el modelo de datos orientado a objetos (BDOO). Sin embargo, es común encontrar que el desarrollo de un sistema se basa en el paradigma orientado a objetos en la etapa de diseño; pero la implementación del banco de datos sigue el modelo relacional.

De ahí que existan muchos productos de bases de datos cuya interfaz de aplicación (las pantallas de captura, front-end) esté basada en un lenguaje de programación orientado a objetos; pero la estructura de la base de datos (back-end) se defina en términos de tablas.

A continuación se mencionarán de manera general los conceptos principales del modelo orientado a objetos.

[1] Lo anterior considerando que actualmente no existe un lenguaje orientado a objetos cuyo modelo de datos esté suficientemente maduro (o al menos tan maduro como los sistemas relacionales).

a) Clases.

Una clase es el conjunto de atributos, operaciones y reglas que un conjunto de objetos comparten. Se puede decir que una clase es como un molde del cual se crean los objetos. Los objetos que pertenecen a una clase determinada son llamados instancias, por ello una clase puede tener una o varias instancias, pero una instancia tiene sólo una clase, a esto se le conoce como herencia simple.

Una clase describe las propiedades (variables de una instancia o atributos), el comportamiento (protocolo o mensajes), la semántica (métodos u operaciones que describen el funcionamiento interno) de un conjunto de objetos similares, y las relaciones con otros objetos

Del mismo modo, una clase será a la vez un módulo y un tipo. Como módulo la clase encierra un número de facilidades o recursos que ofrecerá a otras clases. Como tipo describe un conjunto de objetos o instancias que existirán en tiempo de ejecución. La clase se interpreta también como una abstracción del concepto objeto.

Cada objeto sabe a qué clase pertenece, y cada una de ellas puede ser manipulada dinámicamente a tiempo de ejecución, por ejemplo para crear instancias (a partir del mensaje *new*), o bien pasándola como un parámetro a alguna operación.

La mayor parte de los SMBDOO incluyen clases (de la familia de Smalltalk, GemStone, Orion, Flavors, Versant, Gbase), sin embargo en algunos casos, este concepto de clase es substituido por el de *tipo* que a diferencia es un mecanismo estático para la definición de nuevos objetos, ejemplos de ellos son lenguajes de programación como C++, Object Pascal, Simula, y sistemas de bases de datos como Vbase y O_2

Un tipo, al igual que una clase define las propiedades comunes de un conjunto de objetos (atributos y operaciones) Los tipos son una herramienta a tiempo de compilación para verificar que un programa sea correcto en el manejo de variables y parámetros en general, restringiendo el lenguaje a la declaración y definición de variables y funciones con sus respectivos tipos. Los tipos son estáticos, ya que no pueden ser modificados durante la ejecución de un programa, ni tampoco pueden utilizarse como parámetro de una función.

Cada clase, por lo general describe dos grandes grupos de operaciones: de acceso, que dan información sobre un objeto y de transformación, que provocan cambios en el objeto. Las operaciones que dan información sobre el objeto caracterizan lo que se denomina estado del

objeto y las operaciones de transformación cambian ese estado y las operaciones de transformación cambian este estado.

b) Objetos.

Un objeto es cualquier ente identificable, que tiene un estado interno y un conjunto de operaciones que pueden ser aplicadas para consultar, o modificar su estado, estructurado en variables de instancia (campos), que pueden hacer referencia a valores primitivos o a otros objetos compuestos. Las estructuras de datos, conocidas también como variables de instancia, conforman el estado interno del objeto. El valor de las variables puede cambiarse dinámicamente durante la vida de un objeto. Las operaciones de un objeto, conocidas en su conjunto como el protocolo o interfaz pública, son las únicas que pueden consultar o modificar su estado, protegiéndolo de alteraciones derivadas de operaciones externas que lo corrompan. A las operaciones de un objeto se les conoce también con los nombres de métodos o funciones de miembro.

En la metodología de objetos, las funciones y los datos son encapsulados en un solo lugar, los cuales son protegidos contra cualquier alteración de forma directa. El diseño orientado a objetos tiene como objetivo identificar a los objetos en el dominio del problema, con la estructura que definirá el conjunto posible de estados en variables de instancia, y las operaciones que dicho objeto puede hacer.

Otra característica importante de un objeto es su identidad. La identidad de un objeto es un identificador único, por lo general implícito, que distingue a un objeto de todos los demás. Cada objeto se identifica por un simple IDO (Identificador del Objeto)[2]. La identidad de un objeto tiene una existencia independientemente de los valores de los atributos del objeto. Utilizando el IDO, los objetos pueden compartir otros objetos y se pueden construir redes de objetos. Sin embargo, hay algunos modelos en los cuales lo mismo se permiten objetos y valores (a menudo llamados literales), y donde no todas las entidades se representan como objetos. Informalmente, un valor es autoidentificable y no tiene ningún IDO asociado a él. Todas las entidades primitivas, tales como enteros o caracteres, se representan por valores, mientras que las entidades no primitivas se representan por objetos. En general, los valores complejos son útiles en los casos en los que se definen agregados (o conjuntos) para usarse como componentes de otros objetos, pero que no serán utilizados como entidades independientes. Un ejemplo típico es el de las fechas. Éstas se usan con frecuencia como componentes de otros objetos; sin embargo, es raro que un usuario desee hacer una consulta sobre la clase de todas las fechas.

[2] En ingles, siglas OID (Object Identifier).

La identidad de un objeto introduce al menos dos nociones diferentes de igualdad entre objetos:

- La primera denotada por "=", es la igualdad por identidad: dos objetos son idénticos si son el mismo objeto, esto es, si tienen el mismo identificador.
- La segunda denotada por "= =", es igualdad por valor: dos objetos son iguales si los valores de sus valores de sus atributos son recursivamente iguales.

Por tanto, dos objetos idénticos son a su vez iguales, mientras que lo inverso no siempre es cierto. Ciertos modelos de datos también manejan un tercer tipo de igualdad, a menudo denominada igualdad superficial, en la que dos objetos son superficialmente iguales, aunque no sean idénticos, si todos sus atributos comparten los mismos valores y las mismas referencias.

Por otro lado, es importante saber que los datos y procedimientos están encapsulados en una capa protectora a fin de prevenir modificaciones indeseadas. Los objetos tienen un estado, el estado de un objeto es recordado por sus datos componentes. Además, los objetos pueden procesar información, aunque no es mucho el procesamiento que ocurre en el interior del objeto. Los procedimientos solamente establecen los valores de los datos privados y regresan estos valores cuando se solicita. Sin embargo, los objetos reales por lo general tienen una capacidad de procesamiento más compleja. Los objetos además tienen un ciclo de vida a diferencia de los procedimientos, es posible crearlos y destruirlos. Siempre y cuando un objeto este vivo y activo, se puede realizar el acceso a cualquiera de sus elementos públicos. Después una vez destruido, todas sus memorias son irrecuperables y no será posible comunicarse con él.

Objetos complejos.

Una de las grandes desventajas de los SMBDR es la restricción de definición de relaciones a partir de datos atómicos, siendo necesario partir cada entidad compleja a representar, en múltiples tablas relacionadas por llaves foráneas.

En los SMBDOO cada objeto está compuesto de un conjunto de variables de instancia que pueden almacenar o hacer referencia a otro objeto igual de complejo a partir de IDOs, y métodos o funciones que definen su comportamiento. El conjunto de variables de instancia representan la estructura y el estado de un objeto, mientras los métodos son las operaciones que puede llevar a cabo.

Los SMBDOO tienen la característica común de contar con clases de colecciones, estructuras de datos indexadas como arreglos, y no indexadas como diccionarios, conjuntos, bolsas, colecciones ordenadas, etc. que sirven para almacenar objetos de forma equivalente como lo hace una tabla en un SMBDR, pero con la diferencia de que cada una de estas colecciones es capaz de almacenar cualquier clase de objeto, incluso de diferentes clases al mismo tiempo. Esto facilita la construcción de objetos complejos con algoritmos de búsqueda muy eficientes, y de manera transparente

De acuerdo a la estructura de cada objeto y las operaciones que se deseen hacer sobre él, se utilizará cualquiera de las colecciones definidas dentro del SMBDOO, o colecciones definidas por el mismo usuario, sin restringir al uso de tablas de columnas y renglones, y tipos de datos primitivos.

Por otro lado, se requieren de operaciones para el tratamiento de objetos complejos, no importando su estructura y su clase, las cuales se deben propagar a través de toda la estructura del objeto de forma recursiva, por ejemplo sacar una copia a profundidad, borrado en cascada, equivalencia e identidad.

c) Encapsulamiento

A la combinación de datos y de procedimientos en una declaración de objetos se denomina encapsulamiento. Se dice que es la manera de ocultar los detalles de la representación interna de un objeto presentando solamente la interfaz disponible al usuario. Este mecanismo es indispensable si se piensa en la creación de componentes de software reusable.

El principio de encapsulamiento aplicado a los objetos de una base de datos exige la integración de la estructura y el comportamiento de los mismos en torno de una sola entidad: el objeto. Este principio permite ocultar los detalles de implementación (de los métodos aplicables al objeto) y asegurar la integridad de su parte estructural. Sin embargo, es limitativa desde el punto de vista de base de datos, ya que exige la especificación de métodos especializados a cada tipo de consulta, perdiendo con ello la flexibilidad tradicional ofrecida por un SMBD.

d) Herencia

La herencia es una relación entre clases que permite declarar una clase (subclase) como extensión o especialización de otra clase (superclase). Las nuevas clases de objetos pueden ser

definidas a partir de clases previamente construidas, de tal forma que nuevas definiciones pueden reutilizar parte o todo el diseño y la programación de una o varias clases ya probadas, y utilizarlas en problemas reales. En cuestión de herencia podemos hablar de herencia simple y herencia múltiple.

1. Herencia simple: Permite crear una nueva clase basándose en otra clase más general. Una clase derivada adquiere todas las propiedades y métodos de la clase de la que se derivó, pudiendo añadir nuevos elementos o modificar ligeramente los ya existentes.

2. Herencia múltiple: Proporciona la posibilidad de derivar una clase de un conjunto de ellas. En la herencia múltiple se mantiene una liga con sus clases base, de tal forma que si utiliza algún método y no lo tiene, lo solicita a las clases de las cuales se derivó.

Las clases se organizan en estructuras de árboles (herencia simple), o en estructuras de grafos (herencia múltiple). En tales relaciones, a las clases predecesoras se les conoce como clases padre o superclases y las clases sucesoras clases derivadas o subclases. Cuando un objeto es creado, éste tendrá las variables de instancia (campos), y los métodos (funciones) definidos en su clase, además de todas aquellas que estén definidas en su superclase. Una clase al heredar puede ignorar algunas de las operaciones de su superclase o redefinirlas, de acuerdo a su responsabilidad dentro del sistema.

En los SMBDOO la herencia se puede clasificar como de substitución, inclusión, restringida y especializada. En la herencia de substitución, una clase A hereda de la clase B todos sus atributos (donde los atributos son las variables de instancia y las operaciones de la clase), e incluye algunas operaciones adicionales, de tal forma que todas las instancias de B pueden ser substituidas por instancias de la clase A en un programa.

La herencia de inclusión está relacionada al concepto de subtipo. Aquí lo importante es heredar la estructura y no las operaciones. La herencia restringida es un caso particular de inclusión, donde algunos de los atributos son restringidos a cierto rango de valores, lo cual no hace posible que una instancia de la clase B pueda substituir a una instancia de la clase A de la que hereda. Un ejemplo de ello es un objeto de la clase *Esposo* con una superclase *Personas*, y con el atributo estado civil restringido al valor CASADO. Finalmente la herencia de especialización es cuando una clase A hereda de una clase B, agregando nuevas variables de instancia y nuevos métodos.

Una de las mayores ventajas de un SMBDOO es la herencia, que permite utilizar el mismo código en múltiples aplicaciones. En el caso de que dos clases tengan partes similares, estas partes son extraídas y definidas en una clase general o abstracta desde donde se pueden heredar (factorizar). Por otra parte la herencia ayuda a conceptualizar mejor un sistema, siendo de forma natural definir clases, por ejemplo *perro* como subclase de ***mamífero***, dando

una mejor idea de la estructuración del código y operaciones dentro de la aplicación.

e) Polimorfismo

El polimorfismo permite la manipulación de objetos de clases distintas como si fueran de la misma clase, con lo cual es posible definir interfaces uniformes para diferentes tipos de objetos. Las entidades internas de un programa deben poder manejar conjuntos de objetos de diferentes clases de la misma forma en que manejan conjuntos de objetos iguales.

El concepto de polimorfismo se puede aplicar en dos sentidos, en variables y funciones. En variables el polimorfismo quiere decir que una de ellas puede almacenar diferentes clases de objetos, durante un tiempo de ejecución de su ámbito. En funciones, se dice que una función o un método es polimórfico si éste puede recibir diferentes clases de objetos como argumentos en el curso de la ejecución de su programa.

El polimorfismo es resuelto en la mayoría de los SMBDOO dinámicamente, ya que cada variable puede guardar diferentes clase de objetos durante la ejecución de la aplicación, siendo se produce una anomalía de un programa. Se maneja tal manera, el álgebra relacional puede utilizarse en los objetos complejos, y el resultado serán colecciones de objetos más que relaciones puras de dominios atómicos.

2. CLASIFICACIÓN DE PRODUCTOS

A continuación se presentará un análisis de algunos productos clasificándolos de acuerdo a su naturaleza:

- Productos orientados a objetos con persistencia implementada
- Productos relacionales con extensión a objetos
- SMBDOO diseñados desde el principio

Para llevar a cabo dicho análisis, se tomará en consideración cómo son soportadas las siguientes características:

- Usos principales.
- Soporte o no de administración de versiones de objetos
- Recuperación
- Administración de transacciones
- Herencia múltiple
- Concurrencia/bloqueo
- Soporte de bases de datos distribuidas
- Evolución dinámica del esquema (poder modificar la estructura de alguna relación o clase aún teniendo datos ya almacenados)
- Datos multimedia
- Interfaz con lenguajes de programación para enlace del esquema de datos
- Estado actual de desarrollo
- Características especiales

En algunos productos se describen también algunas características adicionales, pero estas características no están definidas en un esquema global para todos los productos, sino que se mencionan conforme los manuales las describen.

2.1 Lenguajes orientados a objetos con extensión de persistencia.

Varios de los lenguajes orientados a objetos han sido enriquecidos con el concepto de persistencia (operaciones o métodos para el manejo de almacenamiento secundario de los objetos); para dar soporte a las estructuras de las bases de datos.

2.1.1 GemStone

Usos principales: Ambientes cooperativos.
Administración de versiones: Sí.
Recuperación: A través de páginas replicadas (shadow page).
Administración de transacciones: Sí.
Herencia múltiple: Sí.
Concurrencia/bloqueo: Tres tipos de bloqueos optimista y pesimista.
Soporte de bases de datos distribuidas: Sí.
Evolución dinámica del esquema: Sí (limitada).
Datos multimedia: Sí.
Interfaz con lenguajes de programación: C, C++, OPAL, Parc Place, SmallTalk, Topaz.
Estado actual de desarrollo: Producto maduro y comercial.

Características especiales:

GemStone comenzó como un proyecto de desarrollo en la compañía Servio Logic en 1985 y se convirtió en un producto comercial en 1988. GemStone une los conceptos de orientación a objetos con los de los sistemas de bases de datos y proporciona un lenguaje de bases de datos orientadas a objetos llamado OPAL que se utiliza para la definición y manipulación de datos y cálculos en general. Es uno de los SMBDOO más maduros disponibles en el mercado.

El modelo de programación y el modelo de datos: GemStone se diseñó como un sistema que pudiera convertir a SmallTalk-80 en un lenguaje orientado a objetos con ambiente persistente. OPAL se creó como una extensión a la semántica de SmallTalk-80. El sistema utiliza el modelo orientado a objetos de SmallTalk-80.

Objetos: Cada objeto tiene una identidad única y es independiente de su estado. Esto es llamado "apuntador orientado a objetos" (AOO) y no es visible para el usuario. La estructura

interna de cada objeto se compone de campos llamados "variables de instancia" que se distinguen de las variables OPAL comunes. Una variable de instancia tiene un nombre definido en su clase, y un valor que también es un objeto. El estado interno de un objeto es, por tanto, un conjunto de objetos AOO que corresponden a las variables de instancia de sus clases. Una variable de instancia puede definirse como restringida, (lo que significa que sus valores deben ser objetos de cierta clase); o no restringidas. No obstante, existen clases como la clase conjunto o la clase arreglo que no tienen variables de instancia nombradas. Sus instancias son objetos cuyo estado interno es un conjunto de AOOs que cambian de tamaño dinámicamente.

Métodos y mensajes: El modelo OPAL soporta objetos completamente encapsulados; es decir, las variables de instancia son accesibles sólo a través de métodos. Después de que el usuario define una nueva clase, el sistema proporciona un método necesario para accesar los valores de instancia de sus instancias. El usuario puede modificar estos métodos. En OPAL la verificación de tipos no se lleva a cabo en la compilación. Si los objetos pueden responder a mensajes, esta verificación se llevará a cabo sólo en tiempo de ejecución. Sin embargo, soporta completamente el enlace tardío (late binding) y sobrecarga de operadores.

Clases: Cada clase se representa por un tipo de objeto llamada "definición de clases de objetos" (CDO). Este objeto describe la estructura interna y la interfaz de las instancias de la clase. Cada objeto hace referencia al apuntador AOO de su CDO. Cada CDO almacena los nombres y restricciones de sus variables de instancia, así como los cuerpos de los métodos y sus mensajes. Cuando un objeto recibe un mensaje, encuentra qué método debe ejecutarse por la referencia del CDO de su clase. Los CDOs también se utilizan para la creación de nuevos objetos. El modelo OPAL soporta sólo herencia simple; esto es, una clase puede heredar variables y métodos sólo de otra clase, que posteriormente puede modificarse con herencia de sustitución.

El modelo de datos OPAL proporciona todas las características mandatorias que un SMBD debe tener, tales como:

- Persistencia. Todos los objetos que se crean de los CDOs se convierten en persistentes después del final de una transacción que los haya creado.

- Concurrencia de transacciones. GemStone es un sistema multiusuario y por lo tanto proporciona mecanismos para soportar conflictos. Inicialmente el sistema se construyó con un esquema de control de concurrencia optimista y posteriormente se proporcionó concurrencia pesimista también.

- Recuperación. Cada vez que una transacción comienza, se crean copias de los objetos que tienen que ser accesados en disco sólo hasta el final de la transacción.

- Almacenamiento secundario. Soporta indexación y bloques de agrupación.

- Mecanismos de consulta. OPAL proporciona sentencias especiales de consulta que implementan el acceso asociativo. Es decir, selección de un conjunto de objetos que satisfagan una condición dada.

Existe un conjunto de subclases predefinidas heredadas de la clase colección para manipular conjunto de objetos:

- Array: Colección secuencial (acceso indexado a sus elementos).

- Bolsas: Colección no secuencial sin orden alguno (puede tener elementos repetidos).

- Conjuntos: Colección no secuencial sin orden alguno, sin elementos repetidos.

Deficiencias del modelo:

- Las clases tienen solamente semántica intencional, es decir, el usuario debe crear un conjunto de objetos y llenarlos con instancias.

- Los objetos no pueden borrarse explícitamente.

- La herencia es simple y tiene una semántica de sustitución.

Restricciones de integridad:

- No soporta la integridad de llave única. Debe implementarse manualmente en el método de actualización.

- La integridad referencial se da por la imposibilidad de borrar objetos directamente. Sin embargo, esto genera problemas al no poder borrar objetos que probablemente queden con un apuntador "volando" (apuntador a un objeto que ya no existe).

- El usuario debe modificar el método creador de instancias "new" para agregar valores por omisión a ciertos atributos.

El lenguaje de consulta:

Proporciona tres operadores para consulta a la base de datos Están implementados como mensajes para las instancias de las clases del tipo colección:

A select: [x | predicado lógico].
donde
A es una instancia de la clase colección.

Ejemplo:

Cuadros select: {is | (s.centro.x valor = 0 0) & (s.centro.y.valor = 0.0) }
(Encuentra en el conjunto de cuadros aquellos que son concéntricos al origen).

A reject: [x | predicado lógico]:
donde
A es una instancia de la clase colección.
Similar a select, sólo que con la operación contraria.

A detect: [x | predicado lógico]:
donde
A es una instancia de la clase colección.
Se ejecuta la búsqueda hasta encontrar el primer elemento que cumpla con la condición.

Arquitectura del sistema.

El sistema esta basado en dos procesos fundamentales:

- Stone. proceso interno que manipula los recursos. Controla las entradas y concurrencia de usuarios. Es un servidor de recursos internos para el proceso Gem.
- Gem: provee compilación y ejecución de programas OPAL, almacenamiento y recuperación de objetos, autorización de usuarios y un conjunto predefinido de clases y métodos OPAL para los programas de usuarios. Constituye un servidor para los programas de aplicación.

