UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN

Unidad Académica de los Ciclos Profesional y de Posgrado
del Colegio de Ciencias y Humanidades

HACIA UN ESTÁNDAR EN MODELOS DE DATOS

T E S I S
que para obtener el grado de
MAESTRO EN CIENCIAS DE LA COMPUTACIÓN
p r e s e n t a
PABLO MARTÍNEZ CASTRO

Asesor: **M.EN C. JAVIER GARCÍA GARCÍA**

Ciudad Universitaria, Enero de 1997

Hacia un Estándar en Modelos de Datos

A Dios

Por haberme permitido, una vez más, cumplir una más de mis metas; y otorgarme la vida y salud para seguir...

A mis padres

Por proporcionarme, nuevamente, el apoyo incondicional en esta etapa de mi formación

A la H. Universidad:

Forjadora de ilustres pensadores e investigadores, de quienes aspiro aprender un poco de su conocimiento; pero mucho de su sencillez

A los profesores:

Por haber aceptado dedicarme un poco de su tiempo, compartiendo comingo su acervo de conocimientos

A mis compañeros

Individuos que como yo pasaron momentos alegres, momentos de estudio; pero sobre todo momentos de convivencia y regocijo en el aprendizaje

ÍNDICE

INTRODUCCIÓN

Considerando la diversidad de métodos y modelos de datos que últimamente han surgido (o que surgieron desde hace tiempo, pero no se les había dado tanto seguimiento como en la actualidad), y tomando en cuenta que el *modelo relacional* de bases de datos propuesto a finales de los 60s por E.F. Codd, ha demostrado su eficiencia y utilidad por más de 25 años; y que al mismo tiempo, en algunas ocasiones no resulta la alternativa más viable para aplicaciones más específicas (CAD, CASE, SIG, etc.); ha surgido entre la comunidad investigadora y comercial una interesante polémica.

Por una parte, se establece que el modelo relacional tiene todavía mucho que dar, y que está aún lejos el día que desaparezca; aunque su filosofía inicial ya no se sigue del todo. Además, que si bien es cierto que los últimos desarrollos proporcionan una considerable cantidad de adiciones a los productos comerciales basados en este modelo; la base teórica en la que subyace es muy sólida.

Pero por otra, existe una corriente que considera que el modelo relacional y sus fundamentos ya no son suficientes para manejar *datos* de sistemas que van más allá de una aplicación financiera o informática, o en general aquellas donde el manejo de datos estructurados tabularmente y sus interrelaciones resulta prácticamente *natural*; además de que existe una gran brecha semántica entre el modelo de datos relacional y los datos en la realidad. Unos ejemplos son: Sistemas de Información Geográfica (SIG); las herramientas de ingeniería de software (CASE); los sistemas que manejan objetos espaciales, de diseño, arquitectura e ingeniería (CAD); las aplicaciones multimedia; etc.; que no pueden manipularse de forma suficientemente transparente utilizando solamente un manejador de bases de datos relacional.

Esta corriente afirma que los conceptos de la *programación orientada a objetos* (como la herencia y el comportamiento de objetos, entre otros), que comercialmente se ha puesto muy de moda en los últimos años, permitirán *realzar* las capacidades de los sistemas manejadores de datos; por lo que resulta básico plantear un modelo formal que tome estos conceptos y permita explotar la información al máximo.

La corriente partidaria del modelo de Codd no da a la programación orientada a objetos suficiente crédito, argumentando una serie de problemáticas manifestadas por el

modelo subyaciente a este paradigma (entre otras, la aparente falta de un modelo matemático formal como el que sirve de plataforma al modelo relacional).

Finalmente, una última corriente opina que es necesario aprovechar las facilidades más representativas y efectivas de ambos modelos; y definir el estándar a partir de éstas, considerando seguramente características interesantes de algunos otros modelos semánticos además de estos dos modelos de datos.

En la década de los 80s hubo constantes reuniones entre la comunidad investigadora para llegar a un consenso en definiciones y conceptos. En realidad no se llegó a nada concreto; por lo que en esta década, diversos grupos han surgido entre investigadores de la academia y de la iniciativa privada para llegar a la definición de un estándar que defina *de facto* las características que debe seguir todo sistema que se haga llamar *Manejador de Bases de Datos Orientadas a Objetos* (O.D.B.M.S.)