2.1.2 Objectstore.

Usos principales: Ambientes colaborativos (varios usuarios actualizando el mismo dato).
Administración de versiones: Sí.
Recuperación: A través de bitácora de transacciones (log file).
Administración de transacciones: Sí, un poco restringida.
Herencia múltiple: Sí.
Concurrencia/bloqueo: Sí, dos tipos de bloqueos.
Soporte de bases de datos distribuidas: No.
Evolución dinámica del esquema: Sí (limitada).
Datos multimedia: Sí.
Interfaz con lenguajes de programación: C, C++.
Estado actual de desarrollo: Ya es un producto maduro y comercial.

Características especiales:

Creado por Object Design, es un sistema manejador de bases de datos para aplicaciones escritas en C++. Dirigido a desarrolladores con soluciones interactivas, proporciona funcionalidad completa para ambientes distribuidos que corran en plataformas Windows.

Este SMBD orientado a objetos proporciona manejo de versiones y control de concurrencia colaborativa para grupos de trabajo. Manipula bases de datos distribuidas con múltiples clientes y servidores. El manejo del conjunto de datos permite concentrarse en aumentar la funcionalidad de las aplicaciones. Actualmente proporciona configuraciones distribuidas, control de concurrencia, de acceso, capacidades de consulta, recuperación y administración de la base de datos.

Corre sobre la mayoría de las plataformas UNIX. Contempla datos persistentes que aparecen como si transitaran en la aplicación. Por tanto, no se necesitan instrucciones adicionales para referirse a los objetos persistentes. De esta manera la persistencia es independiente del tipo empleando un mapeo virtual de memoria.

Object Store soporta concurrencia. La integridad referencial se soporta a través de relaciones compuestas entre dos o más datos. Al declarar dos objetos referenciados uno con el otro, la integridad es forzada automáticamente como una dependencia de actualización Si un objeto se borra, el apuntador relacional inverso también se elimina.

Una de las características especiales de este sistema es el manejo de colecciones. Una colección es una agrupación de objetos que proporciona un medio conveniente de almacenar y manipular grupos de objetos ordenados o desordenados. Las consultas aprovechan el manejo de colecciones. Todas las consultas son no procedurales, no navegacionales y libres de tipos. Las expresiones de las consultas son predicados que pueden llevar a consultas anidadas.

Es posible agregar o cambiar tipos, atributos, métodos o definición de asociaciones. El esquema de la base de datos de ObjectStore se actualiza dinámicamente para reflejar estos cambios o adiciones. Cuenta con un visualizador (browser) que permite en forma gráfica inspeccionar el contenido de las bases de datos.

Proporciona una opción de lenguajes. Se soporta C++ a través de lenguajes DDL/DML de alto nivel incrustables (embedded) o una biblioteca de funciones API. Los lenguajes DDL/DML se manejan a través de un preprocesador. Soportan herencia múltiple, funciones virtuales y tipos parametrizados. Soporta mecanismos de notificación.

La persistencia que proporciona para los programas C++ es a través de una arquitectura de mapeo de memoria virtual. La idea principal del mapeo de memoria es la misma de la memoria virtual en los sistemas operativos. Los apuntadores de memoria principal se utilizan para apuntar a elementos de datos. Todos los datos residen en páginas. Estas páginas pueden estar en la memoria principal o en disco. Esto constituye una desventaja puesto que las referencias a los objetos son dependientes de la implementación, son físicas.

Una consulta típica de este sistema es:

```
empleados [ :salario > 100000 : ] ;
```

2.1.3 ONTOS

Usos principales: CAD/CAM.
Administración de versiones: Sí.
Recuperación: Sí.
Administración de transacciones: Sí
Herencia múltiple: Sí.
Concurrencia/bloqueo: Sí, 4 tipos de bloqueos.
Soporte de bases de datos distribuidas: Sí.
Evolución dinámica del esquema: Sí (algo limitada).
Datos multimedia: No.
Interfaz con lenguajes de programación: C++.
Estado actual de desarrollo: Ya es un producto maduro y comercial.

Características especiales:

Desarrollado por Ontologic of Billercia, es un producto que hace a C++ un lenguaje de bases de datos. ONTOS reemplaza al Vbase, de la misma compañía, cuyo modelo orientado a objetos trabajaba con los lenguajes propietarios COP y TDL ONTOS se basa en el poder y portabilidad de C++. Las transacciones se definen estáticamente en tiempo de compilación; no hay flexibilidad en tiempo de ejecución. Por otro lado, existe verdadero encapsulamiento (en formas variables privadas y protegidas de C++), y herencia múltiple. ONTOS serializa todas las transacciones concurrentes, ésto reduce concurrencia.

Una característica particular de ONTOS es que facilita referencias entre objetos totalmente diferentes a través de referencias directas en C++ y referencias abstractas. En el primer caso, el mantenimiento de las referencias se delega al programa; en el segundo, las referencias abstractas permiten mantener la referencia cuando el objeto es movido a la memoria principal sin que el programador deba especificarla. Está contemplado en el proyecto de desarrollo de una nueva versión el incluir la mayoría de las características que hoy son limitantes.

2.1.4 VERSANT

Usos principales: Ingeniería colaborativa.
Administración de versiones: Sí.
Recuperación: No.
Administración de transacciones: Sí.
Herencia múltiple: Sí.
Concurrencia/bloqueo: Bloqueo en dos fases, 4 tipos de bloqueos.
Soporte de bases de datos distribuidas: Sí.
Evolución dinámica del esquema: Sí (limitada).
Datos multimedia: No.
Interfaz con lenguajes de programación: C, C++.
Estado actual de desarrollo: Ya es un producto maduro y comercial.

Caracteristicas especiales:

Versant fue desarrollado por Versant Object Technology en Menlo Park, California. Proporciona administración de transacciones clasificándolas en cortas y largas. Las transacciones largas son persistentes, pueden terminar después de cualquier tiempo largo, y pueden incluir muchas sub-transacciones. Las transacciones cortas proporcionan un tipo de

bloqueo que permite compartir datos con otras transacciones cortas. Estas son atómicas y trabajan con un protocolo de grabación en dos fases (two phase commit) [1].

2.1.5 O_2

Usos principales: CAD/CAM, FIS, OIS, Procesamientos financieros y transacciones.
Administración de versiones: Limitada.
Recuperación: Sí.
Administración de transacciones: Sí.
Herencia múltiple: Sí.
Concurrencia/bloqueo: Sí, optimista.
Soporte de bases de datos distribuidas: Sí.
Evolución dinámica del esquema: Sí (limitada).
Datos multimedia: Sí.
Interfaz con lenguajes de programación: C.
Estado actual de desarrollo: Ya es un producto maduro y comercial.

Características especiales:

O_2 fue desarrollado por O_2 Technology, compañía francesa.

Modelo de datos: Tiene su propio modelo de datos. Fue diseñado para tener como anfitrión a cualquier lenguaje C y C++. Por tanto, la definición de operaciones se basa en un derivado del lenguaje C llamado O2C. Soporta la herencia múltiple y enlace tardío (late binding) porque los métodos se almacenan en la base de datos y se enlazan en tiempo de ejecución.

La persistencia se lleva a cabo insertando los objetos a través de un nombre dado predefinido como variables persistentes. Incluye un lenguaje de consultas similar al select de SQL; últimamente adopta los estándares definidos por la versión OQL establecida en el documento ODMG-93.

[1] Este tipo de actualización a dispositivo de almacenamiento consistente en un ambiente concurrente que envía una señal a todos los usuarios de la base de datos solicitando se confirme si se puede grabar delbúfer al disco (fase 1); y cuando la respuesta es afirmativa por parte del software agente de cada usuario, el servidor poseedor de la base de datos ejecuta el commit (actualización a disco, fase 2).

2.2 Sistemas relacionales con extensión de orientación a objetos o híbridos.

Los grandes fabricantes de sistemas basados (o parcialmente basados) en el modelo relacional, han decidido incorporar las características del modelo orientado a objetos a sus sistemas, entre los que tenemos:

2.2.1 Illustra.

No se cuenta con detalles esquematizados bajo el criterio seguido en los demás productos; pero, dado que este producto es de los más avanzados en bases de datos orientados a objetos, es necesario incluirlo. Es el nombre de la compañía y del producto: el servidor Illustra, que es una base de datos relacional con soporte interconstruido de características de orientación a objetos, especialmente extensión de tipos de datos. Illustra fue diseñado por el Dr. Michael Stonebraker, quien previamente desarrolló Ingres y Postgres. La compañía se fundó en agosto de 1992.

El producto soporta SQL-92, integridad referencial declarativa (es decir, soporta cláusulas en donde se declaran las llaves foráneas y la restricción que éstas implican en los objetos a los que hacen referencia), control de transacciones y sentencias de seguridad SQL (grant, revoke); así como respaldo y recuperación en línea.

Arquitectura: Incorpora la tecnología DataBlade, así como herencia múltiple y simple, funciones (métodos), conjuntos y arreglos, versiones e identificadores de objetos.

Lenguaje: El lenguaje de definición de datos de Illustra es más relacional que orientado a objetos dado que utiliza la sentencia CREATE TABLE; aunque la sintaxis de esta sentencia tiene más opciones que un sistema típico relacional. Sin embargo, las reglas de integridad están implícitas en las cláusulas UNIQUE, NOT NULL, PRIMARY KEY, DEFAULT y CHECK.

No siempre se almacena la información en tablas. Las columnas que contengan datos alfanuméricos existen normalmente en una tabla en el espacio de las bases de datos; pero los objetos complejos (tipo multimedia) se graban sobre archivos físicos manejados por el SMBD. Las columnas en una definición de tabla pueden declararse como tipos simples o como tipos definidos por el usuario predefinidos con la sentencia CREATE TYPE.

Con la cláusula UNDER las tablas pueden heredar definiciones de columnas de otras tablas. La sentencia SELECT de Illustra permite a los usuarios recuperar los registros a todos los niveles en la jerarquía de clases. El resultado no se da en tablas, sino en "renglones

engranados" (objetos en pantalla donde se seleccionan campos para ampliar su estructura y profundizar a cualquier nivel, en lugar de observar la información en forma de tabla). También se pueden crear métodos en los que regrese un valor que sea utilizado como columna. Incluye también indexación B-Tree.

Interfaz: La interfaz de Illustra está dada con aplicaciones (llamadas DataBlades) puestas a disposición del usuario como API. La mayoría de estos sistemas está escrito en C y C++; pero ofrece herramientas para algunos productos de Microsoft.

Desde hace pocos meses, el gigante de bases de datos Informix, adquirió todos los desarrollos de esta empresa para convertir su base de datos en un estándar en la industria (tomando como punto de partida el producto de Illustra), sujetándose además a las definiciones del lenguaje SQL3 de ANSI.

2.2.2 UniSQL

UniSQL es un sistema que pretende unificar las bases de datos relacionales y orientadas a objetos. Está diseñado para soportar desarrollo de aplicaciones en lenguajes de programación convencionales como anfitrión, como el C; o para lenguajes orientados a objetos (C++ o Smalltalk).

El modelo básico de UniSQL/x está diseñado para soportar el modelo de objetos adoptado por el consorcio Object Management Group. Este último modelo incluye conceptos de orientación a objetos como encapsulamiento de datos y métodos, identidad de objetos, herencia múltiple, tipos de datos abstractos y objetos anidados.

Es posible aprovechar los beneficios de este paradigma manteniendo todos los beneficios del modelo relacional y un superconjunto de ANSI SQL Existen tres ineficiencias típicas de este último modelo:

- Incapacidad de manejar tipos de datos arbitrarios
- Incapacidad de manejar estructuras complejas
- Incapacidad de manejar estructuras jerárquicas

El producto soporta vistas, optimización automática de consultas, manejo de transacciones, control de concurrencia, evolución de esquemas dinámicos, triggers, etc. Así mismo maneja tipos de datos multimedia.

Su desempeño es por lo menos equivalente al de sistemas relacionales.

Lenguaje: Soporta tres tipos de colecciones: conjuntos, multiconjuntos o secuencias. Las consultas en UniSQL utilizan la sentencia SELECT. Sin embargo, existen diferentes opciones para soportar las extensiones de objetos. Se pueden invocar métodos en casi todas las cláusulas salvo HAVING. Las listas de columnas pueden incluir atributos que regresen colecciones. Los atributos pueden ser acoplados (cast) de un tipo a otro. La inclusión del nombre de la clase en la lista de columnas regresa un identificador de objeto para dicha clase. La cláusula FROM hace referencia a clases en lugar de a tablas. La inclusión de una palabra clave adicional "ALL" en la cláusula FROM regresa todas las instancias de la clase referida, además de todas las instancias de las subclases dependientes (opera varias capas). Puede ejecutar comparaciones menor que, mayor que o igual sobre colecciones.

2.3 Sistemas Manejadores de Bases de Datos Orientados a Objetos puros.

Muchos de los productos actualmente desarrollados no tienen una base relacional ni son productos de lenguajes orientados a objetos · a los que se les haya incorporado características de persistencia. Más bien, este tipo de productos han sido pensados con el fin de resolver a fondo los problemas que ambos modelos presentan, aprovechando precisamente las bondades de los dos.

2.3.1 OpenODB (Iris)

Usos principales: OIS, CAD/CAM, Sistemas basados en conocimiento.
Administración de versiones: Sí.
Recuperación: Vía HP-SQL.
Administración de transacciones: Sí.
Herencia múltiple: Restringida.
Concurrencia/bloqueo: Sí.
Soporte de bases de datos distribuidas: No.
Evolución dinámica del esquema: Sí (limitada).
Datos multimedia: No.
Interfaz con lenguajes de programación: C, LISP, OSQL.
Estado actual de desarrollo: Aún es prototipo de investigación.

Características especiales:

Desarrollado por Hewllett-Packard. Es resultado de la investigación cuyo prototipo desarrollado fue llamado Iris. Utiliza una arquitectura cliente/servidor. El servidor es un administrador de almacenamiento. Una base de datos relacional, sobre la cual está una capa de un manejador de objetos. Con esto es factible la migración de tecnologías integrando la base de datos sin perderla.

Modelo de datos: Está basado en el modelo funcional de datos (OMT). En consecuencia, los atributos, las operaciones y las asociaciones están modeladas a través de funciones. Soporta tres tipos de funciones definidas por el usuario:

- Funciones almacenadas.
- Funciones basadas en su lenguaje OSQL que son derivadas o calculadas.
- Funciones externas referenciadas para el código externo de OpenODB.

Soporta bolsas, conjuntos y listas, así como herencia múltiple. Durante su ciclo de vida, los objetos pueden asociarse dinámicamente a diferentes tipos. Incluso un objeto puede pertenecer simultáneamente a varios tipos (incluso tipos no relacionados por la jerarquía sub/super).

Lenguaje de consulta:

El lenguaje de consultas de este producto se llama OSQL. Es un lenguaje computacionalmente completo con respecto a la base de datos OpenODB; especifica autorización para individuos o grupos, define transacciones, programas lógicos incrustados en funciones y administración de la base de datos.

OSQL incluye sentencias de flujo de programación: IF/THEN/ELSE, FOR, WHILE. Con este lenguaje procedural se puede modelar comportamiento complejo. OpenODB almacena lo mismo código que datos; por tanto, las aplicaciones y las reglas de negocios pueden definirse uniformemente

Manipula identidad de objetos con un OID proveído por el sistema. No es necesario crear llaves únicas. El lenguaje está basado en conjuntos. Soporta los conceptos de tipos

agregados, jerarquía de tipos, herencia múltiple, funciones sobrecargadas, enlace tardío, tipificación dinámica, indexación, agrupación (clustering) y encapsulamiento.

Componentes:

- Clientes OpenODB. SQL interactivo orientado a objetos.
- Visualizador gráfico.
- Interfaz de programación para C.
- Aplicaciones y herramientas de usuario.
- Manejador de objetos.
- Manejador de almacenamiento relacional y funciones externas.

2.3.2. Itasca

Usos principales: CAD/CAM, Inteligencia artificial, OIS, Información Multimedia.
Administración de versiones: Sí.
Recuperación: Combinación de bitácoras (log) y páginas replicadas (shadow pages).
Administración de transacciones: Sí.
Herencia múltiple: Sí.
Concurrencia/bloqueo: 5 tipos de bloqueo.
Soporte de bases de datos distribuidas: Sí (ORION-2).
Evolución dinámica del esquema: Sí.
Datos multimedia: Sí.
Interfaz con lenguajes de programación: LISP, C.
Estado actual de desarrollo: Ya es un producto maduro y comercial (ORION-2).

Características especiales:

Itasca comenzó como un sistema prototipo de bases de datos llamado ORION en 1985 en el programa de arquitectura de computadoras avanzadas del Microelectronics and Compute Technology Corporation (MCC), en sus laboratorios de sistemas distribuidos y sistemas orientados a objetos. El producto ITASCA se comercializa por ItascaSystems, Inc y es una extensión del prototipo ORION-2. Esta implementado en Common LISP.

Modelo de programación: El sistema extiende al Common LISP agregando orientación a objetos y capacidades de bases de datos. Dichas extensiones son:

- Creación / supresión de clases, subclases e instancias.
- Definición de métodos.
- Terminación de transacciones.
- Derivación de versiones.

Modelo de datos: Aunque ha sufrido estas adiciones, nunca ha sido un lenguaje completamente orientado a objetos como OPAL. Su modelo de datos es similar a OPAL. Las diferencias principales radican en la semántica de las clases. En Itasca son soportadas las clases intencionales y extensionales. Un identificador de objeto no se llama AOO como en GemStone, sino OID. Adicionalmente, este modelo incorpora nuevas características del modelo orientado a objetos:

- Objetos compuestos (estructurados recursivamente desde otros objetos).
- Los valores de los atributos son apuntadores a referencia de objetos.
- Contempla la referencia compuesta, que captura la semántica parte de (is part of).
- Los objetos compuestos se almacenan (y recuperan) en un espacio continuo de memoria.
- Proporciona manejo de versiones de objetos.
- Cambios de esquema (cambio, supresión o adición de clases de objetos, de atributos, de métodos, de la jerarquía de clases) en forma de invariantes.

Lenguaje de consulta:

Una expresión de consulta puede contener operadores de igualdad, desigualdad o de equivalencia. Soporta operadores existenciales.

Arquitectura del sistema:

Está basado en el modelo cliente/servidor. Es decir, existe un proceso servidor que espera peticiones de información monitoreando para responder a ellas. El subsistema de almacenamiento proporciona acceso al disco: asigna y desasigna páginas en memoria, sitúa los objetos en las páginas y mueve los objetos de las páginas al disco y viceversa. Cuenta también con un mecanismo de notificación por las actualizaciones de las bases de datos.

Interfaz: La interfaz básica de Itasca es el ambiente de Common LISP; pero cuenta con una serie de subsistemas auxiliares como un ambiente gráfico, editor de esquemas de objetos, editor de datos activos. Proporciona APIs para programación en C, C++ y LISP.

3. DIRECCIONES ACTUALES DE ESTÁNDARES.

La necesidad de una formalización de los conceptos del paradigma orientado a objetos dada por un estándar aceptado universalmente, ha originado el surgimiento de diferentes grupos o asociaciones que trabajan en la elaboración de especificaciones para el modelo orientado a objetos desde diferentes enfoques tales como lenguajes de programación, bases de datos, interfaces de usuarios, sistemas operativos, sistemas distribuidos, diseño y análisis de objetos, modelado de empresas y modelado de información

El comité X3H7 de la American National Standards Institute (Object Information Management) ha elaborado un reporte que incorpora los diferentes modelos y estándares hasta el momento desarrollados relativos al paradigma orientado a objetos. En este reporte se han incluido definiciones de grupos y sistemas por ejemplo:

- OMG (Object Management Group)
- ODMG (Object Database Management Group)
- EXPRESS (Lenguaje)
- Open Distribuited Processing
- Management Information Model
- SQL 3 (X3H2 ANSI)
- Matisse
- + (Lenguaje) (X3 J16 ANSI)
- OO COBOL (Lenguaje) (X3 J4 ANSI)
- SmaliTalk (Lenguaje) (X3 J20 ANSI)
- Eiffel (Lenguaje)
- System Object Model
- OLE 2 Component Object Model
- OSQL (HP Open ODB)

... entre otros más.

El reporte elaborado por el comité es presentado como una matriz teniendo como renglones los diferentes modelos propuestos y como columnas los diferentes aspectos analizados, mostrando:

- Conceptos básicos
- Objetos.
- Operaciones (mensajes, semántica de comportamiento, métodos, estado, tiempo de vida de los objetos, modelo de comunicación, eventos).
- Ligas (interfaz de aplicación para lenguajes de programación).
- Polimorfismo.
- Encapsulamiento de cada modelo.
- Identidad, igualdad y copia de objetos.
- Definición de tipos y clases en cada modelo.
- Herencia.
- Aspectos de los objetos (asociaciones, atributos, literales, contención, agregados, etc.).
- Extensibilidad.
- Lenguajes definidos en el modelo.
- Semántica de las clases base.

En este capítulo, se analizarán elementos cuya aplicación principal sean las bases de datos. Concretamente el modelo del grupo ODMG y el modelo de ANSI para el nuevo lenguaje SQL- 3. También se muestra un resumen de las características del modelo OMG que son la base tomada por el modelo ODMG.