En este documento se presentan en forma sintetizada los conceptos fundamentales de cada modelo, así como una visión introspectiva de lo que desde hace algunos años y hasta la fecha se ha desarrollado. Al final se identificarán los modelos propuestos (su estado actual) por comités de reconocidas organizaciones de estándares internacionales que trabajan sobre el asunto: ANSI, OMG, y ODMG. Aunque se estima que estos estándares quedarán totalmente definidos hasta 1997 o después; se habla ya de nuevos lenguajes (SQL3 y OQL) que subyacen sobre los modelos presentados por estas organizaciones.

OBJETIVOS ESPECÍFICOS:

Después de presentar los conceptos fundamentales del modelo relacional y de la orientación a objetos, se intentará cumplir con los siguientes objetivos:

1.- Presentar y describir las características de los modelos (en su etapa actual) que se están desarrollando en instituciones internacionales de definición de estándares (capítulo V). Estos modelos, en conjunción, y una vez llegando a un acuerdo entre las organizaciones (lo cual se espera que suceda después de 1997), proporcionarán un *estándar* en la estrategia para modelar datos, que deberá ser *seguido* por los desarrolladores de nuevos productos (o nuevas versiones) de software para asegurar la

integración de la información y su mejor explotación.

2.- Explicar y clarificar los diferentes problemas que han surgido con respecto al modelado de datos con el paradigma orientado a objetos (capítulo III). Muchos de estos problemas son infundados (mitos, según el enfoque de Won Kim, uno de los primeros investigadores del área, y editor de diversos libros que recolectan artículos y puntos de vista de diferentes autores); y muchos de los cuales realmente existen (de algunos ya se ha investigado y propuesto la solución correspondiente).

3.- Presentar las características más importantes que los productos actualmente en el mercado proporcionan, clasificándolos de acuerdo a la corriente a la que pertenecen: sistemas que originalmente eran orientados a objetos y fueron enriquecidos con conceptos de persistencia, sistemas relacionales a los que se les agregó tipos de datos abstractos y herencia, o sistemas totalmente pensados partiendo de cero (capítulo IV).

Adicionalmente se presentan las características más representativas de los modelos relacional (capítulo I) y de objetos (capítulo II) con el fin de posteriormente discutir sus semejanzas y diferencias de una manera más objetiva (capítulo III).

I. MODELO RELACIONAL

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

Modelos 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 95]
- Una notación para describir dichos objetos [Ullman 88]
- 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][1].

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 complementan uno con otro.

Descripción

El *modelo relacional de base 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:

---

[1] Es importante destacar que en este trabajo no se abundará sobre estas estructuras algebraicas, dado que su naturaleza redunda en aspectos de implementación, y el nivel al que se discutirán las características de los modelos

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).

Conceptos

Conceptos matemáticos

Para poder definir adecuadamente al modelo relacional, es importante establecer primero algunos conceptos generales relativos al concepto matemático de relación, y por ende las definiciones a continuación tratarán de hacerse desde el enfoque abstracto del nivel conceptual que se describió en párrafos anteriores. Cabe hacer hincapié en que muchas de estas definiciones se harán utilizando la simbología formal de esta teoría, dada su naturaleza de conjuntos, a saber:

{ } : Llaves que encierran los elementos de un conjunto

| : Tal que

$\in$: pertenece ($x \in R$: x **pertenece** al conjunto R)

$\wedge\vee$: Conectivos lógicos de conjunción y disyunción (y, ó).

Dominio: unidad mínima de datos con significado, que contempla valores escalares

---

será más bien abstracto.

o atómicos[2] (no compuestos) [Date 95]. Conjunto de valores que comparten una serie de características específicas y poseen operaciones perfectamente definidas. Por ejemplo: conjunto de números enteros, conjunto de todas las fechas posibles, etc.

Relación: Conjunto de elementos compuestos formados por objetos pertenecientes a otros conjuntos entre los cuales existe alguna *correspondencia.*

Sea un conjunto un grupo o colección de elementos con características en común ; además, recordemos también que entre diferentes conjuntos pueden existir correspondencias o relaciones dadas por algún hecho o regla. Por tanto, la combinación de elementos de un conjunto que tengan correspondencia con elementos de otro conjunto se conoce como *producto cartesiano.*

El producto cartesiano de diferentes conjuntos constituye una serie de elementos (duplas en el producto de dos conjuntos, triplas en el de tres, cuádruplas en el de cuatro, y en general tuplas en cualquier producto cartesiano); de los cuales, no necesariamente todas las combinaciones satisfacen la regla de correspondencia, por lo que una relación entonces la constituye un subconjunto de estas tuplas o combinaciones (aquellas que cumplan la regla que la relación establece).