3.1 OMG

La idea básica del modelo OMG es el objeto. Un objeto puede modelar cualquier entidad y una de sus características principales es la identidad, la cual es inmutable y persiste mientras exista el objeto además de ser independiente de las propiedades y comportamiento de éste

Dado que existen diferentes perfiles en los que se puede aplicar el modelo de objetos (objetos distribuidos, bases de datos de objetos, análisis y diseño, interfaz de usuario); OMG definió un conjunto de características básicas (Core Model) comunes a todos los perfiles.

Los objetos se crean como instancias de tipos. Un tipo describe el comportamiento de sus instancias describiendo las operaciones que pueden aplicarse a los objetos de dicho tipo Los tipos se relacionan entre sí a través del concepto de subtipo y supertipo. Un objeto que es una instancia de un tipo T implica que dicho objeto es del tipo T.

Operaciones: Una operación describe una acción que puede llevar a cabo un objeto con una serie de argumentos especificados. Una invocación de una operación es un evento. Un evento puede originar:

- Un conjunto inmediato de resultados.
- Efectos colaterales, manifestado en cambios del estado del objeto.
- Excepciones· Una excepción contiene información que indica que ha ocurrido algo inusual y proporciona información de lo ocurrido al manipulador de excepciones. (El modelo no especifica cómo manipular las excepciones).

Toda operación debe tener especificada un prototipo, que denota el nombre de la operación, una lista de parámetros con sus tipos correspondientes y una lista de valores resultantes también con sus tipos correspondientes. No existe especificación formal en el modelo respecto al orden o secuencia de ejecución de las operaciones. Pueden ser secuenciales o concurrentes, eso ya depende de la refinación en alguna especificación particular de un modelo derivado del OMG.

En este modelo las operaciones no son objetos. No requiere una especificación formal de la semántica de las operaciones pero resulta correcto incluir comentarios que especifiquen el propósito de cada operación, los efectos colaterales y las invariantes con las que cuenta.

El modelo básico OMG es un modelo clásico dado que las operaciones están definidas sobre tipos simples (un modelo cuyas operaciones no están definidas sobre tipos simples se llama modelo generalizado).

Tipos de objetos: Los tipos de objetos se ordenan en una estructura jerárquica donde el nodo raiz es el tipo objeto. Las aplicaciones introducen nuevos tipos subtipificando a Objeto. Al conjunto de todos los tipos se le llama Otipos.

La interfaz y el conjunto de instancias de un tipo pueden cambiar en el tiempo sin que cambie la identidad de ese tipo

Implementación: El modelo no especifica implementación de los objetos Cada aplicación o submodelo tendrá su propia especificación. La implementación de un tipo (codificación de sus características) representará una clase. Puede haber múltiples implementaciones para un tipo dado, dependiendo del dominio del problema.

3.2 ODMG-93

El grupo ODMG se inició a finales del 91 para atacar la carencia de un estándar en bases de datos orientadas a objetos. Este grupo está formado por algunas de las empresas que han desarrollado productos de bases de datos orientadas a objetos (en 1994 eran 13 empresas, entre las que se encuentra gente como R.G.G. Cattell, Mary E.S. Loomis de H.P., Illustra, Versant Technologies, Itasca, O_2 technologies).

El trabajo de ODMG es análogo al de ANSI para la redefinición de SQL con soporte de objetos, con la diferencia que ODMG no contaba con un lenguaje de-facto con qué empezar; teniendo que definir las características (incluyendo interfaces como C++ y SmallTalk) desde su raíz.

Por lo anterior, el primer documento de especificaciones ODMG-93, no cubre todas las áreas posibles de funcionalidad (no cubre bases de datos distribuidas, interoperabilidad entre versiones); pero cubre todas las características básicas para crear, modificar y compartir.

Modelo de datos.

El modelo de objetos de ODMG pretende otorgar portabilidad de aplicaciones a los productos de bases de datos de objetos. Proporciona un modelo común para estos productos definiendo extensiones del modelo de objetos OMG que soporta requerimientos de bases de datos de objetos. En particular, el modelo de objetos ODMG extiende el núcleo OMG para proveer objetos persistentes, propiedades de objetos, tipos de objetos más específicos, consultas y transacciones.

Un modelo común permite compartir datos entre lenguajes de programación. Los integrantes del grupo esperan que se convierta en un estándar de facto[1] a través del tiempo, y han acoplado sus productos y el lenguaje OQL para cumplir tal fin.

[1] Estándar de facto: el estándar que la industria impone con su comercialización, aunque no exista una organización oficial (como ANSI, ISO, IEEE, etc.) que haya proporcionado o especificado sus características.

ODMG-93 difiere del trabajo actual ANSI SQL3 en varios aspectos importantes. El estándar ODMG-93 busca la integración, de forma transparente, entre los lenguajes de programación y los sistemas de bases de datos con el objeto de lograr compatibilidad global de las aplicaciones, no sólo de las partes escritas en el lenguaje de consulta, y evitar el aprendizaje de un lenguaje de manipulación de datos, además de solucionar los problemas de desajuste entre la representación de los datos en los lenguajes de programación y en las bases de datos. Desde el punto de vista del programador, ODMG extiende su lenguaje orientado a objetos con persistencia, manejo de transacciones, capacidad para manejo de consulta y la posibilidad de compartir objetos entre aplicaciones que se ejecutan concurrentemente.

Para trabajar con un sistema manejador de bases de datos orientado a objetos (SMBDOO) el programador proporciona un programa fuente para su aplicación además de escribir declaraciones de datos e interfaces para el esquema de la misma. El esquema puede escribirse en un lenguaje para definir objetos (ODL) y el programa fuente en un lenguaje de programación que tenga extensiones para procesar transacciones y consultas. Las declaraciones y el programa fuente se compilan y ligan en el SMBDOO para producir una aplicación ejecutable. La aplicación accede bases de datos nuevas o existentes cuyos tipos forman parte en las declaraciones.

Conceptos básicos: En este modelo los conceptos básicos son los objetos, tipos, operaciones, propiedades identidad y subtipos, estado (definido por los valores de sus propiedades), comportamiento (definido por operaciones) e identidad. Todos los objetos del mismo tipo tienen comportamiento y propiedades en común.

Objetos: Cada entidad del mundo real es modelada como un objeto al cual se asocia un estado y un comportamiento. El estado se define por los valores de las propiedades y el comportamiento por las operaciones que actúan sobre el estado del objeto. Los objetos se categorizan por tipos. Una base de datos almacena objetos, con la posibilidad de ser compartidos por múltiples aplicaciones y usuarios.

El estado de los objetos puede modificarse durante su vida o bien puede permanecer inmutable. En el primer caso se habla de objetos mutables y en el segundo de objetos inmutables o literales. Otra forma de clasificar a los elementos del modelo de objetos es mediante su estructura: los objetos pueden ser atómicos (definidos por el usuario) o bien de tipo *colección*.

Una colección tiene un número arbitrario de elementos del mismo tipo, por ejemplo, una lista de empleados, o un conjunto de discos. Las colecciones pueden ser arreglos, listas o colecciones de elementos cuyo orden es irrelevante y con la posibilidad de tener valores repetidos (sacos) o sin repetir (conjuntos). En las listas se incluyen las cadenas de bits y las de

caracteres Los objetos estructurados pueden componerse libremente, por ejemplo, se pueden tener conjuntos de listas, listas de listas, arreglos de sacos, etc. De esta forma es factible, por ejemplo, definir un objeto *Diccionario* como un conjunto ordenado de estructuras, cada una con dos campos: una clave y un identificador.

Las literales u objetos inmutables pueden ser atómicas, colecciones o estructuradas. Una literal atómica puede ser un número entero o real, un carácter, un octeto, un booleano, una cadena o una enumeración. Las colecciones inmutables pueden ser conjuntos, sacos, arreglos o listas. En las literales estructuradas se encuentran la fecha, hora, marca de tiempo (timestamp) e intervalo definidas como en la especificación del ANSI SQL.

Identidad: Una característica básica de cualquier objeto es su identidad, la cual es inmutable e independiente de sus propiedades y comportamiento. Todos los elementos del modelo de objetos tienen identidad, cuya representación es distinta en las literales y en los objetos. La identidad de toda literal está dada por la sucesión de bits que codifican su valor. La identidad de un objeto se implementa por medio de un identificador único llamado *Object identifier* (en adelante OID) que es generado por el sistema. En ODMG no se especifica ni el tamaño ni la estructura interna de un OID.

El concepto de identidad es opuesto al de clave del modelo relacional, en donde una nada se identifica por el valor de su clave que es un elemento del estado de la misma. De manera que una clave es única dentro de su relación en tanto que el OID es único en la base de datos.

Mediante los OID's es posible construir objetos complejos debido a que pueden identificar los diversos componentes del objeto complejo. Mediante los OID's los objetos se pueden compartir permitiendo distinguir entre dos objetos que tienen como componente objetos que son iguales y dos objetos que tienen el mismo OID (es decir, son el mismo objeto), y son iguales si los valores de todos sus atributos son iguales. Por tanto, dos objetos idénticos mientras que el contrario no es verdadero.

Además del OID asignado por el sistema, el programador puede proveer de un nombre descriptivo a cada objeto que utilice en sus programas. El SMBDOO proporciona una función para mapear un nombre de objeto a su OID. Tal nombre debe ser único en la base de datos.

Estado: El estado del objeto se define por el valor de las propiedades del mismo, éstas pueden ser atributos del objeto en sí o relaciones entre dicho objeto y uno o varios objetos distintos. El valor de cualquier atributo es una literal (o conjunto de literales), que puede accederse por un par de funciones predefinidas. Los atributos no pueden tener propiedades ni

ser subtipados, sin embargo, las operaciones mencionadas pueden reescribirse para darles más funcionalidad, por ejemplo haciendo validaciones antes de asignar un valor.

Una relación (*relationship*) es un tipo de propiedad definida entre dos objetos. Se permiten relaciones binarias con funcionalidad uno-uno, uno-varios, varios-varios. Las relaciones en sí mismas no tienen nombre, son los enlaces para cada dirección los que deben tenerla Las relaciones mantienen la integridad de referencias cruzadas. Si se elimina un objeto de una relación cualquier intento posterior de participación en ella produce una excepción. Las instancias de las relaciones no tienen OIDs, se identifican por los objetos que participan en ellas.

Comportamiento: El comportamiento potencial de un objeto se define por el conjunto de operaciones que pueden ejecutarse sobre él. Las operaciones se definen sobre un solo tipo. Los nombres de operaciones pueden repetirse siempre y cuando ellas estén definidas sobre diferentes tipos, al invocar una operación sobrecargada se elige aquella que concuerde con el tipo de objeto proporcionado como primer argumento puede tener varios tipos, en este caso se selecciona la operación definida sobre el tipo más específico.

El modelo de objetos proporciona manejadores de excepciones anidados dinámicamente, mediante un modelo de terminación. Las excepciones son, en sí mismas, objetos y pueden organizarse en una jerarquía. El SMBDOO debe proporcionar un tipo raíz llamado *exception* con una operación para imprimir un mensaje notificando que ocurrió una excepción de cierto tipo y finalizar el proceso. La información sobre la causa de la excepción o el contexto donde ocurrió se devuelve al manejador de excepciones como propiedad del objeto excepción.

Tiempo de vida: El tiempo de vida de un objeto se especifica en la creación del mismo y no puede modificarse en ningún momento. El modelo de objetos contempla dos tiempos de vida para objetos, éstos son:

- Transiente. Significa que el objeto será almacenado por el lenguaje de programación en tiempo de ejecución, generalmente se declara en el encabezado de un procedimiento y por tanto vive sólo durante la ejecución del mismo.
- Persistente: Significa que el objeto será manejado por el SMBD y que existirá aún después de que el procedimiento o proceso que lo creó haya terminado.

El tiempo de vida de un objeto es independiente de su tipo, por tanto un tipo puede tener algunas instancias persistentes y otras transientes y ambas pueden manipularse usando las mismas operaciones.

Tipos y clases: Los objetos pueden categorizarse en tipos. Un tipo define el conjunto de propiedades a través de las cuales sus instancias pueden interrogar y en algunos casos manipular directamente su estado. Todos los objetos se crean como instancias de su tipo y éste no puede cambiar. El conjunto de todas las instancias de un tipo se llama extensión.

Todo tipo tiene una interfaz dentro de una BD y una o más implementaciones aunque sólo una implementación puede usarse en un programa particular. En la interfaz se define el estado visible y el comportamiento de las instancias del tipo. El estado se define mediante prototipos para atributos (nombre y tipo) y para relaciones (el tipo del otro objeto involucrado en la relación y el nombre del enlace). El comportamiento se define como un conjunto de prototipos de operaciones. La implementación de una operación se conoce como método.

Los atributos definen estados abstractos, por ello están dentro de la definición de la interfaz y no en la implementación, sin embargo, esto no implica que un atributo tenga que estar implementado como un campo dentro de una estructura de datos. La implementación de un tipo consta de una representación (conjunto de estructuras de datos) y de varios métodos (cuerpos de procedimientos), uno por cada operación definida en la especificación. Los métodos implementan el comportamiento especificado por su operación asociada modificando la representación del objeto o invocando las operaciones definidas sobre objetos relacionados. Puede haber métodos y estructuras de datos que no tengan contraparte en la interfaz por ser locales y que sólo son empleados para la implementación de los métodos de la interfaz.

Una clase es la combinación de: la especificación de la interfaz de un tipo y una de las implementaciones del mismo. Un objeto individual es instancia de una clase. La especificación de una clase se usa para implementar todas las instancias del tipo. El modelo de objetos no especifica si un objeto debe conservar la implementación elegida o si puede cambiarla.

El modelo de objetos es fuertemente tipado: cada objeto o literal tiene tipo y cada operación requiere el tipo de sus operandos. Dos objetos o literales tienen el mismo tipo si son instancias del mismo tipo. Los tipos se organizan en jerarquías, el subtipado es una relación entre tipos que define las reglas para que objetos de un tipo sean aceptados en el contexto de otro tipo. Un objeto del tipo S (subtipo de T) puede asignarse a un objeto del tipo T, pero el caso contrario no es posible. Las instancias de un subtipo cuentan con las propiedades (estado y comportamiento) del supertipo así como propiedades únicas del subtipo debidos a su

naturaleza especializada. Los subtipos pueden tener varios padres, con la implicación que una instancia de un tipo S también lo es de todos sus supertipos.

Algunos tipos son directamente instanciables, otros no. Los denominados tipos abstractos definen las características heredadas por sus subtipos sin definir una implementación, por tanto no pueden ser directamente instanciables. Un subtipo instanciable de un tipo abstracto debe definir una implementación que soporte cada característica heredada de sus supertipos abstractos.

Cada base de datos es instancia del tipo *Database* que tiene operaciones para abrir una base de datos, cerrarla, determinar si está vacía o no, y para buscar algún objeto, además de soportar operaciones diseñadas por el administrador de la base de datos.

3.3 ANSI - SQL3.

Los comités ANSI X3H2 e ISO DBL actualmente (y en los últimos 3 años) están trabajando en un nuevo estándar para el lenguaje SQL llamado SQL3. Contemplará compatibilidad con SQL-93 y se espera terminarlo en 1997.

SQL3 extiende las capacidades de SQL-92 principalmente agregando un sistema de tipos extendible, orientado a objetos; dado que SQL-92 no ofrece la facilidad de definir tipos de datos complejos. SQL 3 ofrece esta capacidad basándose en la noción de los tipos de datos abstractos: TDA's (Abstract Data Types) que permiten al usuario capturar como parte de la base de datos, el comportamiento específico de la aplicación. Un TDA está completamente encapsulado y sólo puede observarse su comportamiento fuera de la definición de tipos; es decir, no está visible la implementación.

Existen diferentes niveles de encapsulamiento para un atributo o rutina, que puede ser PUBLICO, PRIVADO o PROTEGIDO. Los TDA's pueden implementarse utilizando las extensiones procedurales de SQL-3 o con lenguajes externos; así mismo pueden contener operadores unarios o binarios sobre sus instancias.

Los TDA's pueden ser tanto objetos identificadores únicos, así como valores cuyas instancias no tienen un identificador; y pueden estar interrelacionados en subtipos y supertipos soportando herencia múltiple. Se soporta enlace dinámico (en tiempo de ejecución) y verificación de tipos en tiempo de compilación. Una familia de tipos puede definirse con TDA's parametrizados. Existen tipos de colecciones predefinidas como CONJUNTOS, MULTICONJUNTOS y LISTAS

Los TDAs pueden utilizarse como tipos de variables, parámetros de rutinas, atributos de mismos TDAs, o de columnas o tablas. Las instancias persistentes de los TDA's pueden almacenarse en columnas de tablas y pueden consultarse con el SQL normal. Las consultas pueden llevarse a cabo sobre atributos y funciones sobre instancias TDA utilizando notación de trayectos (ramificaciones establecidas a través de separaciones de puntos de los elementos y subelementos que componen el trayecto). Se está trabajando sobre la inclusión de tipos de arreglos y tuplas como TDA's.

Modelo de datos. La especificación actual de SQL soporta tipos de datos abstractos (TDAs) que incluyen métodos, identificadores de objetos, subtipos y herencia, polimorfismo e integración con lenguajes externos. También pueden definirse tablas (relaciones) en este lenguaje, incluyendo tipos e identificadores de renglones, y un mecanismo de herencia.

SQL3 será un lenguaje completo computacionalmente para la creación, manejo y consulta de objetos persistentes.

Conceptos básicos: Las extensiones de objetos de este lenguaje proporcionan compatibilidad tanto para el tipo como la tabla. Los TDA's pueden utilizarse como los tipos preconstruidos. Las operaciones que definen la igualdad y ordenación de los TDA's están encapsuladas en la definición de éstos. Existen atributos derivados implementados a través de procedimientos llamados rutinas. Se incorpora el identificador de renglón (aceptado por ANSI pero todavía no por ISO) que permitirá mantener un identificador único y ser utilizado por las aplicaciones de RDBMS; pudiendo utilizarse como valor para una columna o como llave foránea. Esta capacidad reducirá la diferencia entre los renglones SQL y las instancias en el modelo convencional de objetos.

Se incorpora el tipo renglón definible por cada tabla, al cual pertenecen las tuplas de las tablas. Las propiedades de este tipo son los campos. Dos tipos renglón son equivalentes y pueden intercambiarse instancias si tienen el mismo número de propiedades en la misma posición y del mismo tipo[2]. Esta característica permitirá interactuar con los elementos de las tablas y variables de memoria, así como regresar renglones de tablas como resultado de funciones. Incluso, pueden definirse dominios[3] y columnas en base a los tipos renglón de cada tabla.

[2] Equivalente a la compatibilidad en la unión del modelo relacional.

[3] Nuevamente en este punto se hace referencia a la ecuación clase = dominio y clase = relación; donde esta última es válida siempre que se agregue una tercera: relación = dominio.

Objetos: Como se mencionó SQL3 permitirá la definición de TDA's utilizando tipos preconstruidos o reutilizando otros TDA's definidos por el usuario; y las columnas de las tablas relacionales pueden tomar sus valores de los dominios correspondientes a estos TDA's.

Existen dos tipos de TDA's, TDA's objetos y TDA's valores[4]. Las instancias de los primeros tendrán un OID que lo identifique de manera única (los segundos se identifican por los valores de sus atributos). Este OID se mapeará al identificador de renglón en las tablas relacionales. Existe un tipo OID diferente por cada tipo TDA. Dos OID's iguales se refieren al mismo objeto. Si se incluye la cláusula WITH OID VISIBLE, será posible utilizar este atributo en el contexto de la programación (como parámetro de funciones, por ejemplo).

Es posible especificar TDA's como partes de otros TDA's o como columnas de tablas; teniendo así mismo la capacidad de incorporarlos completamente o sólo referirse a ellos a través de sus OID's. Para ello se incorpora la cláusula INSTANCE en la definición de atributos de un TDA persona dentro del cual exista un atributo del mismo TDA persona; si esta definición se lleva a cabo con la cláusula INSTANCE, pueden modificarse directamente atributos de este subobjeto en una instancia sin que se actualice en el objeto original asociado. Esto es: un dato X del tipo T que incluye un atributo Y también de este tipo T; si se modifica directamente un atributo z (que ambos tienen) a través del primer dato:

X.Y.z := algo;

en lugar de hacerlo directamente:

Y.z := algo;

puede redundar en no estar afectando al mismo objeto Y.

SQL3 proporciona compatibilidad para la definición de tablas del modelo relacional; permitiendo en la cláusula CREATE TABLE la definición de tablas de tres tipos:

- SET: Tablas con elementos únicos.
- MULTISET: Tablas típicas de SQL-92.
- LIST: Tablas con un orden especificado.

Así mismo, las tablas están asociadas a un tipo renglón. La única forma de dar persistencia a una instancia de un TDA es asociándolo a un renglón de una tabla en la base de

[4] Los primeros equivalen a los objetos mutables y los segundos a los inmutables en el modelo ODMG.

datos. Además, un identificador de renglón puede utilizarse como llave foránea La inclusión de identificadores de renglones como argumentos en las rutinas, permiten asociar éstas a tablas para implementar operaciones tipo-objeto entre renglones, y rutinas más especializadas con subtablas para soportar polimorfismo.

Operaciones: La implementación de un TDA está dada por código que puede almacenarse como datos SQL. Las rutinas asociadas con TDA's incluyen la definición de FUNCIONES que permitan programar comportamiento definido por el usuario específico de los tipos. Las funciones pueden ser escritas en SQL o llamadas a funciones externas definidas en lenguajes estándares.