Así, formalmente una relación es un subconjunto del producto cartesiano de una serie de dominios (no necesariamente diferentes):

$$R \subseteq \{ (x_1, \ldots, x_n) \mid x_1 \in D_1 \wedge \ldots x_n \in D_n \}$$

$$R \subseteq D_1 \times \ldots \times D_n$$

Donde pueden existir dos D_i, $D_j \mid D_i = D_j \wedge i \neq j$.

Más especialmente, en el modelo relacional se entiende una relación como un conjunto de tuplas *además* de un encabezado constituido por un conjunto de características de esas tuplas, que indican los conjuntos de los cuales cada elemento de cada tupla toma su valor.

Atributo: Cada elemento del encabezado de la relación, que define a qué conjunto (dominio) pertenece cada elemento de las tuplas. No confundir con el valor de cada elemento en una tupla.

[2] En el capítulo del modelo orientado a objetos se observará cómo esta condición ya no es cierta.

Esquema de la relación: encabezado o conjunto de nombres de atributos que componen una relación [Date 95] [Ullman 88].

Grado: Es el número de atributos que definen una relación. Es equivalente al número de conjuntos o dominios sobre los cuales se lleva a cabo el producto cartesiano que dará como resultado la relación en cuestión.

Cardinalidad: Es el número de tuplas que contiene una relación. El número de elementos del subconjunto del producto cartesiano de dominios que efectivamente están relacionados.

Conceptos semánticos

A continuación se presentarán algunos conceptos relativos a características más especiales del modelado de datos, que van más allá del concepto matemático de relación, y que de alguna manera comienzan a mostrar las semejanzas con el modelo orientado a objetos:

Llaves: En una relación, es importante que no existan tuplas duplicadas. Para lo cual debe definirse el conjunto de atributos que identifican en forma única a cada tupla (*llaves primarias*) o a tuplas de relaciones asociadas (*llaves foráneas*).

Entidad: Abstracción de la realidad de un objeto tomando aquellas características (o atributos) más relevantes para el modelo a definir. Existen dos tipos de entidades: normales y débiles. Una entidad débil es la que depende de otra entidad para existir. Una entidad equivale a una tupla dentro de una relación.

Subtipo: Entidad cuyas características son similares a las de otra entidad superior; agregando características adicionales o modificando algunas de las que originalmente posee la entidad superior.

Asociación:[3] Atributos que constituyen un enlace entro dos o más entidades.

Valores nulos: valores que pueden tomar los atributos de una relación para algunas tuplas. Existen tres posibles valores para los atributos de estas tuplas (a esto se le llama "lógica tri-valuada"):

- Valor no disponible
- Valor no aplicable

[3] En este texto, en adelante, se traduce *relationship* como *asociación*, dado que traducirlo como *relación* podría originar una confusión entre los conceptos reales de *relation* y *relationship*.

- Cierto o falso (booleano)

Vistas: Un tipo especial de relación. Es la forma en que los elementos de los conjuntos son visualizados por el usuario, independientemente del esquema conceptual de la información. Una relación base es la que conceptualmente se define en términos de los elementos a modelar. Una vista es una relación derivada de la operación o combinación de relaciones base.

Definición extensional e intensional de conjuntos: Formas diferentes en que pueden definirse los elementos de un conjunto. La definición *extensional* es aquella en que se listan los elementos del conjunto uno a uno hasta el último. Ejemplo:

$$X = \{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 \}$$

La definición *intensional* es la que enuncia algún predicado lógico (condición) que cumplen los elementos que pertenecen al conjunto, en lugar de listar uno por uno. Ejemplo (utilizando el mismo conjunto X del párrafo anterior):

$$X = \{ x \mid 1 \leq x \leq 20 \; y \; x \in I \}$$

Independencia de los datos

La independencia de la base de datos con respecto a la implementación de la misma, es una de las principales aportaciones del modelo relacional. La imposición de este modelo a un conjunto de datos permite que la base de datos sea independiente de la máquina en la que está implementada y del lenguaje con el que se lleven a cabo las consultas o actualizaciones de la información. Así mismo, provee derivabilidad (vistas), eliminación de redundancia (al establecer una normalización[4] de los datos), y consistencia en las relaciones. Por otro lado, los modelos pre-relacionales[5] contienen una serie de confusiones y dependencias de plataforma e implementación [Codd 70].

Son tres los principales tipos de independencia que proporciona el modelo relacional [Codd 70, secc. 1.2]:

- Orden: la información debe almacenarse de diferentes maneras, no importando el orden en que esté dada. Es problema del manejador de base de datos conservar algún orden en

[4] El concepto de normalización se verá con más detalle en párrafos posteriores.
[5] Modelos de datos previos al relacional. Por ejemplo el de red y el jerárquico, de los cuales no se profundiza en este trabajo; sólo se remarcan sus problemas y cómo el relacional trata de resolverlos.

particular.