Una rutina se especifica dando su nombre, parámetro y puede ser:

- FUNCION: regrese un valor. Incluye una cláusula RETURN que especifica el tipo de valor regresado y puede ser:
    - DESTRUCTOR: destruye instancias TDA.
    - ACTOR: Accesa para leer o actualiza componentes de instancias TDA.
- PROCEDIMIENTO: No regresa valor.

La lista de parámetros en una rutina consta del nombre del parámetro y su tipo, además de una cláusula IN, OUT o INOUT.

Invocación: Un TDA debe tener operaciones ACTOR con prototipo. Se invocan utilizando una notación funcional, que también se utiliza para invocar rutinas asociadas a tablas.

Métodos: Si una rutina es completamente definida en SQL, puede incluir sentencias compuestas y sentencias de control; SQL3 ha sido enriquecido con nuevas sentencias para convertirlo en un lenguaje computacionalmente completo de manera que pueda especificarse totalmente el comportamiento de objetos.

Estado: El estado de un TDA lo constituyen la secuencia ordenada de sus componentes sin incluir el OID. SQL3 incorpora el concepto de objetos sin estado; que no son otra cosa que los renglones de las tablas.

Tiempo de vida de un objeto: Los TDA's existen durante el tiempo de ejecución pero no pueden almacenarse directamente en dispositivo secundario. Para tal efecto es necesario asociarlos a los renglones de una tabla Así mismo, no es posible darles nombre, de manera que no puede consultarse directamente. El usuario debe explícitamente definir un objeto persistente (tabla) como valores de columnas, en el cual residirán los TDA's sobre los que se desee hacer consultas.

Polimorfismo: SQL3 soporta la definición de rutinas con el mismo nombre en diferentes TDA's, además de la redefinición de rutinas heredadas en un subtipo. Esto es cierto tanto para los TDA's como para las operaciones en tablas. La forma en que SQL3 resuelva la rutina a ejecutar depende de los tipos de todos los parámetros especificados.

Encapsulamiento: SQL3 maneja tres tipos de encapsulamiento:

- PUBLICO: Componentes que constituyen la interfaz del TDA y son visibles para todos los usuarios autorizados del TDA específico.
- PRIVADO: Componentes encapsulados totalmente, que sólo son visibles dentro de la definición del TDA.
- PROTEGIDO: Componentes encapsulados parcialmente visibles dentro del propio TDA y en la definición de los subtipos heredados del TDA.

Tipos y clases: Existen dos tipos básicos soportados por SQL3:

- Tipos TDA
- Tipos de renglones

Las tablas con identificador de renglón equivalen de alguna forma a los tipos de objetos. SQL3 soporta tipos primitivos atómicos predefinidos (char, int, float, etc.); así como colecciones predefinidas. Así mismo incorpora tres tipos de tablas: SET, MULTISET y LIST.

Herencia: La herencia de TDA's es soportada con la cláusula UNDER. Existe un tipo mínimo específico que corresponde al más bajo nivel de subtipos asignados a una instancia. Corresponde a la definición de una clase no abstracta que herede de superclases probablemente abstractas.

Un subtipo tiene acceso a la representación de todos sus supertipos pero no de los tipos "hermanos". Puede cambiar los nombres de rutinas o propiedades que existen en más de uno de sus supertipos, además de poder redefinirlas.

Las tablas también pueden definirse como subtablas agregando atributos. Las subtablas y supertablas son independientes de los subtipos en los TDAs. Las reglas del DML (insert, update, delete) se definen de manera que mantengan los renglones de las tablas en forma consciente entre tablas y supertablas.

Asociaciones: En SQL3 puede definirse la integridad referencial al igual que en SQL-92; pero además en SQL3 pueden declararse datos cuyo tipo corresponde a un identificador de elementos de otra tabla, en lugar de utilizar una llave foránea correspondiente a la llave primaria de la segunda tabla. En este caso, el OID al que se hace referencia constituye la llave foránea.

Atributos: En SQL3 existen dos tipos de atributos para los TDAs:

- Almacenados: Atributos de cualquier tipo (incluso otros TDA's). Cada atributo declarado genera implícitamente atributos para visualizar y modificar (get, set) sus valores (aunque estos atributos pueden recodificarse).
- Virtuales: Son atributos a los que se les asocia una operación (atributos calculados).

Los atributos pueden designarse como:

- Actualizables: Se genera una operación set en forma implícita.
- Sólo lectura: Sólo pueden verse pero no modificarse (no se genera operación set).
- Constantes: En la definición del TDA se inicializan y no pueden modificar su valor después.

Las columnas de las tablas representan también atributos.

Extensibilidad: Se pueden definir nuevas tablas y nuevos TDA's con base en los ya existentes. Los esquemas pueden modificarse aplicando sentencias ALTER y DROP para columnas, tablas, supertablas, subtablas, tipos de tablas, etc. En SQL3 no existe la metaclase.

Lenguajes de objetos: Se han adicionado una serie de cláusulas nuevas para convertir al SQL3 en un lenguaje computacionalmente completo

- DESTROY: Elimina los objetos TDA existentes y debe estar en una función DESTRUCTOR.
- ASSIGNMENT. Permite asignar el resultado de una expresión SQL a una variable local, una columna o atributo de un TDA.
- CALL: Permite invocar procedimientos SQL.
- RETURN: Permite regresar un resultado de una expresión SQL en una función SQL.
- CASE: Permite seleccionar alternativas de ejecución de acuerdo a valores opcionales.
- IF ... THEN ... ELSE ... ELSEIF: Permite seleccionar subbloques de ejecución basado en condiciones de expresiones lógicas.
- LOOP WHILE: Permite ejecutar un bloque de instrucciones SQL en forma repetitiva hasta cumplir la condición especificada en la cláusula WHILE.

Puede además crearse sentencias compuestas agrupando bloques de sentencias con sus propias variables locales y manejo de excepciones.

El estándar SQL-92 contenía mapeos a lenguajes anfitriones; pero en algunos tipos de datos (como el Timestamp que no contempla C++) era necesaria una operación de acoplamiento (CAST) de tipos. El SQL3 no contempla actualmente ningún tipo de interfaz a lenguajes como C++ o SmallTalk; pero se está trabajando en ese aspecto, aunque C++ guarda más similitud con el modelo de SQL3 que SmallTalk.

CARACTERISTICAS POR RESOLVER.

Aunque hasta la fecha se han definido y unificado muchos conceptos del paradigma orientado a objetos, aún existen características no soportadas o bien soportadas sólo por algunos productos, como:

- Algunos OODBMS requieren que el usuario maneje explícitamente los bloqueos del esquema concurrencia.

- La mayoría de los OODBMS no tienen implementado el control de autorización de usuarios.
- Se necesita un lenguaje de consulta unificado (OQL y SQL3 son un buen principio, pero ambos no son compatibles totalmente entre sí ni están completamente definidos).
- Se necesita la implementación de un optimizador de consultas.
- Se requieren algoritmos de procesamiento de consultas que evalúen la complejidad de éstas (join, conjuntos, subqueries, etc.).
- Se requieren métodos de acceso formalmente definidos (índices, árboles-B, etc.).
- Se requiere modelar un catálogo de estadísticas de acceso a objetos que permita implementar un optimizador de consultas.
- Se necesita definir un modelo consistente de concurrencia y bases distribuidas de objetos.
- Se debe definir formal y universalmente la semántica de cambios en los esquemas de datos (evolución de esquemas).
- Esta semántica debe extenderse también para objetos residentes en memoria.

Adicionalmente a todos estos problemas, Shiner define un problema particular del ODMG-93 que sus implementadores han dejado de lado. El estándar ODMG-93 ha sido publicado y puesto a disposición de los fabricantes de software precisamente para agilizar su comercialización y unificación. No han podido esperar a que la ANSI publique completa la especificación SQL3; y decidieron realizar su propio estándar.

Sin embargo, esta especificación está vinculada en cierta forma a una implementación no formal (SmallTalk y C++) dependiente de plataformas; de tal manera que el esquema global de una base de datos no está disponible para cualquier otro lenguaje si es creado siguiendo el modelo ODMG. Es cierto que C y C++ son lenguajes muy portables; pero las bibliotecas de funciones y clases de C++ no son compatibles en todas las plataformas. Con lo anterior, se hace necesaria la definición de acceso a bases de datos con interfaces, y para cada biblioteca implementada en una plataforma en particular. Esto redunda en una problemática mayor; porque la definición de un estándar formal se hace necesaria para enriquecer al modelo y hacerlo independiente de la plataforma, para ello debe existir un modelo formal para definir el esquema de la base de datos consistente en enunciar los atributos o características del modelo a través de una secuencia de estos atributos residentes en un formato de texto plano en el administrador de base de datos y que siga un protocolo común basado en especificar el nombre del atributo y el dominio al que pertenece.

4. EJEMPLOS DE SOFTWARE DE PRUEBA EXISTENTE EN EL MERCADO.

Las bases de datos orientadas a objetos ofrecen parte de la misma funcionalidad que los lenguajes orientados a objetos. Permiten la encapsulación dentro de objetos de los datos y métodos que actúan sobre ellos. Activan métodos a través del uso de la herencia. Además, las bases de datos orientadas a objetos ofrecen a las bases de datos tradicionales la funcionalidad de la que carecen los lenguajes orientados a objetos, como persistencia y participación.

Por ello en este capítulo se describirán brevemente algunos de los sistemas actualmente disponibles como productos comerciales, prototipos terminados o prototipos en desarrollo. Algunos aspectos de estos sistemas se describirán en su arquitectura general. De cada uno de los sistemas se presentarán las características más significativas desde el punto de vista del lenguaje y del modelo de datos a la vez que sus características operacionales más importantes. Desde el punto de vista de la arquitectura, la mayoría de los sistemas que se analizarán se basan en un enfoque cliente-servidor, donde el servidor es el que usualmente soporta las funciones de los SGBD. La mayor parte de estos sistemas tienen un entorno interactivo con herramientas basadas en interfaces gráficas para facilitar la interacción con el usuario. Los sistemas que se describirán incluyen a: GemStone, IRIS, Picquery, Postgres, Sim, Vbase, O_2, ObjectStore y Orion.

4.1 GEMSTONE.

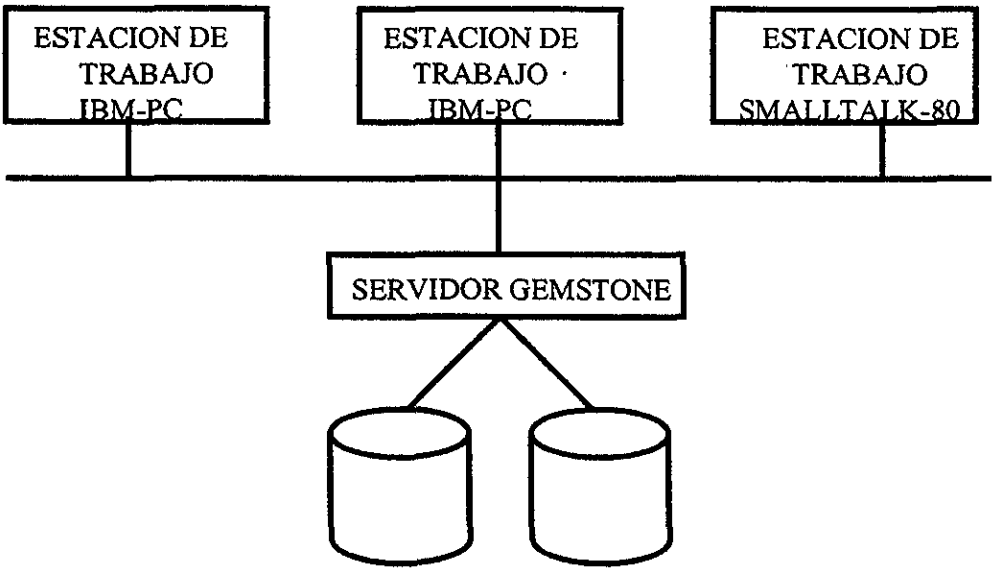
El sistema GemStone fue desarrollado por Servio Logic Development Corporation con el objetivo de ofrecer un SGBD caracterizado por un modelo de datos muy poderoso y, por consiguiente, reducir el tiempo requerido para desarrollar aplicaciones complejas. El sistema se basa en el entorno de SmallTalk-80, extendiendo éste para incluir funciones de los entornos de SGBD. Este tiene una arquitectura distribuida, que consiste de un conjunto de IBM-PC y/o estaciones de trabajo SmallTalk-80 y un Servidor de Objetos (Object Server) implementado sobre un sistema de archivos VAX/VMS, conectado a través de una red local

El Servidor también está disponible para entornos UNIX. El modelo de GemStone es en lo fundamental completamente idéntico al de SmallTalk-80, y se basa en los conceptos de objeto, clase y mensaje. Las clases se organizan en jerarquías con herencia simple. Las aplicaciones se pueden escribir en los lenguajes OPAL, C, C++ y PASCAL. OPAL es una extensión de SmallTalk-80. Este se usa como lenguaje de definición y manipulación de datos (LDD, LMD), como lenguaje general de programación, y como lenguaje de órdenes del

sistema. El OPE[1] es un conjunto de aplicaciones compatibles con Microsoft Windows, que incluyen:

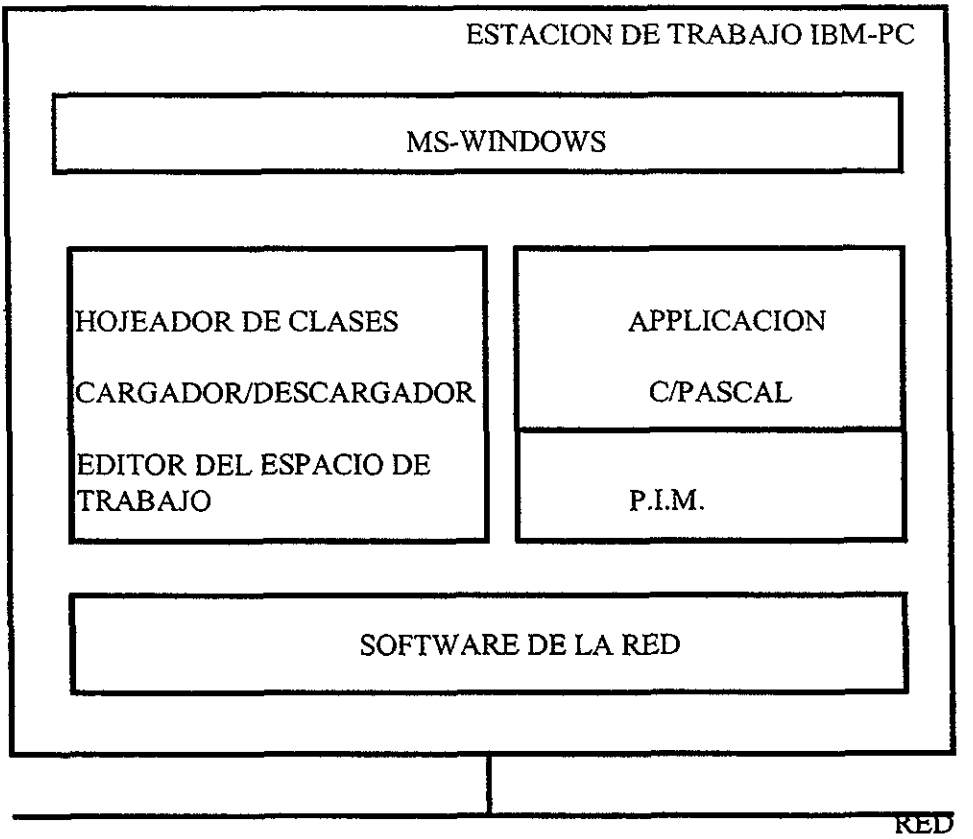
- Un hojeador de clases (Class Browser), que permite al usuario inspeccionar, añadir y/o modificar las definiciones de clases.
- Un cargador/descargador (Bull Loader/Dumper), con el cual los usuarios pueden transferir datos fomateados (registros de longitud fija) entre los archivos de PCs y el servidor GemStone.
- Un editor del espacio de trabajo (Workspace Editor), con el cual el usuario puede crear, editar y ejecutar expresiones OPAL.

ARQUITECTURA DE UN SISTEMA GEMSTONE.



[1] Del inglés OPAL Programming Environment (Entorno) de Programación de OPAL).

Para dar soporte a las aplicaciones escritas en lenguajes procedimentales, el sistema ofrece módulos objeto que se pueden llamar desde C, C++ y PASCAL, y que se denominan PIM[2], con los cuales el usuario <<conecta>> las aplicaciones que ejecuta sobre el PC. Estos módulos implementan llamadas a procedimientos remotos con las funciones que provee el Servidor de GemStone.

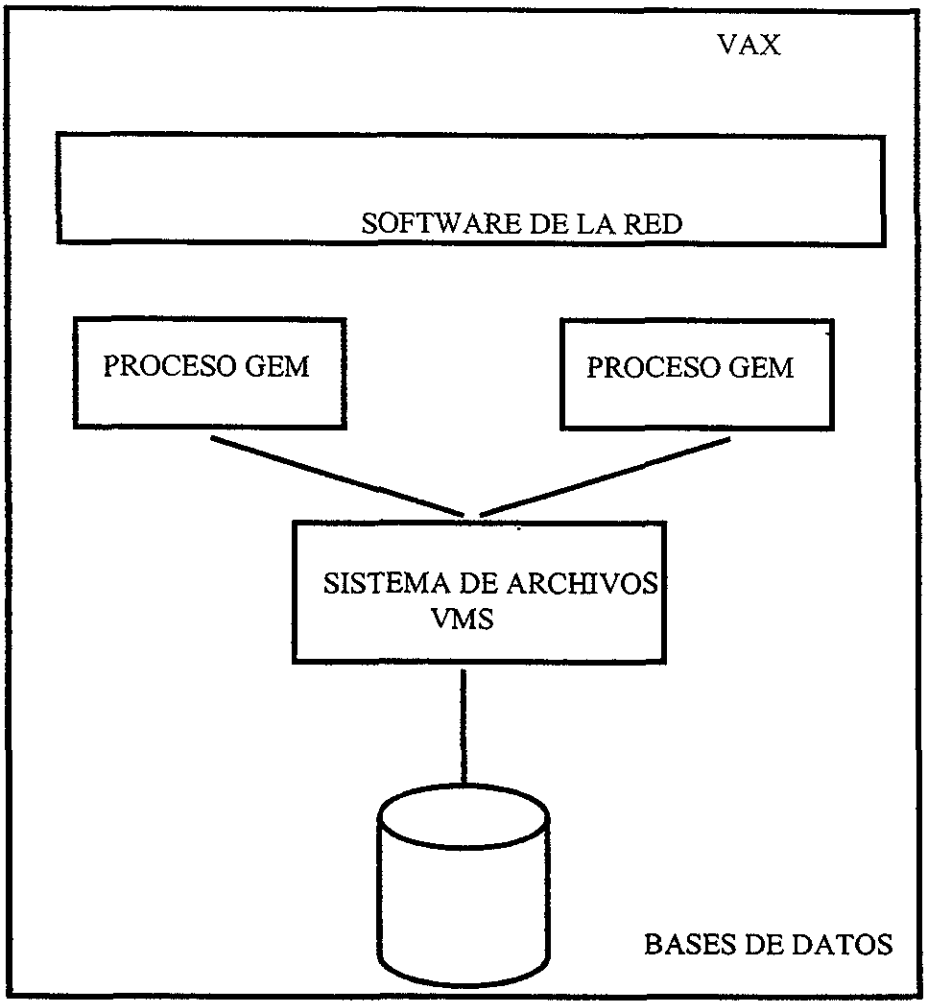


ARQUITECTURA CLIENTE EN GEMSTONE.

En el Servidor (centralizado) se pueden identificar las dos componentes siguientes:

- Gem implementa la memoria del objeto y la <<máquina virtual>> SmallTalk estándar. Gem compila y ejecuta los métodos escritos en OPAL y maneja el control de las sesiones y las autenticidades (verificaciones de autorización).