- Indexación: Un índice es un aspecto meramente de implementación para optimar las consultas y ofrecer diferentes ordenaciones de la información. No es parte del modelo en sí mismo.
- Independencia del trayecto de acceso: Para acceder a un dato en particular, debe estar en una relación explícita. No debe recorrerse un trayecto ramificado (arbóreo) para hacer referencia al dato[6]. No es práctico el desarrollo de programas de aplicación que prueben todas las estructuras arbóreas permitidas por el sistema para encontrar un dato, ya que puede caerse en una referencia a un dato inexistente.

Sin embargo, desde el enfoque de los tres niveles de abstracción de las bases de datos, existen dos tipos de independencia:

- Física: Posibilidad de modificar el esquema físico sin alterar el esquema conceptual ni requerir una re-definición de subesquemas. Esto implica que las modificaciones a la organización física de la base de datos puede afectar a la eficiencia de los programas de aplicación, pero nunca debe requerir que sean re-escritos.
- Lógica: La relación entre las vistas (nivel externo) y el nivel conceptual también debe proporcionar un cierto tipo de independencia, de tal manera que la modificación del modelo conceptual de la base de datos no modifique la visualización externa de ésta. Puede ser necesario agregar elementos de información que re-modelen semánticamente la base de datos; pero no deben alterarse las vistas.

La realidad es que dicha independencia no es del todo lograda en las implementaciones actuales del modelo relacional; ya que un manejador de base de datos (DBMS[7]) es capaz de reconocer los cambios en el esquema de la relación; pero en la aplicación final (la cual en la práctica siempre es un cliente del DBMS; o sea que solicita a éste los resultados de la manipulación de datos, y después le da algún tratamiento como visualizar en pantalla o enviar a la impresora bajo algún formato), regularmente un cambio en la base de datos implica cambios en el código de dicha aplicación.

Por otro lado, muchas de las características de la aplicación se almacenan como parte de la base de datos. Esto se hace a través de fragmentos de código correspondiente a

[6] Como sucede en el modelo jerárquico, que no se estudia en detalle aquí.

procesos almacenados (stored procedures) que permiten la ejecución más rápida de éstos. Entonces, en la práctica no existe del todo tal independencia entre datos y aplicación[8].

Debe hacerse la aclaración de que cada nivel de abstracción visualiza los datos de una forma diferente pero equivalente. A continuación se presenta un comparativo de los principales conceptos del modelo vistos desde sus tres niveles de abstracción:

Matemático	Intermedio	Físico
Relación	Tabla	Archivos, Bloques o segmentos, almacenamiento en disco, etc.
Atributo	Columna	Campo
Tupla	Renglón	Registro
Esquema de la relación	Encabezado de tabla	Estructura de tabla

Integridad

El modelo conceptual de una base de datos debe tener como característica el contar con una serie de reglas que le permitan mantener su integridad (que siempre se encuentre en un estado válido y consistente). Estas reglas o restricciones se clasifican en cinco tipos principales, que son cada uno integridad de...[9]:

- *Dominio:* En cada tupla de cada relación, los valores de los elementos en dicha tupla son tomados exclusivamente del dominio sobre el que están definidos.
- *Columna:* Además de la propia restricción del dominio, cada columna de una relación puede tener restricciones adicionales particulares. Por ejemplo: un atributo *edad* estará definido en el dominio de los enteros; pero tendrá como restricción adicional ser un valor entre 0 y 150.
- *Entidad:* No puede existir una tupla cuyos valores en los atributos de la llave primaria sean nulos. Toda tupla en una relación debe ser *perfectamente* identificable.[10]

---

[7] Data Base Manager System

[8] Al analizar el modelo orientado a objetos, se observará que lo que en los sistemas "relacionales" tuvo que implementarse como *stored procedures*, en los objetos es un modelado natural del comportamiento.

[9] El orden en que están enunciados los tipos de integridad es el orden dado por [Codd 69] y además es el orden en que se verifican estas reglas: de dominio a relación y de relación a base de datos.

[10] Este concepto se revisa en el capítulo del modelo orientado a objetos, como una propiedad de éstos, en particular los OIDs (Object Identifacators)

- *Referencial:* Es la regla de entidad pero para las llaves foráneas (atributos que asocian una relación con otra) de una tupla. No puede existir una tupla cuyos valores en una de estas llaves sean nulos. Es decir, por cada valor de la llave foránea de una relación debe existir una tupla asociada en la relación a la que se hace referencia.
- *Usuario:* Se refiere a diversos tipos de reglas que tienen que ver con información faltante, actualización en cascada, fecha y hora de actualización, seguridad y niveles de acceso, etc. En concreto se refiere a las condiciones y políticas que el DBA debe definir para el momento en que un usuario actualice o consulte la información.

Dependencias funcionales

Dado que el modelo relacional de base de datos se fundamenta en conjuntos matemáticos, es necesario analizar los componentes de la información y descomponerlos hasta que los valores en los dominios sean atómicos (como se estableció en párrafos anteriores) para que su operación matemática sea más consistente.

Para llevar a cabo este objetivo, se requiere ejecutar un proceso de *normalización* de la base de datos, en el cual es tomada en cuenta la *dependencia funcional* que existe entre los diferentes atributos de una relación.

La dependencia funcional de los atributos es la que existe entre un conjunto de uno o más atributos con respecto a otro conjunto igual, donde se da el caso de que por cada valor que toma el primer conjunto, el segundo siempre toma un valor asociado. Por ejemplo: en una relación de alumnos donde se tenga un atributo *matrícula* y uno *nombre del alumno* (además de otros atributos), cuando en algún documento o información escolar se muestre la matrícula, se tendrá siempre el nombre del alumno correspondiente a dicha matrícula. Es decir, el nombre del alumno depende funcionalmente de su matrícula.

Formalmente: Sea R una relación, y sean X y Y subconjuntos arbitrarios del conjunto de atributos de R. Se dice que Y depende funcionalmente de X (o X determina funcionalmente a Y), expresado como:

$$X \rightarrow Y$$

sí y sólo si cada valor de X en R está asociado precisamente a un valor de Y en R.

Existen cinco niveles de normalización, cada uno de los cuales considera una serie de condiciones explícitas. Se dice que una relación está en primera forma normal (1NF) si

todos sus atributos son atómicos (no compuestos). En esta tesis no se profundizará en el proceso de normalización (por no corresponder al objetivo central), pero es importante tomar en cuenta la condición de *valores atómicos*, para su comparación con el modelo orientado a objetos.

Operaciones del modelo

Entre las relaciones de una base de datos existen operaciones que son propias de conjuntos, dado que ésta es la base matemática del modelo. Dichas operaciones son:

- *Unión:* La incorporación de las tuplas de dos relaciones para generar una tercera, siempre que aquellas sean *compatibles en la unión* (Que tengan la misma cardinalidad y cada atributo pertenezca al mismo dominio en ambas relaciones).
- *Intersección* entre dos relaciones: extracción de tuplas que aparezcan tanto en una como en la otra relación, siempre que ambas sean compatibles en la unión.
- *Diferencia* de dos relaciones: Extracción de aquellas tuplas que aparezcan en una de las relaciones y no en la otra, siendo ambas compatibles en la unión.
- *Producto cartesiano* de dos relaciones: Obtención de una tercer relación en la que se tendrán todas las posibles tuplas resultado de la combinación de cada tupla de la primer relación con cada una de las tuplas de la segunda.
- *Selección:* Extracción de tuplas específicas que cumplan una *condición* dada (predicado lógico).
- *Proyección:* Extracción de un subconjunto de campos especificados.
- *Juntado natural (natural join)*[11]: Dadas dos relaciones (no necesariamente compatibles en la unión) que contengan un subconjunto de atributos en común, la selección de tuplas en el producto cartesiano de ambas relaciones, cuyos valores en dichos atributos sean los mismos, constituyen un join natural. Existen otros tipos de join, siempre que se operen sobre dos relaciones con atributos en común; pero con condiciones diferentes (mayor que, menor que, etc.) Así mismo, existe el *outter join* (juntado más a la derecha), inner join, etc. (que son operaciones con características más específicas fuera del objetivo de esta tesis).
- *División:* Operación entre dos relaciones, una de las cuales es el producto cartesiano de

la segunda relación y de la relación resultante de esta operación. Es decir, hay que obtener una relación que al multiplicarse por la segunda, se obtenga la primera.

Álgebra y cálculo relacional

El álgebra relacional está basada en las operaciones mencionadas en la sección anterior. El cálculo relacional está basado en las mismas operaciones, con la diferencia de que en éste se expresa lo que desea obtenerse, y con el álgebra relacional se expresan las operaciones a llevar a cabo para la obtención de un resultado[12].

- *Cálculo orientado a tuplas:* Son las operaciones entre relaciones que dan como resultado un conjunto de tuplas (una relación).
- *Cálculo orientado a dominios:* Son aquellas operaciones entre relaciones que dan como resultado una relación de un solo atributo (un dominio).