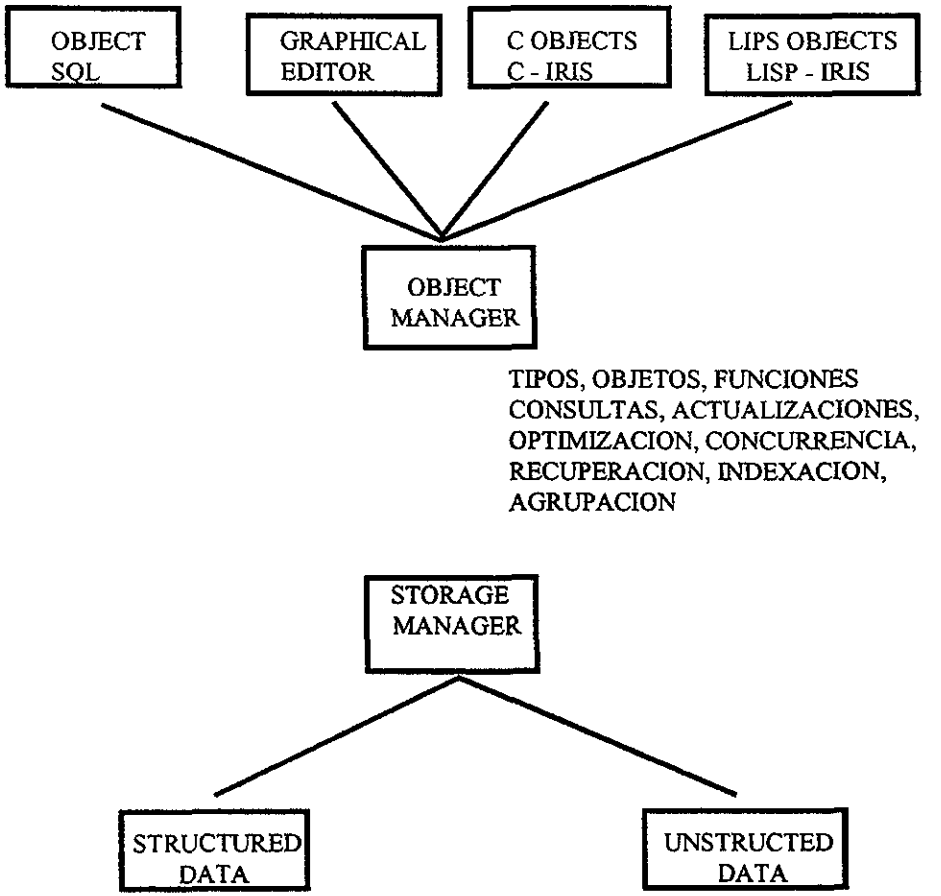
- Stone realiza la administración del almacenamiento secundario, el control de concurrencia, las autorizaciones, las transacciones, las recuperaciones ante caídas del sistema, y da soporte para el acceso asociativo. También administra los espacios de trabajo asociados con las sesiones.



ARQUITECTURA DEL SERVIDOR EN GEMSTONE.

4.2 IRIS.

Las capas que componen la arquitectura de Iris son las que se muestran en el siguiente esquema:



El centro de esta arquitectura es el administrador de Implementaciones de Objetos del Modelo de Datos Iris el cual cae dentro de la categoría general de los modelos orientados a objetos que soportan un alto nivel de abstracción estructural tal como clasificación, generalización / especialización y agregación tan bien como el comportamiento de abstracciones. El traductor del procesador de consultas a las consultas Iris y funciones dentro de un formato de álgebra relacional, esto es optimizado y luego interpretado otra vez en la base de datos almacenada. El manejador de Almacenamiento Iris es actualmente un

subsistema de almacenamiento relacional convencional. Esto suministra accesos asociativos y capacidad para actualizar una relación simple a un tiempo, incluyendo soporte de transacción.

Como otros sistemas de bases de datos, Iris es una accesible vía de una sola posición de interfaces interactivas o a través de módulos de interfaces empotradas en lenguajes de programación. Los módulos de interface, tales como esas etiquetas C-Iris y Lips-Iris, facilita accesos para la persistencia de objetos de varios lenguajes de programación. La construcción de interfaces es hecha posible por un conjunto de subrutinas definidas en lenguaje C. Al mismo tiempo, tres interfaces interactivas son soportadas. Una es simplemente un manejador de interface Administrador de Objetos. Otra interface interactiva, es SQL Objeto (OSQL) es una extensión de SQL de objetos orientados. Se ha escogido extender SQL en lugar de inventar totalmente otro lenguaje por la importancia de SQL en la comunidad de base de datos y porque se ha explorado la posibilidad de extensiones parecidas. Una tercera fase interactiva, es el editor gráfico, que permite al usuario explorar interactivamente el tipo de estructura Iris Metadata tan bien como la estructura Interobject Relationship definida sobre una base de datos Iris data. Esto es descrito en Object-C y soporta la actualización para esquemas y datos.

MANEJADOR DE OBJETOS IRIS.

El manejador de Objetos Iris implementa el modelo de datos por proporcionar soporte para la definición de esquemas, manipulación de datos y procesamiento de consulta. El modelo de datos, el cual esta basado en las tres construcciones, objetos, tipos y funciones que soportan herencia y propiedades genéricas, obligaciones, complejidades o datos no normalizados, funciones definidas por el usuario, control de versiones, inferencia y tipos de datos extendidos. Las rutas del modelo pueden ser encontradas en trabajos previos sobre DAPLEX, el Modelo de Datos Integrados, la extensión DAPLEX y el lenguaje TAXIS.

OBJETOS.

Los objetos representan entidades y conceptos de dominios de aplicaciones que han sido modelados. Ellos son entidades especiales en la base de datos con sus propias entidades y existencias, y ellos pueden ser referidos para ser indiferentes de sus valores de los atributos. Por ejemplo, cada objeto tiene asignado un amplio sistema, especialmente para identificar objetos, OID. Este soporte referencial es íntegro y representa una gran ventaja sobre los modelos de datos de registros orientados en los cuales los objetos son representados como registros, pueden ser referidos solamente en términos del valor de su atributo.

Los objetos son descritos por su comportamiento, y pueden sólo ser accesados y manipulados por el significado de sus funciones. Tan grande como la semántica de las

funciones similares que quedan, la base de datos puede ser físicamente tan bien como lógicamente reorganizada fuera de la afectación de la aplicación de programas. Esto suministra una gran cantidad de abstracciones de datos y de independencia de los datos.

Los objetos son clasificados dependiendo de su "tipo". Los objetos que pertenecen de un tipo similar comparten funciones comunes. Los tipos son organizados dentro de un tipo jerárquico con funciones heredadas. Consecuentemente un objeto puede tener tipos múltiples Los objetos sirven como argumentos para funciones y pueden ser aplicados como resultado de funciones. Una función puede ser aplicada para un objeto solamente si la función es definida en un tipo para el cual el objeto pertenece.

Las funciones pueden ser definidas incluyendo predicados y funciones de múltiples argumentos, proporcionando directamente soporte binario o n-ary relationship. El modelo de datos Iris se distingue entre objetos literales, tal como cadenas de caracteres y números, y objetos no literales tal como personas y departamentos. Los objetos no literales son representados internamente en la base de datos por identificadores de objetos. Los objetos literales no tienen accesibles el uso de identificadores de objetos y son directamente representables. Tal como ellos no pueden ser creados, destruidos o actualizados por los usuarios.

El manejador de objetos proporciona explícitamente sencillez creando y borrando objetos no literales, y para asignación y actualización de valores para sus funciones. La referencia íntegramente es soportada: cuando un objeto dado es borrado, todas las referencias para el objeto son borradas también.

TIPOS Y JERARQUIAS DE TIPOS.

Los tipos son nombrados colecciones de objetos. Los objetos pertenecen a un tipo igual que comparten funciones comunes. Por ejemplo, todos los objetos que pertenecen al tipo Persona tienen una función Nombre y una función Edad. Las funciones son calculadas, definidas en tipos. Ellas son aplicables a las instancias de los tipos. Los tipos son organizados en un tipo de estructura que soporta generalización y especialización. Un tipo puede ser declarado para ser un subtipo de otro tipo. En este caso, todas las instancias del supertipo. Esto significa que estas funciones son definidas en el supertipo y son también definidas en el subtipo.

4.3 O_2.

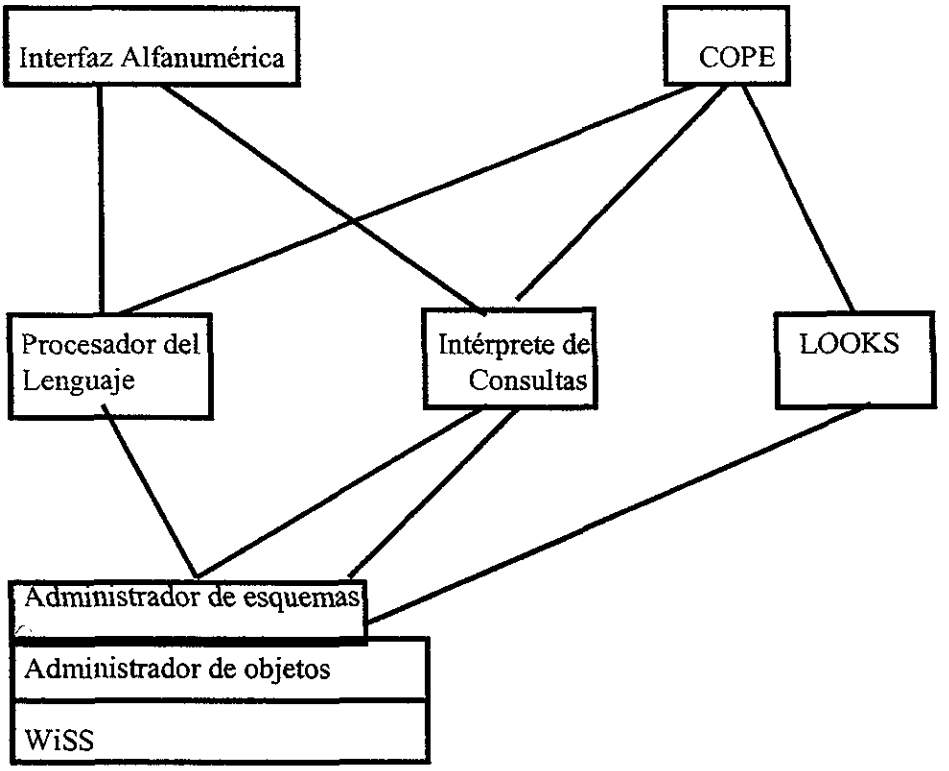
O_2 es un sistema desarrollado por el consorcio Altair. El sistema actual fue implementado en C, con algunas componentes desarrolladas en un dialecto de LISP sobre estaciones de trabajo Sun. El sistema se basa en una arquitectura cliente/servidor. El modelo de base de datos de O_2 es bastante flexible. Por ejemplo, se pueden definir no solamente objetos, sino también valores complejos; esto último mediante constructores de listas, tuplas y conjuntos. Las instancias de clases pueden tener métodos y atributos adicionales. Esto significa que es posible tener objetos con un comportamiento <<excepcional>> con respecto a la clase a la que pertenecen. El lenguaje que se ofrece para implementar los métodos es CO_2, una extensión de C con construcciones que pertenecen al modelo orientado a objetos de O_2. La persistencia de los objetos en O_2 es ortogonal al mecanismo de clases. Las instancias de una clase no son necesariamente persistentes. Inversamente, los valores complejos se pueden transformar en valores persistentes. Las aplicaciones se pueden programar tanto en CO_2 como en Basic O_2. Este último es una extensión del lenguaje Basic que incluye primitivas de O_2.

La arquitectura del sistema, está organizada en tres niveles:

- El nivel más alto es el nivel de administrador de esquemas. Las funciones que se dan a este nivel incluyen las de creación, acceso, modificación y eliminación de clases, métodos y nombres globales. El àdministrador de esquemas es también responsable del control de consistencia del esquema y de las reglas de verificación de subtipo sobre jerarquías de herencia.

- El nivel intermedio es el administrador de objetos. Esta componente maneja los objetos y los valores complejos independientemente de su persistencia. El administrador de objetos se encarga del intercambio de mensajes y una configuración cliente/servidor. Aún más, implementa todas las funciones relativas a la persistencia, recolección de basura, mecanismos de acceso, tales como índices, y agrupamiento (clustering). Por último, provee también todas las funciones para el manejo de transacciones.

- El nivel más bajo es el WiSS (Wisconsin Storage Subsystem), el cual maneja el almacenamiento secundario. WiSS provee funciones para la persistencia, la administración de disco y el control de concurrencia para los registros (es responsabilidad del administrador de objetos <<traducir>> los objetos complejos a registros WiSS).



Adicionalmente a estos niveles, O_2, ofrece a los usuarios una serie de entornos y herramientas:

- El procesador de lenguaje (Language Processor), que maneja todas las instrucciones del lenguaje para la definición de datos y compilación de programas.
- El procesador de consultas (Query Processor), que maneja todas las funciones de consulta
- El entorno de generación de interfaces (LOOKS), que provee una serie de herramientas para el desarrollo y manipulación de interfaces de bases de datos. LOOKS ofrece todas las primitivas para la evaluación de los objetos y los valores en la interfaz de usuario y trata la interacción de éstos con el administrador de objetos

- El entorno de programación OOPE[2], que ofrece una serie de herramientas para el desarrollo de aplicaciones. Utiliza los servicios que brinda LOOKS para el manejo de las interfaces.

- La interfaz alfanumérica, que brinda acceso directo a los diferentes lenguajes del sistema, sin necesidad de usar herramientas gráficas.

4.8 OBJECTSTORE.

ObjectStore fue diseñado para simplificar la conversión de las aplicaciones existentes y para áreas de aplicación tales como la modelación interactiva, y diseño y análisis asistido por computadoras. Uno de los objetivos específicos del sistema ha sido el de lograr altos niveles de rendimiento. Con vistas a lograr esto, se adoptó una arquitectura basada en memoria virtual con falta de páginas. Esta consiste en que cuando una aplicación se refiere a datos que no están en la memoria principal, ocurre una falta de página. ObjectStore intercepta la falla de página y carga en memoria principal el segmento de la base de datos que contiene los datos requeridos. Se debe notar que esta solución significa que el sistema debe ser altamente portable, puesto que se requiere una interacción con las capas más bajas del sistema operativo.

ObjectStore se basa en una arquitectura cliente/servidor. Los clientes son PC's de alto nivel y estaciones de trabajo, mientras que el servidor puede manejar diferentes arquitecturas sobre la misma red.

El sistema consta de:

- El SGBD ObjectStore.
- El ObjectStore ejecutor.
- El diseñador/hojeador (Designer/Browser) de esquemas.
- Un LMD construido como un preprocesador C++.

ObjectStore es accesible desde programas escritos en C y C++ con una biblioteca de interfaz, pero no se ofrece soporte directo para SQL. ObjectStore está actualmente disponible para plataformas Sun-3, Sun-4 y Sparcstation. Este sistema será tratado más a fondo en otro capitulo posterior.

[2] Del inglés Object Oriented Programming Environment (Entorno de Programación Orientada a Objetos).

3.5 ORION.

El sistema ORION fue diseñado y desarrollado dentro del programa de arquitectura avanzada de computadora (Advanced Computer Architecture) de Microelectronics and Computer Corporation (MCC) El sistema ha sido desarrollado con el objetivo de proporcionar un soporte directo al paradigma de la orientación a objetos. En éste se desarrollaron también avanzadas funciones para aplicaciones CAD y CAM, inteligencia artificial y sistemas de información. ORION ha sido utilizado para dotar de persistencia al sistema experto Proteus que fue desarrollado dentro del mismo programa de investigación. Algunas de las funciones más avanzadas suministradas por ORION incluyen soporte para versiones y notificaciones de cambio, evolución dinámica de esquemas, objetos compuestos y datos multimedia.

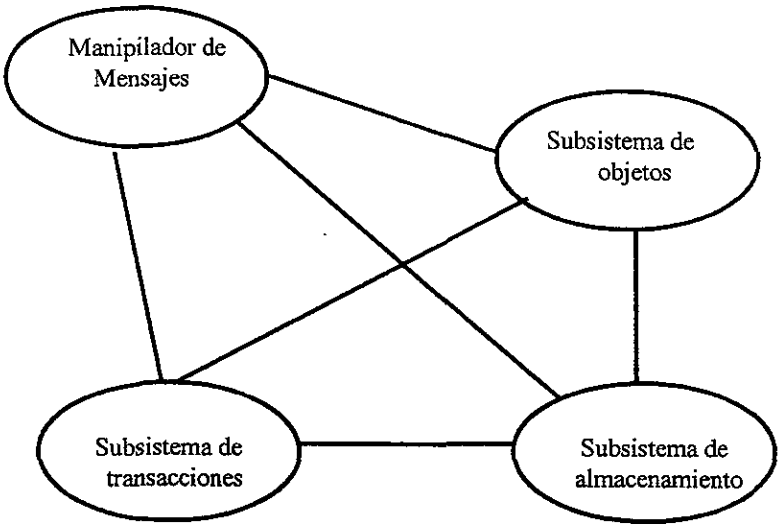
La primera versión de ORION fue un sistema de un único usuario, implementado sobre Common LISP sobre una máquina LISP Symbolic 3600, y trasladado posteriormente a una Sun bajo el sistema operativo UNIX. La arquitectura del sistema es la siguiente:

- El manipulador de mensajes (Message Handler) es responsable de manipular todos los mensajes que se envían al sistema ORION. Estos mensajes se utilizan para invocar métodos de acceso. Los métodos de acceso son para buscar o modificar los valores de los atributos. Los métodos del sistema brindan todas las funciones para el manejo de los datos, tales como definiciones de esquema, creación y eliminación de instancias y manejo de transacciones.

- El módulo subsistema de objetos (Object System) brinda el nivel más alto de funciones; por ejemplo, las modificaciones de esquema, el control de versiones, la optimización de las consultas y la manipulación de información multimedia.

- El subsistema de almacenamiento (Storage Subsystem) se ocupa del manejo de los objetos en disco. Se encarga de asignar y liberar los segmentos de páginas en el disco, determina la localización de los objetos en las páginas y transfiere las páginas de disco a memoria principal y viceversa. También incluye mecanismos de índice para aumentar la eficiencia del procesamiento de las consultas.

- Por último, el subsistema de transacciones (Transaction Subsystem) brinda todas las funciones para el manejo de transacciones, incluyendo el control de concurrencia y la recuperación ante caídas del sistema.



La segunda versión ORION es multiusuario y se caracteriza por una arquitectura distribuida. En una arquitectura distribuida se identifican dos tipos de bases de datos: una base de datos pública, la cual es compartida por varios usuarios y varias bases de datos privadas. La base de datos pública puede estar distribuida en varios nodos de la red local. Sin embargo, las aplicaciones no ven esta distribución; esta se lleva a cabo sólo por razones de eficiencia, subdivisiones de la carga y capacidad de los nodos. Una base de datos privada pertenece a un usuario y sólo puede accederse a ella por este usuario. El usuario puede obtener los datos de la base de datos pública y almacenarlos en su propia base de datos, y viceversa -utilizando operaciones de verificación-de-entrada y verificación-de-salida-. Consecuentemente, las aplicaciones distinguen entre las bases de datos privadas y las públicas.

4.6 PICQUERY.

Picquery ha sido diseñado de forma similar al Query By example de IBM, como un lenguaje altamente conversacional y no procedural para un sistema de bases de datos gráfica, diseñado y desarrollado por UCLA. Picquery y el lenguaje de consultas Query by Example pudieran formar un conjunto de lenguajes a través de los cuales un usuario pudiera manejar una base de datos gráfica. Esta interfaz forma parte de una arquitectura dirigida al manejo de bases de datos gráficas y no gráficas.

Muchos de los sistemas de manipulación de datos que han sido implementados para manejar información gráfica, tales como imágenes digitalizadas, dibujos, etc., fueron desarrollados para áreas de aplicación muy específicas, como por ejemplo, aplicaciones geográficas, exploraciones militares y aplicaciones en medicina. Se han desarrollado muy pocos sistemas de propósito general. Consecuentemente los lenguajes de acceso o consulta a este tipo de sistemas son también muy específicos

Se han identificado y propuesto un conjunto de operaciones para la manipulación de datos que están soportados sobre un sistema de datos gráfica y se ha desarrollado Picquery como un lenguaje de alto nivel para implementar esas operaciones.

Desde el punto de vista de arquitectura, este esfuerzo es parte de un proyecto de la UCLA, cuyo interés es el desarrollo de lenguajes y manipuladores para bases de datos gráficas y no gráficas, al igual que para bases de datos convencionales.

EL MODELO DE DATOS Y SU MANEJO DE DATOS EN PICTURE DATABASE MANAGEMENT SYSTEM (PICDMS).

PICDMS usa el esquema de representación en forma de rejilla para los datos de tipo gráfico. Posee la única estructura de datos de imágenes, lógica empilada dinámicamente, la cual consiste en almacenar en una misma rejilla todos los datos de un gráfico o imagen y aunque pareciera que estos valores se almacenan de forma convencional, la diferencia radica en que se hace en forma dinámica, no se prevé un tamaño para la base de datos y cada "récord" tiene el espacio que necesitan y aumentará éste si así lo requiriese la imagen. Esto permite flexibilidad y eficiencia en la adición y borrado de datos, que es un requerimiento esencial para los manejadores de bases de datos. Una nueva imagen es añadida como un atributo en un récord y no como un nuevo récord en la estructura convencionalmente estática ya conocida, ya que no soporta este esquema dinámico.

El acceso a los datos se hace a través de un algoritmo que muestra una ventana rectangular que contiene las celdas con la pila en la que están almacenados los datos de la imagen en el orden en que fueron capturadas. El lenguaje de manipulación de datos de PICDMS es un lenguaje procedural, con la siguiente sintaxis:

<Comando>::<Nombre de comando>
(<conjunto de variables>)
<conjunto de campos>,

FOR <condición>,

El nombre del comando especifica la operación que se desea realizar. El conjunto de variables define las variables que se usarán en la ejecución de la operación. El conjunto de campos son aquellos sobre los que se ejecutará la operación y la condición tiene el objetivo de filtrar aquellos valores de campos que se deseen. Las operaciones principales y sus correspondientes comandos son: COMPUTE, LIST, DISTANCE, ADD, REPLACE, DELETE and PRINT. Con estos comandos se pueden ejecutar casi todas las operaciones de manipulación de los datos de una base de datos gráfica.

Las operaciones fundamentales de los lenguajes de consulta convencionales no son manejadas por este lenguaje y las que se pueden realizar ocupan mucho tiempo de procesamiento, dado por las características de los datos que contiene la base de datos.

Picquery tiene dos formas de funcionar:

1 Como lenguaje no procedural puro, que es bueno emplear para el acceso en línea. Sin embargo, cuando el número de operaciones es muy grande se vuelve muy engorroso,

2. Formato tabular de consultas, que es una forma muy fácil de escribir una solicitud de información a la base de datos, pues el lenguaje muestra un esquema preestablecido de consulta para cada operación y el usuario sólo debe llenar en la tabla el valor de los datos que se piden.

Muchas áreas de tratados y revistas especializadas en sistemas de administración de datos se examinaron para proporcionar el común denominador del acceso ilustrado de datos y la manipulación de las operaciones requeridas.

Tres grandes áreas de aplicación ilustran las operaciones comunes que podrían estar disponibles en un PDBMS generalizado: Geográfica industrial y médica.

Un PDBMS para sistemas de información gráfica ilustrada necesita proporcionar las siguientes operaciones en el análisis de la imagen de datos: La capacidad de observar imágenes con diferentes niveles de detalle (zooming), rotación de imágenes por diferentes ángulos, tabulación cruzada de datos, identificación de los objetos más cercanos, operaciones de estadísticas, como medidas de promedios, operaciones geométricas o espaciales, enclaustrado y clasificación de puntos, detección de umbrales.

El PDBMS industrial en contraste con el geográfico tiende a trabajar con volúmenes mas pequeños de datos ilustrados y estos sistemas pueden efectuar cálculos complejos sobre las bases de datos de imagen. Operaciones tales como rotación de imágenes, acercamiento,

detección de filos, ajuste de plantillas, medición de texturas, cálculos geométricos y medidas estadísticas, son relevantes para estos sistemas.

Los PDBMS médicos son utilizados para el almacenamiento de imágenes de rayos X, también trabajan con grandes cantidades de datos. Las operaciones importantes para estos sistemas incluyen el mejoramiento de rotación de imágenes, extracción de contornos y segmentación para diagnóstico automático y semiautomático.

En general el PDBMS debe proporcionar muchas de estas capacidades de manejo de datos (compartidas a través de varias áreas de tratado) como sea posible. Las diferentes capacidades de manejo de datos que deberán ser proporcionadas por un PDBMS pueden ser clasificadas en seis categorías.

Principales operaciones del manejador de bases de datos gráfica.

1. Operaciones de manipulación de imágenes.
   Estas operaciones permiten ofrecer perspectivas diferentes de una misma imagen.

- Trasladar o panear (shifting operations).
- Rotar (rotation).
- Zoom (Zooming operations).
- Superponer (Superimposing operations).
- Transformación de colores (Color transformations operations).
- Proyección (projections operations)

2. Operaciones de reconocimiento de patrones.
   Estas operaciones permiten reconocer y dibujar patrones establecidos o buscar aquellas imágenes que en la base de datos gráfico coinciden con dichos patrones.

- Detección de filos; para detectar filos mediante cambios en la intensidad de la luz a través de la figura.
- Umbral; para construir una imagen binaria que sea blanca en las regiones con intensidad de luz menor que un límite de umbral y negra en el resto.
- Dibujado de contornos; para dibujar líneas de contorno en la figura asociada con el mismo valor de atributo.

- Recuperación de similaridad (ajuste de planillas); para identificar o recuperar objetos dibujados que sean similares a otros de otra figura usando una cierta medida de similaridad o bien que cumplan con ciertos patrones de plantilla. La recuperación por similaridad puede ser hecha sobre la base de tamaño, perfil, textura, etc.

- Establecimiento de la frontera o perímetro de una región.

- Medición de texturas para medir o cuantificar la textura de una imagen.

- Agrupamiento (clasificación de puntos); para agrupar objetos o puntos que están cercanos en una figura.

- Segmentación; para dividir una figura sobre la base de un mismo criterio.

- Interpolación; para interpolar valores de puntos dispersos de una función en particular.

- Vecino más cercano que identifique y recupere el objeto más cercano (de un tipo particular).

3. Operaciones geométricas o espaciales.

- Determinación de distancias punto a punto, punto a línea, línea a línea, línea a región, región a región.

- Determinación de longitudes.
- Determinación de áreas.
- Búsqueda de líneas iguales en diferentes imágenes.
- Búsqueda de regiones iguales en diferentes imágenes.
- Determinación de intersecciones.
- Unión de dos regiones.
- Determinación de diferencias.

4 Operaciones funcionales.

- Cálculo de máximos, mínimos, valores totales, contadores.

- Funciones estadísticas (promedios, desviaciones estándar, histogramas, intervalos, referencias cruzadas, etc.)

5. Operaciones de entrada y salida.

- Listar o imprimir la información de una imagen.
- Grabar una imagen o partes de ella en algún dispositivo de almacenamiento.
- Colorear imágenes.
- Cambiar o actualizar los valores de una imagen.
- Almacenar nuevos datos relacionados con alguna imagen.

EL LENGUAJE DE CONSULTAS PICQUERY.

El lenguaje de consultas Picquery puede operar en base de datos, gráficas completas o en parte de ellas. Una imagen es identificada de manera particular y un objeto de ella es por ejemplo, el nombre, los valores que describen una línea, una región o un punto. Al hacer una consulta se hace referencia a uno de estos objetos o a la imagen completa y si al realizar la consulta Picquery no encuentra estos nombres ofrece como respuesta la base de datos completa. Asimismo hay operaciones que sólo pueden ejecutarse sobre la imagen completa y otras sólo sobre partes de ella.

Estas funciones estarán disponibles para aquellos usuarios que desean implementar nuevas funciones y construir consultas con ellas.

Se ha presentado la motivación y justificación para el desarrollo de un lenguaje altamente interactivo para el incremento y desafío de la administración generalizada de datos gráficos. PICQUERY y un lenguaje relacional tipo QBE formarían el lenguaje mediante el cual el usuario puede accesar bases de datos relacionales y al mismo tiempo administrar base de datos por PICDMS (u otros PDBMSs fuertes). También se ha presentado un conjunto racionalmente comprensible de accesos a datos gráficos y operaciones de manejo requeridas por un PDBMS. El lenguaje PICQUERY diseñado provee el soporte para la gran mayoría de estas operaciones utilizando pocos comandos de entrada. Aparentemente los PDBMSs convencionales (basados en lenguajes relacionales, CODACYL, DBMS jerárquico) no proporciona el conjunto adecuado de operaciones fundamentales para soportar adecuadamente la mayoría de las operaciones gráficas.

Es claro que la aproximación PICQUERY es un lenguaje finalmente abierto el cual puede ser extendido para acomodar funciones nuevas o adicionales que pueden ser de interés particular para áreas específicas. De hecho la arquitectura de los PICDMS, de los PICDMS DML orientados a procedimientos y de PICQUERY es tal que proveen una estructura

fundamental sobre la cual posteriores operaciones de lenguaje y necesidades de manejo de datos pueden ser construidas fácilmente.

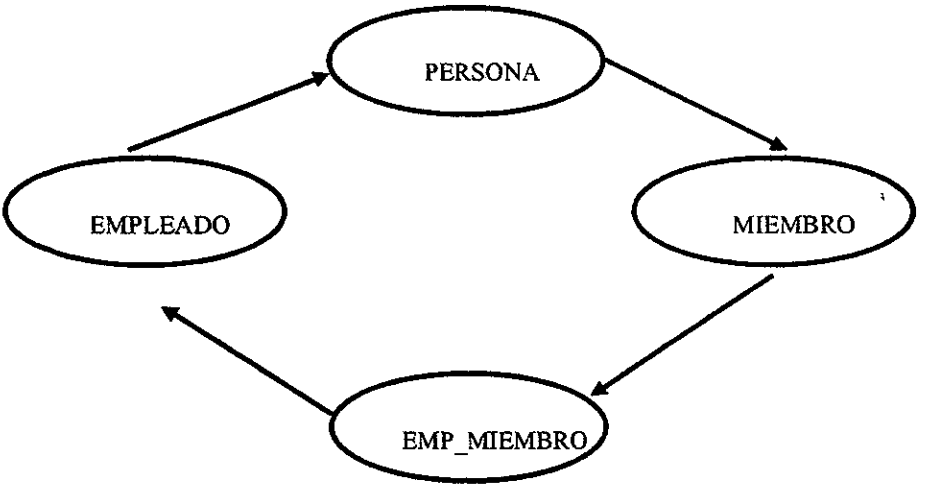
4.7 POSTGRES.

El modelo de datos de POSTGRES incorpora ideas de tipos de datos abstractos, datos de tipo procedimiento y herencia para extender el modelo de datos relacional. Estas ideas se incorporan para simular los conceptos de varios modelos semánticos de datos pero se concentran en modelar conceptos de Orientación a objetos. Ideas similares pueden usarse para incorporar objetos compuestos y objetos compartibles. La filosofía de este enfoque consiste en aprovechar la experiencia y madurez del modelo relacional agregándole elementos que le proporcionen tanto la flexibilidad necesaria para las nuevas y sofisticadas aplicaciones como la facilidad de modelado de la metodología Orientada a objetos que reduzca los costos de desarrollo.

Modelo de datos.

En POSTGRES la base de datos está compuesta de una colección de relaciones que tienen eneadas las cuales representan entidades del mundo real o relaciones. Una relación tiene atributos de tipos fijos que representan las propiedades de las entidades y relaciones y una llave primaria. Los tipos de atributos pueden ser atómicos (enteros, punto flotante o booleano) o estructurado (arreglo o procedimiento). La llave primaria es secuencia de atributos de una relación, los cuales, al tomarse en conjunto, identifican la eneada o registro de manera única.

La definición de una relación específica opcionalmente la llave primaria u otras relaciones de las cuales se heredan atributos. La herencia de datos se especifica en la cláusula **inherits**. Supongamos, por ejemplo, que las personas en la base de datos de un club deportivo son empleados o clientes, y que diferentes atributos definen cada categoría. La relación para cada categoría incluye los atributos de PERSONA y los atributos de la categoría específica. Estas relaciones pueden definirse duplicando los atributos de PERSONA en cada definición de una relación o heredándolos de la definición de PERSONA. La siguiente figura muestra la relación y una jerarquía de herencia que puede usarse para compartir la definición de los atributos elementales de una persona.



Los comandos para definir las relaciones de las subclases de PERSONA de la figura anterior son:

```
create  EMPLEADO  (Dept = char[25],
          Status = int2, Jefe = char[25],
            Plaza = char[25], Salario = money)

inherits (PERSONA)


create  MIEMBRO (Mnum = char[12],
            Status = int2, Votante = bool,
              Dept_favorito = char[20])


inherits (PERSONA)
create  EMP_MIEMBRO (EsActivo = bool)

inherits (MIEMBRO, EMPLEADO)
```

De esta manera, todo empleado y todo miembro es una persona con nombre y domicilio y hay algunas personas que son tanto miembros del club como empleados del mismo

Una relación hereda todos los atributos de su superclase a menos que un atributo particular sea redefinido. Por ejemplo, la relación EMPLEADO hereda de la relación PERSONA, los atributos Nombre, Fecha_nacimiento, Altura, Peso, Domicilio, Ciudad y Estado. La especificación de la llave también se hereda, así que, si hubiéramos distinguido a Nombre como la llave de PERSONA, ésta también lo sería de EMPLEADO.

Las relaciones pueden heredar atributos de más de una superclase. Por ejemplo, EMP_MIEMBRO hereda atributos de las clases MIEMBRO y EMPLEADO. Un conflicto de herencia ocurre cuando el mismo nombre se usa en un atributo para dos superclases. Si los atributos heredados tienen el mismo nombre y tipo, un solo atributo de ese tipo es incluido en la relación que está siendo definida. De otra manera, la declaración es prohibida.

El lenguaje de consulta de POSTGRES es una versión generalizada de QUEL, llamada POSTQUEL. QUEL ha sido extendido en varias direcciones. Primero, POSTQUEL tiene una cláusula from para definir variables de tipo registro en lugar de un comando range. Segundo, cualquier expresión que tome un valor de relación puede aparecer en cualquier parte donde el nombre de una relación puede aparecer. Tercero, un comando para la cerradura transitiva y un comando execute han sido añadidos al lenguaje. Finalmente, POSTGRES mantiene una cronología histórica de los datos para poder hacer consultas a versiones anteriores de la base de datos.

POSTQUEL también provee un conjunto de operadores para comparaciones y un operador que es constructor de relaciones, lo cual facilita la descripción de ciertas consultas que en otros lenguajes serían más complicadas.

Tipos de datos.

POSTGRES suministra una colección de tipos de datos atómicos y de tipos de datos estructurados. Los tipos atómicos predefinidos incluyen int2, int4, float4 float8, bool, char y date. Las operaciones aritméticas usuales y las operaciones usuales de comparación se ofrecen para estos tipos de datos e incluso para cadenas de caracteres. Los usuarios pueden definir nuevos tipos de datos mediante la definición de tipos de datos abstractos (TDAs). En realidad todos los tipos de datos del sistema se definen como TDAs. Un TDA se define especificando el nombre del tipo, la longitud de su representación interna en bytes, los procedimientos para convertir a otros tipos y entre representaciones, y un valor por defecto. El comando:

```
define type int4 is (InternalLength = 4,
InputProc = CharToInt4,
OutputProc = Int4ToChar, Default = "0")
```

define al tipo int4 (predefinido en el sistema). Los procedimientos CharToInt4 y IntToChar son rutinas en "C" u otro lenguaje convencional. El TD es abstracto pues los detalles de su definición se encapsulan y protegen de sus clientes.

Los operadores de un TDA se definen especificando el número y tipo de los operandos, el tipo del resultado, la presencia y dirección de asociación del operador, y un procedimiento que implanta el operador.

Datos de Tipo Procedimiento.

Los constructores estructurados de tipos de POSTGRES pueden usarse para representar datos complejos. Es posible construir arreglos de longitud variable. El segundo tipo de constructor es el tipo procedimiento el cual da la facilidad de almacenar valores de tipo procedimiento dentro de un atributo en un registro. Los valores de tipo procedimiento son representados como una secuencia de comandos de POSTGRES. El valor de un atributo de tipo procedimiento es una relación porque eso es lo que resulta del comando retrive. Más aún, el valor resultante puede consistir de registros de varias relaciones diferentes (es decir, de tipos diferentes) ya que un procedimiento con dos comandos retrive produce la unión de las tablas resultantes.

POSTGRES define una relación donde los registros son de varios tipos: una multirelación (las multirelaciones no son almacenadas por el sistema, sólo pueden analizarse por la interfaz con el usuario).

POSTQUEL es posible definir dos tipos de procedimientos· variable y parametrizado. El tipo de datos pocedimiento-variable permite que un procedimiento distinto sea almacenado en un atributo de cada registro en una relación. Por otro lado, un procedimiento parametrizado puede ser almacenado en un atributo y recibir parámetros cuando es invocado.

4.8 SIM.

Sim es una demostración de bases de datos de técnica disponible basado en un modelo de datos semánticos similar al de Hammer y McLeod's SDM. Sim tiene dos efectivos primarios de modelamiento. El primero es estrechar la puerta entre la percepción del usuario del mundo real de datos y el modelo conceptual impuesto por la Base de Datos debido al nombramiento de tres suposiciones o limitaciones. El segundo objetivo es permitir tanto como sea posible que los datos semánticos puedan ser definidos en un esquema y hacer que el

sistema de bases de datos sea responsable de mantener una integridad. Sim provee un conjunto rico de construcciones para la definición de los temas, incluyendo aquellos para la generalización especificada, jerarquías en modelamiento por gráficas directas acíclica, relaciones interobjeto y reestructuras de integridad. También trabaja novedosamente y fácil de usar un modelo de manipulación de datos en lenguaje inglés para recuperación y actualización de operaciones. Este trabajo describe la clave de las características de los modelos de Sim, la arquitectura del sistema y sus consideraciones de implementación.

El Sim (Manejador de Información semántica) fue iniciado en 1982 en Unisys con el objetivo de producir la próxima generación de sistemas administradores de bases de datos (DBMS). El modelo semántico fue escogido porque es rico, expresivo conceptualmente natural y provee una trayectoria de crecimiento para los usuarios del DM SII. DMSII, es un manejador de bases de datos basado en las estructuras de trabajo de los modelos de datos que corren en las máquinas Unisys serie A. Han sido instalados en bases para usuarios y han sido usados para implementar grandes aplicaciones complejas. Sim inicialmente ha sido construido sobre la base del DMSII y descansa en DMAII para transacción, y manejo del cursor Y/O. Sin embargo la arquitectura del sistema es virtualmente diseñada de manera que cualquier fuente de datos, incluyendo otros sistemas de bases de datos, puede ser sustituida en el lugar del DMSII. Sim forma la base para el Infoexec, el cual provee un arreglo de bases de datos y herramientas de aplicación, incluyendo ADDs.

ESQUEMAS EN SIM.

La mayoría del trabajo en Modelo Semántico de datos ha sido concentrado en su utilidad como herramientas de diseño de bases de datos. Mientras que algunos prototipos de bases de datos pueden implementar un conjunto seleccionado de propiedades de modelos semánticos existentes, hacia el mejor de nuestros conocimientos, SIM es uno de los más grandes, completos en modelos comercialmente implantados.

ENTIDADES.

Las entidades son objetos abstractos que pueden representar una idea o una cosa en un mundo real. Ellos juegan un papel en el SIM a las grabaciones en el DBMS convencional o tuplas en el sistema relacional. Sin embargo, las entidades son construidas lógicamente y no implican particularmente implementaciones físicas como grabaciones y tuplas que así lo hace. Las entidades y objetos en sistemas orientados a objetos son similares en su estructura pero diferentes en que los objetos pueden poseer descripciones algorítmicas llamadas métodos que las entidades de SIM no tienen.

CLASES.

La unidad primaria de una encapsulación de datos en SIM es una clase, la cuál representa una significante colección de entidades. Una clase es también una base de clases y subclases. Una base de clases es definida independientemente de todas las otras clases en la base de datos, mientras que la subclase es definida en una o más clases, llamada superclases.

Cada base de clases tiene un especial sistema que se ha mantenido con atributos llamados surrogate. Todas las eventuales clases derivadas de una base clase heredan los atributos de surrogate. El surrogate evalúa para cada entidad en una clase que debe ser única, y no puede ser cambiada una vez definida.

En SIM, los surrogates juegan un papel central en la implementación ó generalización de jerarquías y de relaciones de entidades.

En SIM, una distinción es hecha entre atributos de valores de datos (DVA) y atributos de valores de entidad (EVA). Un atributo de valor de datos describe una propiedad de cada entidad en una clase por asociación de las entidades con un valor ó un multiconjunto de valores de un dominio de valores. La definición de DVA en SIM y de atributo en el modelo E-R son similares. Los valores de DVA, pueden ser puestos en una pantalla ó impresora, nombre y día de nacimiento de la clase persona son ejemplos de DVAs.

Un atributo de valor entidad (EVA) describe una propiedad de cada entidad de una clase por relación de una entidad o entidades con otra clase o quizá con otra entidad de la misma clase. Un EVA representa una relación binaria entre la clase que le pertenece, el dominio y la clase de los puntos rango. En SIM usa las EVAs de relaciones de modelo entre entidades. Aquí, el valor de un EVA no puede ser puesto en un aparato de salida.

CONSIDERACIONES DE IMPLEMENTACION.

Objetivos.. .

En adición a sus funciones objetivos, SIM trabaja específicamente con objetivos de implementación.

Ambiente.

SIM ha sido inicialmente implementado sobre el sistema Unisys lines de estructuras series A. Todas las series A son compatibles con códigos de objeto y varían en sistemas de nivel de entrada hacia estructuras de gran escala Los principales datos y módulos de software de la base de datos de SIM deben residir sobre una serie A y los datos deben ser accesibles a través de estaciones de trabajo así como terminales conectadas hacia la computadora madre. La heterogeneidad y la distribución de los accesos de datos también debe ser direccionada.

Compatibilidad.

Un mayor objetivo de SIM es ser totalmente compatible e integrada con los sistemas administradores de datos (DMSII), una estructura DBMS en la cual es utilizada casi toda la serie A. Las actuales bases de datos DMSII deben ver el SIM como funcional y adicional en sus aplicaciones de datos existentes y códigos para reservarlos. Así una trayectoria de migración debe ser proporcionada para una base de datos DBMSII que pueda ser convertida en una base de datos SIM.

Desarrollo.

Los sistemas de serie A primeramente soportados con aplicación comercial que cumple un amplio rango de necesidades de procesamiento de datos. Al final de este rango los sistemas que requieren altas transacciones de velocidad (cien transacciones por segundo o más) y los sistemas que accesan grandes cantidades de datos (10 gigabytes de datos o más) SIM también proporciona desarrollos a nivel de producción para el rango completo de aplicación de sistemas encontrados en la serie A, incluyendo aquellos superiores.