Aspectos lingüísticos

Un conjunto de datos del tipo relacional con un conjunto de operaciones como el álgebra y el cálculo relacionales, requiere de un lenguaje basado en el cálculo de predicados de primer orden (predicados lógicos con conectivos $\neg$, $\wedge$, $\vee$ y cuantificadores $\exists$, $\forall$); con la capacidad de poder ser sublenguaje de un lenguaje de programación.

Los productos comerciales basados en el modelo relacional utilizan el estándar SQL[13], que proporciona dos conjuntos de comandos o expresiones para operar las bases de datos relacionales: un lenguaje de definición de datos (DDL) y uno de manipulación de datos (DML). El primero permite crear y modificar relaciones y sus atributos, además de definir índices para ordenaciones lógicas particulares de la aplicación. El segundo facilita la manipulación de relaciones en base a las operaciones del álgebra relacional, y utiliza principalmente la cláusula **SELECT** con sus diferentes combinaciones y posibilidades. No obstante, el SQL en implementaciones comerciales ha constituido a la larga una deformación de los conceptos formales del modelo relacional; pues permite una serie de incongruencias y violaciones a las reglas del modelo [Date Darwen 94].

---

[11] Es incorrecto traducir join como unión, ya que la unión es otra operación totalmente diferente.

[12] Por ejemplo : el cálculo relacional dirá : *el conjunto de materias que estudia un alumno X.* El álgebra relacional expresará : *la selección de elementos de la relación cuyo atributo alumno sea igual a "X", tal que dicha relación sea el producto cartesiano del conjunto de* ***alumnos*** *y el de* ***materias***.

Características de un sistema RDBMS[14]

Además de las características del modelo relacional hasta el momento descritas, es importante hacer mención de algunas características concernientes a la seguridad de la base de datos, ya que son pieza fundamental de las cuales se hablará en el modelo orientado a objetos, con la intención de encontrar el equivalente; además de que Codd tomó en cuenta en versiones actualizadas de su modelo. En estas definiciones se hablará de tablas o relaciones, registros o tuplas, campos o columnas en forma equivalente.

Indexación **: La indexación (generación de tablas auxiliares con elementos llave o identificadores almacenados en alguna estructura de datos que permita su búsqueda y ordenación de forma más eficiente) no es una característica del modelo relacional, ya que uno de sus principios es que los conjuntos de tuplas que pertenezcan a una relación no tienen orden alguno semánticamente hablando. Sin embargo, los RDBMS regularmente contemplan la posibilidad de generación de índices (almacenamiento secundario) para agilizar el desempeño de las operaciones de consulta y relacionales en general de la base de datos. Esta indexación puede llevarse a cabo a través de estructuras de datos que utilicen ligas entre elementos (árboles, apuntadores, colas, etc.[15]) ; o bien almacenando en forma física las llaves según su orden lógico para aumentar la rapidez (almacenamiento en bloques de agrupación o *clusters*).**

Transacción**: Es el conjunto de operaciones sobre tablas que deben modificarse derivadas de una actualización global. Es decir, la actualización de los datos puede involucrar diferentes tablas y sobre éstas diferentes tipos de alteraciones (desde cambiar un valor en una columna hasta eliminar renglones). Sin embargo, para no perder la *integridad* de la base de datos (estado válido), la actualización debe llevarse a cabo en todas las tablas involucradas o en ninguna. En este sentido se dice que una transacción es atómica (indivisible).**

Recuperación**: Un DBMS debe también tener la capacidad de recuperarse en caso de falla o error. Las fallas pueden darse por daño en el equipo, por acceso no autorizado de usuarios o por definición errónea de los procedimientos. Un DBMS debe contemplar estrategias de recuperación o esquemas que permitan re-procesar transacciones (deshacer una**

---

[13] SQL : Structured Query Languaje. Lenguaje de consultas estructuradas

[14] RDBMS : Relational Data Base Manager System. Sistema administrador de bases de datos relacionales

[15] En este párrafo se habla de estructuras de datos en general, sin ahondar en los detalles de cómo están

transacción parcial o rehacerla hasta la última tabla actualizada). Los sistemas relacionales regularmente manejan el concepto de archivo de bitácora de actualización de transacciones (transaction log), el cuál es una relación adicional (*relación de sistema*) en la que el RDBMS actualiza una a una las operaciones de inserción, modificación o supresión de elementos del resto de las relaciones en la base de datos especificando qué relación se actualizó, en qué atributo y por qué valores.