Independencia de datos.

SIM también debe proveer el mas alto grado de independencia de datos posible de manera que los cambios físicos de las bases de datos pueden ser hechos sin impactar en las aplicaciones existentes. La lógica de los cambios de las bases de datos no deben afectar los programas de consulta aunque sean relevantes y cuando se necesiten, las consultas, la reorganización y reoptimización deben ser automáticas.

Accesos heterogéneos de datos.

Los objetivos iniciales de SIM son para proporcionar accesos simples de datos en los cuales residen las bases de datos de DMSII y simples archivos de datos. Sin embargo, su

arquitectura debe ser acomodada a otras fuentes y tipos de datos incluyendo estructuras de datos complejas, datos transitorios tales como interfaces de procesos y residencia de datos en hosts ajenos. La meta de este objetivo es asegurar que ambos el modelo y la implementación de la arquitectura de SIM sean lo suficientemente flexibles para permitir una organización para usar modelos uniformes con los cuales pueden representar y accesar toda esta información

Arquitectura funcional.

La arquitectura funcional básica utilizada para SIM es una versión mejorada del ANSI/SPARC. SIM emplea las capas externas y conceptuales, pero la capa interna propuesta ha sido puesta en una capa de interface y en una fase física.

La arquitectura funcional de SIM es una versión mejorada del ANSI/SPARC en la cual el nivel más bajo, la capa interna ha sido puesta en una interface y en una fase física. Cada capa posee un esquema asociado a los componentes DBMS del Software.

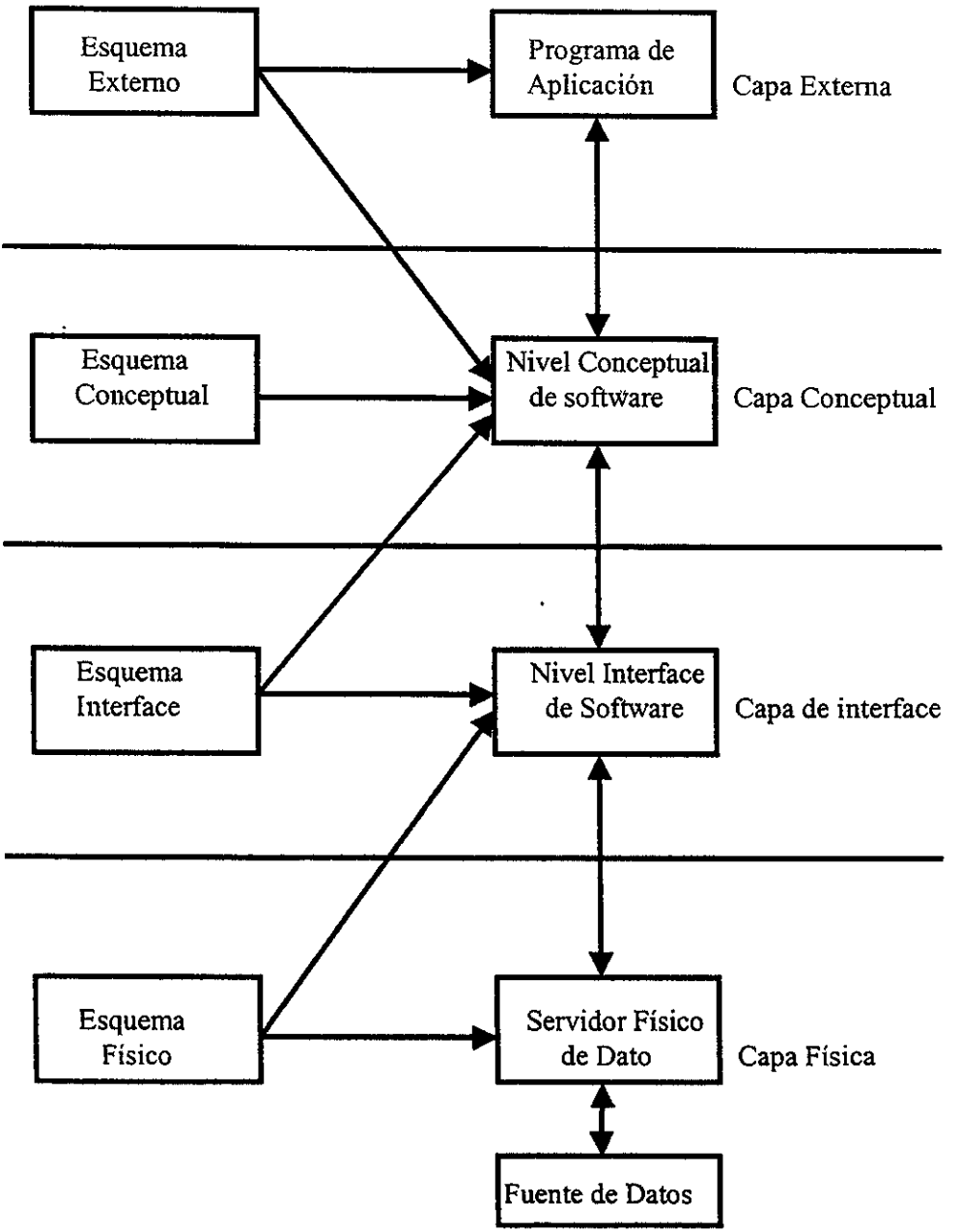
Cada capa funcional posee un esquema la cual define la base de datos de un nivel sucesivo mínimo Cada capa posee también cierto software DBMS el cual es involucrado en crear mantener o accesar la base de datos. Cabe mencionar que la arquitectura ANSI/SPARC define una información de mapeo la cual existe entre cada capa.

La ventaja primaria de esta arquitectura es que permite a un servidor de datos existente proporcionar tantas funcionalidades como esto puede ser para una fuente particular de datos. En muchos casos la capa física puede ser proporcionada por un sistema de software ya existente. Las capas externas y conceptuales de SIM son idénticas sin importar el tipo de datos para la fuente utilizada. Aunque la interface SIM debe ser añadida para cada tipo de fuente de datos, si necesita la implementación de funcionalidad no proporcionada en la capa física. Esta aproximación minimiza la cantidad de ajustes que deben ser hechos para cada tipo de fuente de datos que nosotros queramos accesar a una base de datos SIM.

La arquitectura básica de SIM esta compuesta de:

Capa externa, capa conceptual, capa de interfase y capa física, y su esquema se muestra en la siguiente figura.

ARQUITECTURA BASICA DE SIM.



Capa Externa.

El propósito de esta capa es proporcionar al usuario final una aplicación de acceso a la base de datos. Cada usuario o programa de aplicación que accese a SIM estará relativa a los esquemas externos. Con el SIM los esquemas externos predeclarados no son necesarios para propósitos de seguridad dado que los esquemas conceptuales ofrecen predicados declarativos orientados a mecanismos que causan seguridad propia para ser mejorada automáticamente.

Capa Conceptual.

A este nivel la base de datos total es definida vía un esquema conceptual. Un esquema conceptual para una base de datos SIM contiene ambas definiciones estructurales (por ejemplo generalización de jerarquías, sus atributos y funciones) y definiciones imperativas por ejemplo: seguridad y restricciones de integridad. Sin embargo, para proporcionar independencia de datos, el esquema conceptual es completamente libre de referencias para la distribución de registros u otros detalles físicos de almacenamiento.

Capa de Interface.

A este nivel, un esquema de interface es generado para cada base de datos del programa utilitario SIM. El esquema interface es definido en términos de un modelo más simple y de menor nivel que el modelo semántico utilizado para el esquema conceptual. El modelo consiste de componentes lógicos fundamentales. (LUC) en el cual cada uno de los cuatro tipos: registros, campos de registros, índices y relaciones estos tipos LUC son escogidos porque:

- Los componentes del esquema conceptual son fácilmente mapeados en el LUC de los objetos de estos tipos.
- Los objetos LUC poseen suficiente información física para permitir consultas efectivas y optimizadas.
- El procedimiento de consultas del software así como los niveles conceptuales solo necesitan generar operaciones sobre un conjunto limitado de componentes estructurales, por ejemplo: limitadas cuando se comparan con el número de componentes físicos que pueden ser usados.

Colectivamente, estos tipos LUC pueden describir y por lo tanto, los objetos LUC pueden ser mapeados para cualquier tipo de almacenamiento orientado hacia el registro.

Se describe el SIM como un sistema de base de datos basado en un modelo semántico de datos. SIM proporciona un punto de vista natural de datos moviéndolos a través de la simplicidad notacional por modelamiento con un conjunto completo mínimo de constructores.

Entidades, generalización, jerarquías, relaciones interobjeto del esquema definido y restricciones de integridad son las claves conceptuales del modelo. El DML de este sistema es diseñado para tomar ventaja y directamente soportar características. Las nociones del DML de perspectiva y calificación por EVAS son completamente naturales del esquema de definición de estas ventajas.

La experiencia con un gran número de análisis de bases de datos es testimonio de la potencia y utilidad de los conceptos mencionados anteriormente. El estado actual del diccionario de datos ADDS es por sí mismo una base de datos SIM. Este consiste de trece clases base, 209 subclases, 39 EVA – Pares inversos, 530 DVAS y en su nivel más profundo, una jerarquía que representa 5 niveles de generalización.

Actualmente se esta trabajando en varias extensiones del modelo. El trabajo progresa incluyendo el diseño de mecanismos de vista, atributos derivados, ordenamiento de sistemas mantenidos de clase y datos temporales de EVAS, eficientes algoritmos para varias categorías de restricciones de integridad y experimentos en cuantificación de falta de naturalidad y facilidad de uso del DDL y los conceptos DML.

4.9 VBASE.

VBase es un ambiente de desarrollo orientado a objetos que combina un lenguaje procedural de objetos y la persistencia de objetos en un sistema integrado. Fue elaborado por la compañía Ontologic alrededor de 1989 y está implementado para máquinas Sun (ambiente UNIX) y VAX/VMS. Actualmente la compañía se llama ONTOS y el sistema también se denomina Ontos, se encuentra en la versión 3.0, está escrito en C++, soporta una interfaz SQL orientada a objetos y está disponible para la mayoría de las plataformas UNIX.

Como aspectos del lenguaje Vbase este incluye: una fuerte tipificación, parametrización, capacidad de tener tipos miembros de objetos agregados, entre otros. Soporta las relaciones entre objetos de uno-uno, uno-muchos y muchos-muchos, el mecanismo inverso y métodos "triggers" (disparadores).

Vbase esta basado en el paradigma de tipos abstractos de datos. En Vbase el comportamiento de un objeto está dado por la combinación de propiedades y operaciones Las propiedades representan el comportamiento estático y las operaciones el comportamiento dinámico.

Arquitectura del Sistema.

El sistema consta de tres capas o niveles:

- Nivel de abstracción. Implementa el meta-modelo de objetos, proporcionando soporte para la herencia, el despacho de operaciones, la combinación de métodos, etc. Es el nivel donde más interactúan los clientes.

- Nivel de representación. Es donde se hacen las referencias semánticas y se traduce la notación simbólica del nivel de abstracción en objetos denotables.

- Nivel de almacenamiento. Es el responsable del "mapeo" del nivel de representación en almacenamiento físico.

Cada nivel de Vbase es implementado dentro de Vbase mismo. Cada nivel tiene una especificación y una implementación. Para soportar la persistencia de objetos el sistema permite:

- Manipular gran número de objetos y consecuentemente manipular un gran espacio de almacenamiento para dichos objetos.
- Compartir objetos de datos entre múltiples procesos/usuarios, lo que implica coordinación (control de concurrencia) para evitar la corrupción semántica de la base de datos de objetos.
- Estabilidad del software tal que el espacio del objeto es mantenido en un estado consistente en la fase del sistema o errores medios (transacciones).

Vbase ofrece la funcionalidad del método de ligadura dinámica basado en la jerarquía de tipos, como hacen todos los sistemas de objetos. También es totalmente polimórfico.

Componentes del Sistema.

Antes de la versión actual presentaba dos interfaces del lenguaje: TDL (para la definición de tipos del lenguaje) y COP (C Object Processor). TDL es usado para especificar tipos de datos abstractos, o sea, es usado para definir tipos de datos y especificar sus propiedades y operaciones asociadas. COP es usado para escribir el código para implementar las operaciones y para escribir programas de aplicación.

El sistema incluye también un conjunto de herramientas para ayudar al desarrollo como editor, un "object browser", un "debugger", una interfaz SQL con extensiones de objetos y un programa verificador que checa consistencia de imágenes compiladas contra una base de datos de objetos.

- TDL es un lenguaje propietario. Es estructurado con características en común con lenguajes como Pascal, Modula y Algol. Los tipos son las entidades más comunes definidas por TDL. Un tipo sirve como un nexo para el comportamiento de sus instancias. Este determina las propiedades y define las operaciones de las instancias

Esta versión del sistema permite sólo la especificación de un supertipo, dicho supertipo coloca la definición de tipo en la jerarquía de tipo. El comportamiento es heredado vía la jerarquía de tipo de la manera esperada.

- COP es un supraconjunto estricto del lenguaje C. Cualquier programa que compila con C standard compilará con COP. Cuando una operación es invocada esta es despachada de acuerdo al tipo de objeto de la invocación.

El sistema brinda las siguientes funciones de SGBD:

- Agrupamiento de un conjunto de objetos en el almacenamiento secundario y en la memoria principal.
- Definición del inverso atributo. Esto significa que automáticamente se modifica el inverso de un atributo cada vez que se modifica el atributo
- Personalizar el acceso a un atributo, reemplazando las operaciones estándar (set y get) definidas por el sistema. Por ejemplo, el usuario puede leer y escribir un mapa de bits (bitmap) utilizando un algritmo de compresión.

Por ser un manejador de bases de datos puro podemos mencionar que, este modelo incorpora nuevas características del modelo orientado a objetos como:

- Objetos compuestos (estructurados recursivamente desde otros objetos).
- Los valores de los atributos son apuntadores a referencia de objetos.
- Contempla la referencia compuesta, que captura la semántica parte de (is part of).
- Los objetos compuestos se almacenan (y recuperan) en un espacio continuo de memoria.
- Proporciona manejo de versiones de objetos.
- Cambios de esquema (cambio, supresión o adición de clases de objetos, de atributos, de métodos, de la jerarquía de clases) en forma de invariantes.

En la figura de la siguiente página se muestra un cuadro comparativo de los sistemas analizados en este capítulo.

CUADRO COMPARATIVO DE SISTEMAS MANEJADORES DE BASES DE DATOS ORIENTADAS A OBJETOS.

PRODUCTO	TIPO	LENGUAJES UTILIZADOS	PLATAFORMA
GemStone	Extensión a persistencia	OPAL, C, C++, PASCAL	UNIX
Iris	Orientado a Objetos Puro	C, OSQL	UNIX, HP-9000
O2	Extensión a persistencia	C++	SUN
ObjectStore	Extensión a persistencia	C, C++	SUN 3, SUN4, Sparcstation, UNIX
Orion	Orientado a objetos puro	CAD, CAM	SUN, UNIX
Picquery	Relacional con extensión a objetos		
Postgres	Relacional con extensión a objetos	C, Pascal	
Sim	Relacional con extensión a objetos		Unysis serie A
Vbase	Orientado a Objetos puro	C++	SUN, UNIX VAX/VMS

CONTINUACION.

PRODUCTO	DESARROLLADOR	PROVEEDOR	ARQUITECTURA
GemStone	Servio Logic Development Corporation	GemStone	CLIENTE/SERVIDOR
Iris	Hewlett-Packard		CLIENTE/SERVIDOR
O2	Consorcio Altair		CLIENTE/SERVIDOR
ObjectStore	Object Design	Object Design Meta Data	CLIENTE/SERVIDOR
Orion	Microelectronics and computer corporation		DISTRIBUIDA
Picquery	UCLA		CLIENTE/SERVIDOR
Postgres	Universidad de California		CLIENTE/SERVIDOR
Sim	Unysis		CLIENTE/SERVIDOR
Vbase	Ontologic		CLIENTE/SERVIDOR

5. BREVE ESTUDIO DE OBJECTSTORE.

ObjectStore fue diseñado considerando principalmente· 1) proveer una interfase unificada de programación para datos persistentes y 2) asegurar que la unificación de interfases permitan el acceso más rápido a objetos persistentes que son comúnmente datos transitorios. La idea principal de utilizar ObjectStore es que permite utilizar C y C++ para agregar persistencia a las bases de datos así como tener la facilidad de utilizar lenguajes de programación sencillos, sin que para el manejo de la base de datos de ObjectStore necesitemos aprender un nuevo lenguaje de programación. Por ello a continuación mostramos algunos conceptos que nos ayudarán a comprender mejor el manejo de ObjectStore y las facilidades de éste al codificarse en C++.

5.1 GENERALIDADES.

El hecho de proveer una interfase unificada de programación se encuentra mediante una integración de ObjectStore y C++, que son, el SMBDOO y el lenguaje de programación, que comparten los mismos modelos de datos. Los tipos de datos son los mismos para ambos, incluyendo los tipos base, tales como enteros, caracteres, y apuntadores, así como también tipos más complejos, tales como estructuras y clases. Los operadores definidos sobre los tipos de datos son también equivalentes. La persistencia y la manipulación de objetos son idénticas en uno y en otro.

Esta integración tiene las siguientes ventajas:

Reusabilidad - El mismo código puede operar sobre datos persistentes y transitorios, se puede ahorrar trabajo al no tener que escribir dos conjuntos de funciones. También, las instrucciones extras no tienen que ser agregadas al código que accesa datos persistentes, la compatibilidad con bibliotecas y subrutinas es posible. Ambos hechos permiten hacer la conversión de C existente y el C++ se codifica mucho más fácil.

Ninguna traducción - Usted se ahorra desde tener que escribir código extra para traducir entre la base de datos y los tipos de datos de aplicación.

Facilidad de programar - Porque los sistemas de tipo son idénticos y porque el lenguaje de programación se usa como el idioma de manipulación de datos del ODBMS.

Arquitectura Básica.

Para que los datos persistentes puedan sobrevivir más allá de la vida del proceso que los creó, de algún modo se almacenan sobre medios permanentes tales como disco duro. Hay muchas elecciones de cómo deben ser implementados los medios de almacenamiento de la información, debido a que todos ellos son vitales para el desempeño de la base de datos.

Unidades de almacenamiento.

ObjectStore permite apuntadores directos de un objeto a otro, sin considerar si los objetos están en la misma base de datos, o en bases de datos diferentes sobre el mismo o sobre un servidor diferente. Asimismo las bases de datos de ObjectStore son entidades de disco que pueden ser divididas en tres componentes diferentes. Uno de los componentes es llamado página, que es una unidad física que combina directamente el almacenamiento sobre el disco. Los otros dos componentes son llamados segmentos y grupos, que son las unidades lógicas independientes del esquema de disco.

Almacenamiento de objetos.

La manera en que los objetos individuales combinan segmentos, paginas, y grupos es bastante simple. Más de un objeto puede almacenarse sobre una página única, y un objeto único puede medir las páginas múltiples (hasta el límite físico del sistema de archivo). Si existe un conjunto de objetos ocupando menos de 64K, unos grupos pueden usarse físicamente para agrupar juntos los objetos. Para conjuntos más grandes, los segmentos pueden usarse en lugar de los grupos. Cuando un nuevo objeto se destina, ObjectStore permite especificar o un segmento o un grupo como una manera de poner un objeto en la vecindad de objetos conexos. Para recuperar objetos la situación es un poco diferente. Es posible recuperar un objeto en grupos, pero no es posible a recuperar directamente el grupo resultante.

Expansión de base de datos.

Durante la expansión de una base de datos, cuando la cantidad de almacenamiento físicamente destinada deber aumentarse, las tres unidades de almacenamiento se tratan de manera diferente. Los segmentos, que son variables en el tamaño, nunca se llenan. Se expanden para acomodar datos nuevos. Las páginas y los grupos, sin embargo, fijan su tamaño y se llenan. En el caso de las páginas, ObjectStore automática y transparentemente crea nuevas páginas para manejar el almacenamiento adicional de datos. Los grupos, sin embargo, se fijan después de la creación inicial y se llenan sin crear nuevos grupos.

Procesos de la base de datos.

ObjectStore es un SMBDOO cliente/servidor, sus funciones de base de datos se separan en procesos individuales que son capaces de correr sobre máquinas distintas dentro de una red. Además de la aplicación en sí misma, dos procesos auxiliares se requieren: el servidor de ObjectStore y el administrador de Cache (algunas aplicaciones requieren que un tercero, el administrador directivo).

1) Servidor de ObjectStore.

Un servidor de ObjectStore maneja el acceso a las bases de datos bajo su control., almacena y recupera páginas de datos persistentes a solicitud de sus clientes. Estas conexiones son con base en protocolos LAN tales como TCP/IP; cuando los clientes y los servidores son LAN, utilizan los mismos mecanismos de comunicación tales como memoria compartida, cuando ambos procesos radican sobre la misma máquina. El servidor es también responsable de la mayoría de las funciones tradicionales de gestión de base de datos, tales como cerrado, detección de encolamiento y recuperación de transacciones.

Las aplicaciones únicas pueden usar concurrentemente una o muchas bases de datos diferentes. Estas bases de datos pueden ser controladas por servidores diferentes así como también existir en diferentes sistemas de archivo sobre la misma máquina de servidor

2) Administrador de Cache de ObjectStore.

El administrador de cache de ObjectStore comienza automáticamente cuando la primera aplicación de base de datos se invoca. Corre sobre la misma máquina como la aplicación de cliente y consecutivamente maneja las aplicaciones adicionales encontradas sobre que misma máquina. El propósito del cache de cliente es administrar un área de tenencia de una página de base de datos donde los procesos del cliente han sido usados recientemente. Su meta es minimizar el número de veces que los datos se accesan y que frecuentemente se transfieren a través de la red por el servidor. Trabaja sobre la suposición de que esas páginas de base de datos recientemente accesadas en una transacción completa, probablemente se vuelvan a usar. El administrador de cache se llama asincrónicamente desde el servidor de ObjectStore para administrar el cerrado de páginas; el cache por sí mismo no maneja al cliente.

3) Archivo de bases de datos y administrador directivo

Otra manera de mejorar el desempeño es elegir entre planes alternativos de almacenamiento de base de datos. Automáticamente ObjectStore almacena sus objetos de la base de datos en archivos ordinarios. Con este enfoque, llamado archivo de base de datos, las bases de datos existen en el espacio del sistema ordinario y archivan la jerarquía, comenzando por el directorio raíz; y todas las operaciones de I/O de las bases de datos son manejadas por las rutinas estándar del subsistema de archivos.

Las bases de datos se almacenan en los sistemas de archivos de ObjectStore, que son directamente controlados por ObjectStore. Los directorios se usan para organizar las bases de datos en estructuras jerárquicas. En contraste al enfoque de archivo de base de datos, este espacio es lógico, significando que es independiente de la colocación física real de las bases de datos. Por ejemplo, dos bases de datos podrían nombrarse en el mismo directorio pero realmente estar radicadas en diferentes sistemas de archivos de ObjectStore.

Un administrador directivo puede administrar particiones en disco sobre un servidor o servidores múltiples. Cada administrador directivo, administra jerárquicamente un espacio compuesto de quizás, muchos sistemas de archivos de ObjectStore con muchas bases de datos individuales.

4) Arquitectura de la Memoria Virtual.

Para accesar los objetos, un apuntador en una dirección de memoria es idéntico al apuntador del objeto de la base de datos. El objeto de base de datos, podría estar también sobre el disco, pero tan pronto como el programa de corrida intenta accesarlo, se pagina en la verdadera memoria por la activación del sistema de administración de la memoria virtual. Si la base de datos pudiera participar de algún modo en estas operaciones, tendría una manera altamente perfeccionada de desempeñar mapeo virtual del objeto, porque las arquitecturas modernas de máquina y la operación de sistemas se sintonizan a tales operaciones.

Mapeo de la memoria virtual.

Inicialmente, los apuntadores persistentes no son mapeados; no hay entradas de mesa de página para correlacionar sus ubicaciones a la memoria verdadera, dirigir o paginar los espacios. Cuando una aplicación trata de referenciar un apuntador, una excepción se señaliza. En vez de ser manejado por el sistema activo (que no tiene idea qué hacer con la referencia) es interceptado por ObjectStore.

Los Esquemas

ObjectStore necesita ser capaz de identificar que los objetos existan sobre una página particular así como también determinar sus lindes exactos y el esquema de sus estructuras internas. La ayuda no esta disponible en C++ porque las clases no son corridas a tiempo y normalmente no hay ninguna ubicación de dónde puede ponerse esta información. La solución natural está en almacenar la información en C++ como en la base de datos real, bajo el control de ObjectStore, que refiere a esa información como esquemas.

El generador de esquemas procesa la lista completa resultante de clases para producir un archivo llamado base de datos de esquema de compilación. Este archivo contiene la información de esquema que ObjectStore necesita a fin de recuperar ejemplos individuales desde las diferentes de unidades de almacenamiento.

Interfases de programación.

Los compiladores que se puede usar para construir aplicaciones en ObjectStore caen en cuatro categorías:

1. Compiladores de C
2. Compiladores de C++ sin plantilla de clases
3. Compiladores de C++ con plantilla de clases
4. Compiladores de C++ con plantillas de clases y el lenguaje de manipulación de datos (DML) de ObjectStore.

Cada categoría da acceso a las capacidades del SMDOOO ObjectStore. La categoría que se utilice dependerá básicamente del tipo de aplicación a la cuál estará destinado el código, así como también el compilador que esté disponible.

SMBDOO de ObjectStore.

La arquitectura de ObjectStore se diseñó para proveer una interfase unificada de programación. Lo primero que se quiso obtener fue integrar el SMBDOO de ObjectStore con C++; después se quiso lograr la adecuada utilización de la memoria virtual, que también juega un papel vital en combinar los modelos de datos y la base de datos. El resultado es un SMBDOO que permite programar en C y C++ agregar objetos a las bases de datos y persistencia a sus aplicaciones sin tener que aprender un nuevo lenguaje.

5.2 RELACIONES.

ObjectStore permite la representación directa de relaciones entre objetos que usan apuntadores y colecciones de apuntadores. La ventaja de implementar las relaciones de esta manera, es que no son necesarias las extensiones al lenguaje de programación anfitrión; los apuntadores a los objetos de base de datos son exactamente como los apuntadores de datos de C++ La desventaja es que los apuntadores se emplean mal. Debe tenerse cuidado para asegurar que la honradez de la base de datos es mantenida por todas las manipulaciones de apuntador.

Definición de relaciones.

Existen cuatro tipos principales de relaciones y son:

1. uno a uno
2. uno a muchos
3. muchos a uno
4. muchos a muchos

En ObjectStore se representan de la siguiente manera:

1. os_relationship_1_1 - para uno - a - una relaciones
2. os_relationship_1_m - para uno - a - muchas relaciones
3 os_relationship_m_1 - para muchas - a - una relaciones
4. os_relationship_m_m - para muchas - a - muchas relaciones

Como un ejemplo, aquí esta la definición de la clase cliente, con relación al vehículo.

```
Class cliente
 {
  private.
    ....
    os_relationship_m_1(Cliente,automóviles,Vehículo,propietario,os_Set<Vehiculo*>)
automóviles;

 };
```

En el ejemplo un cliente puede tener muchos automóviles. Por lo tanto, usando el ObjectStore os_relationship_m_1, nosotros definimos una relación uno - a - muchos entre las dos clases. Para que un cliente pueda tener muchos vehículos, la clase cliente debe tener un

conjunto de datos capaces de contener apuntadores múltiples. De aquí en adelante el m_1 es el símbolo utilizado para la relación muchos a uno.

La relación de miembros requiere de cinco argumentos

1. El nombre de la clase que define el miembro (Cliente)
2. El nombre del miembro (automóviles)
3. El nombre de la clase que define al miembro (Vehículo)
4. El nombre del miembro inverso (propietario)
5. El tipo del miembro de datos (os_Set<Vehiculo*>).

5.3 EJEMPLO DE CODIFICACION.

A continuación se muestra un pequeño código, mostrando las declaraciones principales que deben existir en cualquier base de datos de ObjectStore

1) Declaración de cabecera.

```
/****************************** customer.h ******************************/
#include <ostore/ostore.hh>          // ObjectStore
#include <ostore/relat.hh>           // - relationships
#include <ostore/coll.hh>            // - collections

class Vehicle;                       // forward declare
class WorkOrder;

class Customer                       // customer class
{                                    //------------------------------ data
 public:
    static    os_Set<Customer*>* extent;   // set of all customers
    os_backptr   bkptr;                    // index maintenance

 private:
    int       customerNumber;        // customer number
    char*     name;                  // customer name
    char*     address;               // customer address

 public:                             //  list of vehicles, 1:m relationship
                                     // maintained by ObjectStore
    os_relationship_m_1(Customer,cars,Vehicle,owner,os_Set<Vehicle*>) cars;
                                     // list of WorkOrders, 1:m relationship
    os_relationship_m_1(Customer,orders,WorkOrder,customer,os_Set<WorkOrder*>)
orders;
 public:                             //------------------------------ methods
```

```
    static      os_typespec* get_os_typespec(); // typespec for ObjectStore

    static void   access_extent(os_database* db); // get extent from database

    Customer(int no, char* n, char* a);          // constructor

    ~Customer();                                 // destructor

    void        addVehicle(Vehicle* car);      // add vehicle to set of vehicles
    void        removeVehicle(Vehicle* car);   // remove vehicle from set
    void        display();                     // print customer information
    char*       getName() { return name; }     // get the name of the customer
    static void   list_names(os_database* db); // list all customer names in db
};
```

2) Declaración de clases:

```
/***************************** vehicle.h ***********************************/

#include <ostore/ostore.hh>                    // ObjectStore
#include <ostore/relat.hh>                     // - relationships
#include <ostore/coll.hh>                      // - collections

class Customer;                                // forward declare

class Vehicle                                  // vehicle class
{                                              //------------------------------ data
 private:
    int         serialNumber;                  // serial number of vehicle
    char*       make;                          // make
    char*       model;                         // model
    int         year;                          // year

 public:                                       // owner of vehicles, m:1 relationship
                                               // maintained by ObjectStore
    os_relationship_1_m(Vehicle,owner,Customer,cars,Customer*) owner;

 public:                                       //------------------------------ methods

    static      os_typespec* get_os_typespec(); // typespec for ObjectStore

    Vehicle(int s, char* mk, char* mdl, int y);  // constructor

    ~Vehicle();                                // destructor
```

```
    void display();                    // print vehicle information

    int         getSerialNumber() { return serialNumber; }    // get the serial number
    char*       getMake()       { return make; }          // get the make
};
```

3) Generación de objetos.

```
/************************** generate.cpp ********************************/

#include <iostream.h>                    // i/o

#include <ostore/ostore.hh>               // ObjectStore
#include <ostore/relat.hh>                // - relationships
#include <ostore/coll.hh>                 // - collections

#include "customer.h"                     // customer class
#include "vehicle.h"                      // vehicle class
#include "part.h"                         // part class
#include "service.h"                      // service class
#include "workord.h"                      // workorder class

/*-------------------------------------------------------------------------*/
extern os_database* db;                   // database

/*-------------------------------------------------------------------------*/
void generate()                           // generate initial data
{
    Part*         pP[30];
    QuPart*       pQ[30];
    Customer*     pC[20];
    Vehicle*      pV[30];
    ServiceItem*  pS[20];
    WorkOrder*    pW[10],

  cout << endl << "Generating sample data..." << endl;

  OS_BEGIN_TXN(t1,0,os_transaction::update)
  {
    // allocate some customers persistently

    Customer::access_extent(db);

    pC[ 1] = new(db,Customer::get_os_typespec()) Customer(101,"Senator, Dale"
,"Washington");
```

6. CONCLUSIONES.

El modelo orientado a objetos otorga una serie de capacidades y beneficios (más allá de nuevos tipos de datos) que el modelo relacional no contempla. Esto no significa que los sistemas manejadores de bases de datos relacionales desaparecerán en un futuro cercano, más bien, ambas tecnologías se complementarán al construir dos marcos de referencia diferentes desde los cuales se ve la estructura de la información.

Son diferentes las ventajas que proporciona que ambos modelos interactúen, por ejemplo:

- Introducción al concepto de tipo de dato abstracto y tipos definidos por el usuario (atómicos y no atómicos), que permite modelar aplicaciones más complejas (gráficas, imágenes, CAD, SIG, series de tiempo, financieras, etc.)

- Incorporación de un modelo semántico más claro que el modelo relacional para usuarios no familiarizados con las matemáticas. Esto implica que una consulta sea más fácil de formular en la propuesta OQL que en SQL tradicional.

- Respecto al modelo orientado a objetos podemos considerar que:

  - Tiene una semántica más clara en el modelado de los datos.
  - Cuenta con mayor facilidad en la definición de la operación de consulta (no es necesario analizar complejos productos cartesianos y sus correspondientes predicados lógicos basados en las llaves foráneas)
  - Tiene mejor desempeño al no ejecutar una operación join, sino un acceso directo al objeto al que apunta el tipo de dato definido en el dominio.

- Otro elemento del modelo de objetos que enriquece al relacional es la capacidad de modelar operaciones de los objetos. La teoría relacional contempla operaciones entre relaciones cuyos operadores son las relaciones completas; y contempla las operaciones propias de los dominios, pero como una caja negra. El orientado a objetos propone la definición de comportamiento de los elementos de las tablas, operaciones propias de las tablas; sin dejar de reconocer el cálculo relacional, sólo que particularmente es el cálculo de dominios.

- Herencia: la posibilidad de definir entidades que hereden atributos de otras entidades previamente programadas. La ventaja no está dada por el hecho de evitar código[1], ahorrando el trabajo de definir tablas con todos los atributos heredados. La ventaja real está dada en la conceptualización o modelado de elementos que

[1] Finalmente existen herramientas para el modelo relacional que contemplan este concepto en el modelo conceptual y genera un modelo físico.

heredan atributos, entre los cuales no sólo existen valores escalares, sino operaciones (comportamiento) que pudieran ser muy complejos y que al heredarlos facilitará el trabajo conceptual (no tanto el de implementación).

Podemos decir que ambos modelos se complementan y proporcionan una serie de ventajas entre sí. Por un lado continuaremos utilizando el álgebra y por el otro, se aprovechan las características de tipos de datos abstractos, herencia, comportamiento, y modelos semánticos más ricos.

Por otro lado, las bases de datos orientadas a objetos toman ventaja en las aplicaciones en las que las relaciones entre elementos de la base de datos llevan la información clave. Las bases de datos relacionales salen beneficiadas cuando los valores de los elementos de la base de datos llevan la información clave. Es decir, los modelos orientados a objetos capturan la estructura de datos; los modelos relacionales organizan los datos en sí. Si un registro puede ser comprendido aisladamente, entonces la base de datos relacional es adecuada. Si un registro tiene sentido solamente en el contexto de otros registros, entonces es más apropiada una base de datos orientada a objetos.

El alto nivel de productividad alcanzado en la investigación y el desarrollo durante los últimos diez años se ha debido principalmente al hecho de que los investigadores y diseñadores en SMBDOO han reutilizado y adaptado la tecnología y las experiencias acumuladas en el desarrollo de sistemas relacionales en los años 70. La fase exploratoria y de experimentación para la tecnología en SMBDOO está en vías de terminar y esta haciendo raíces para lanzarse a los ambientes productivos y llenar las necesidades que los sistemas tradicionales de bases de datos no han sido capaces de satisfacer. Además, la transición de la fase actual a la siguiente tomará sólo algunos años, requiriendo mayor investigación y consolidación. La investigación estará dirigida hacia el establecimiento de puentes entre los SMBD tradicionales y la tecnología de los SMBDOO, y los mejores resultados de los esfuerzos de investigación y desarrollo realizados hasta la fecha, no solo en el área de los SMBDOO sino también de las bases de datos extensibles, las bases de datos distribuidas y las interfaces de usuario se consolidarán para construir la siguiente generación de SMBDOO para ambientes de desarrollo productivos.

Con la nueva metodología ya no es necesario para el diseñador de sistemas convertir el dominio del problema en estructuras de datos y control predefinidas, presentes en el lenguaje de implementación. En vez de ello, el diseñador puede crear sus propios tipos de datos abstractos y abstracciones funcionales y transformar el dominio del mundo real en estas abstracciones creadas por el programador. Esta transformación, incidentalmente, puede ser mucho más natural debido al virtualmente ilimitado rango de tipos abstractos que pueden ser inventados por el diseñador. Además, el diseño de software se separa de los detalles de representación de los objetos de datos usados en el sistema. Estos detalles de representación

pueden cambiar muchas veces, sin que se produzcan efectos inducidos en el sistema de software global.

La naturaleza única del diseño orientado al objeto está ligada a su habilidad para construir, basándose en tres conceptos importantes del diseño de software: abstracción, ocultamiento de la información y la modularidad. Todos los métodos de diseño buscan la creación de software que exhiba estas características fundamentales, pero solo el diseño orientado a objetos da un mecanismo que facilita al diseñador adquirir los tres sin complejidad o compromiso.

Finalmente, el diseño orientado a objetos ha evolucionado como resultado de una nueva clase de bases de datos orientadas a objetos, tales como GemStone, Vbase, Iris, ObjectStore, etc.

Las dos tendencias básicas en Bases de datos orientadas a objetos son:

- Agregar capacidades a los paquetes existentes (extensión del modelo relacional)
- Incorporar persistencia a los lenguajes orientados a objetos (como C++ o Pascal).

Aparentemente las dos tendencias evolucionarán favorablemente y coexistirán, inclusive puede pensarse que tendrán capacidad de interconectarse para importar y exportar datos.

GLOSARIO DE TERMINOS

Clase.- Es el conjunto de atributos, operaciones y reglas que un conjunto de objetos comparten.

Encapsulamiento.- Es la combinación de datos y de procedimientos en una declaración de objetos.

Excepciones.- Son condiciones no usuales en un programa. En general, son anomalías de programas en tiempo de ejecución tales como división por cero, desbordamiento, o de rango de arrays y agotamiento de almacenamiento de memoria libre.

Herencia.- Es una relación entre clases que permite declarar una clase (subclase) como extensión o especialización de otra clase (superclase).

Objeto.- Es cualquier ente identificable, que tiene un estado interno y un conjunto de operaciones que pueden ser aplicadas para consultar, o modificar su estado, estructurado en variables de instancia (campos), que pueden hacer referencia a valores primitivos o a otros objetos compuestos.

Persistencia.- Es la habilidad que tiene un usuario de una aplicación de que su información exista más allá de la ejecución de dicha aplicación o proceso, de tal forma que esa información pueda ser utilizada por otro programa, ejecutado posteriormente.

Polimorfismo.- Permite la manipulación de objetos de clases distintas como si fueran de la misma clase, con lo cual es posible definir interfaces uniformes para diferentes tipos de objetos.

Bibliografía

REFERENCIAS BIBLIOGRAFICAS.

Sistemas de Bases de Datos Orientadas a Objetos. Elisa Bertino y Lorenzo Martino. Addison-Wesley Iberoamericana, S.A., E.U.A, 1995. 278 páginas

Object Oriented Design with Applications. Grady Booch. The Benjamin/Cummings Publishing Company, Inc.

Paradigma Orientado a objetos. Dr. Alfredo Weitzenfeld, Marisol González Lozano. ITAM, Marzo de 1994.

Database and Knowledge - base systems (Volume I: Classical Database Systems). Jeffrey D Ullman, Computer Science Press.

Object Oriented Database Management (Applications in Engineering and Computer Science) Alfons Kemper, Guido Moerkotte, 1984, Pretince Hall.

Object Oriented Databases. Dimitris N. Chorafas , Heinrich Steinmann, Pretince Hall. 1993.

An Introduction to Database Systems. C.J. Date, Addison Wesley, 1995, 6th edition.

REFERENCIAS ELECTRONICAS.

Introduction to Gemstone (Chapter 2). GemStone Systems, Inc. 14 dec 95. http://www.gemstone.com/Products/TechOver/chapter2.htm

ONTOS OIS. http://www.ontos.com/ontos-ois.html

Versant - Development Solutions and Architecture. Versant, http.//www.versant.com/versant/products/datasheet

Object Query Languaje manual. O_2 Technology, 1995.

A standard for Object - Oriented DBMSs. R.G.G. Cattell, SIGMOD. Conference 1994, ACM, INC.

Object Oriented Extensions in SQL3. A status report. Krishna G Kulkarni, SIGMOD Conference 1994, ACM, INC.

OQL: The object Query Language. Davir Jordan, C++ Report, SIGS Publications 1995

ANSI X3H7 Object Model Features Matrix, ANSI X3H7 comitee, ANSI 1995-1996 capítulo OMG, ODMG y SQL3.

REFERENCIAS DE REVISTAS ESPECIALIZADAS

Soluciones Avanzadas
Bases de Datos
Año 1, Número 6, julio de 1993
Artículo: Bases de Datos Orientadas a Objetos: Realidades y Promesas
Autor Ofelia Cervantes de Poty

Soluciones Avanzadas
Nuevas Tecnologías de Bases de Datos
Año 3, Número 22, 15 de junio de 1995
Artículo: Sistemas Manejadores de Bases de Datos Orientadas a Objetos
Autor: Marcos Calderón Macías

Soluciones Avanzadas
Manufactura
Año 2, Número 12, agosto de 1994
Artículo: Extendiendo el modelo relacional para que sea Orientado a Objetos
Autor: Vladimir Estivill-Castro

Soluciones Avanzadas
Outsourcing
Año 5, Número 51, Noviembre de 1997
Artículo: El modelo de Objetos de ODMG-93
Autor: Amparo López Gaona

On the road to standards. Douglas Barry. Object Magazine. October 1995, SIGS publications

Object Database semantics. Mary E. S. Loomis. Journal of Object Oriented Programming, Vol.6, N° 4, July/August 1993, SIGS publications. Pp. 26-33

Querying object databases. Mary E. S. Loomis. Journal of Object Oriented Programming, Vol.7, N° 3, June 1994, SIGS publications. Pp. 56-70, 78.

Wha's missing from ODMG-93?. John F. Shiner, Object Journal, SIGs publications, February 1996.