Seguridad: Así mismo, toda base de datos está orientada a diferentes usuarios, según el nivel de acceso. Usuarios como el administrador de la base de datos, los que la actualizan, y los que la consultan para tomar decisiones. Para evitar que algún usuario haga uso indebido o actualice erróneamente la información de la base de datos, es necesario que el DBMS contenga otro conjunto de elementos donde sea especificado un catálogo de usuarios y de privilegios de acceso a la base de datos (consulta, inserción, modificación, supresión, administración, etc.). Esta información es almacenada en los RDBMS también a través de relaciones (que también forman parte de las *relaciones del sistema*) y operaciones *join* entre dichas relaciones.

Catálogo: El esquema de la base de datos (conjunto de esquemas de las diferentes relaciones de ésta), así como el conjunto de definiciones de llaves primarias, foráneas, reglas de integridad, etc. ; debe estar disponible para que el DBMS sea capaz de reconocer cualquier elemento y poder ejecutar las operaciones entre diversas relaciones. Esta información que define conceptual y estructuralmente a la base de datos en los sistemas relacionales, es almacenada en un conjunto de relaciones también de *sistema* que son conocidas como *catálogo*.

Como puede observarse, un RDBMS está completamente basado en relaciones. Tanto la seguridad como las transacciones, las operaciones de recuperación y la definición misma de una base de datos es registrada y almacenada en un sistema basado en este modelo utilizando primordialmente relaciones matemáticas (conjuntos de tuplas con un esquema dado, según ya se ha definido en párrafos anteriores).

Procedimientos almacenados (Stored procedures): Dado que el modelo relacional no contempla comportamiento de entidades (precisamente este es uno de sus fundamentos) ;

---

implementadas estas estructuras por no corresponder al alcance de esta tesis.

pero los implementadores de RDBMSs han considerado que no puede ser totalmente independiente la estructura de los datos con respecto al comportamiento de éstos (aspecto que el modelo orientado a objetos ataca de una manera más eficiente), la mayoría de las implementaciones soportan la programación de procedimientos que se almacenan también como parte intrínseca de la base de datos.

Disparadores de eventos (triggers): Así como los procedimientos almacenados mencionados en el párrafo anterior, los RDBMS actuales permiten la definición de procedimientos de validación que refuerzan las reglas de integridad de la base de datos (definida en secciones anteriores). Sin embargo, estos procedimientos no son directamente ejecutados por la aplicación que accesa a la base de datos, ni por el usuario final ; sino que son ejecutados cuando el RDBMS percibe que se ha tratado de violar la consistencia de la base de datos.

Concurrencia/esquema de bloqueos : Dado que una base de datos debe permitir el acceso concurrente (al mismo tiempo) de más de un usuario, es necesario un esquema de bloqueos para efectos de actualización. Dos usuarios no pueden actualizar el mismo objeto al mismo tiempo (a nivel de relación, tupla o hasta atributo). Un usuario podrá actualizar un dato una vez que tenga capacidad de bloquearlo en forma exclusiva. Esto lo podrá hacer cuando ningún otro usuario esté bloqueando el mismo dato. Sin embargo, para efectos de consulta, puede permitirse la visualización de los datos aunque estén bloqueados ; aunque no es muy recomendable porque esto implica que la información que un usuario consulta no será la misma una vez que el usuario que había bloqueado el dato lo libere.

El esquema de bloqueos puede ser optimista o pesimista. El primero de ellos se da cuando el bloqueo se logra inmediatamente. El segundo tipo de bloqueo se da cuando el DBMS intenta en varias ocasiones bloquear el dato, y después de un número finito de intentos, si no fue posible el bloqueo, lo indica al usuario sugiriéndole ejecutar la actualización más tarde.

Existen diferentes protocolos de bloqueo, a saber : lectura, escritura, monitoreo, snapshot (visualización instantánea de datos).

Modelos semánticos (ER)

El modelo relacional permitirá definir de manera adecuada la base de datos para cualquier

aplicación diseñada con una herramienta que lo soporte. Sin embargo, durante el análisis de la estructura de la información, es común encontrarse con la necesidad de presentarla de una manera menos abstracta para que el usuario final sea capaz de comprenderla.

Para tal objetivo existen diferentes modelos conocidos como *modelos semánticos*, que permiten al diseñador de la base de datos expresar el comportamiento y significado de la información. Uno de los más útiles y menos difíciles de comprender por un usuario final es el modelo Entidad-Asociación (Entity-Relationship). Este modelo mapea de forma casi natural al modelo relacional, por lo que es analizado en esta sección.

- *Entidad:* Abstracción de la realidad de un objeto tomando aquellas características (o atributos) más relevantes para el problema que se intenta resolver. Se mapea con las relaciones. Existen dos tipos de entidad: normal y débil.
  - ◊ Normal: Una entidad que puede existir sin depender de la existencia de otra.
  - ◊ Débil: Aquella que depende de otra para existir. Es decir, si no existe la otra entidad, la débil no debe existir.
  - ◊ Propiedades: Partícula o pieza de información que constituye un componente de la entidad.
- *Asociación (relationship):* Conjunto de atributos que permiten el vínculo entre dos o más entidades.

  Cardinalidad de asociaciones:
  - 1 a 1
  - 1 a muchos
  - muchos a muchos

El modelo ER utiliza diagramas cuya notación es la siguiente:

- ◊ Una entidad se representa como un rectángulo etiquetado con el nombre, y opcionalmente el conjunto de propiedades o atributos que la conforman (en algunas versiones éstas se diagraman en elipses etiquetadas unidas a la entidad).
- ◊ Una línea que parte de una entidad y llega a otra, anotando en el extremo de cada entidad su cardinalidad (0, 1 o *n*).
- ◊ Cuando existe un vínculo muchos a muchos, físicamente es necesaria otra relación. En el diagrama ER se utiliza un rombo en medio de la línea que une a

las dos entidades cuya cardinalidad es muchos a muchos.

A continuación se muestra un ejemplo de un diagrama E-R correspondiente a una pequeña base de datos de noticieros :

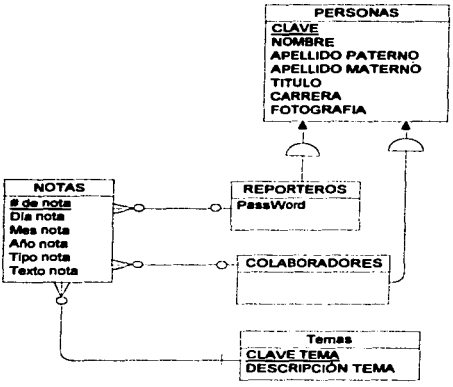

Figura 1 Diagrama entidad-asociación

En este ejemplo, existen dos entidades cuyos atributos se heredan de una misma entidad (denotado con un semicírculo). La entidad principal de la pequeña base de datos es la de NOTAS. Cada nota es redactada por un reportero, leída por un colaborador y corresponde a algún tema en particular.

El reportero y el colaborador heredan las características de la entidad PERSONA ; se puede observar que un reportero puede tener una o más notas (relación uno a muchos, denotada por las tres líneas en el extremo de la línea que enlaza REPORTEROS y NOTAS). Lo mismo sucede con colaborador.

El número de nota, la clave del tema, y las de reportero y de colaborador son las llaves primarias de las entidades (identificadores únicos), y por ello están subrayadas.

Bibliografía

[Codd70] A Relational Model of Data for Large Shared Data Banks, E.F. Codd, *Communications of the ACM, Vol. 13, No. 6, June 1970. pp. 377-387*

[Codd69] The Relational Model for Data Base Management, E.F. Codd, *1969*

[Date95] An introduction to Data Base Systems, C.J. Date, *6th edition. 1995 Addison Wesley*

[Ullman88] Principles of DataBase and Knowledge - Base Systems. Volume 1: Classical database systems, Jeffrey D. Ullman, *1988, Computer Science Press, Inc.*

[Levine92] Introducción a la computación, Guillermo Levine, *McGraw Hill 2ª edición, México 1992.*

[Enderton87] Una introducción matemática a la lógica, H.B. Enderton, *Universidad Nacional Autónoma de México, 1987*

[Abellanas 91] Matemática discreta, M.Abellanas, D. Lodares, *Macrobit ra-ma, 1991. España*

[Date Darwen 94] The third manifesto Hugh Darwen & C.J. Date *1994.*

II. MODELO ORIENTADO A OBJETOS

A continuación se revisarán las características generales comúnmente aceptadas del paradigma orientado a objetos; con el objetivo de analizar posteriormente el modelo de datos basado en este paradigma para hacer una comparación con el modelo relacional, observando semejanzas, diferencias y puntos en los que uno y otro se complementan. Esto nos permitirá revisar en el siguiente capítulo los puntos específicos en los que existe duda respecto a cómo el modelo orientado a objetos trata ciertos aspectos.

Conceptos Generales

Paradigma es a lenguajes de programación como *corriente* a filosofías de pensamiento.

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

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

Como se observa, 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[16]. Existe el análisis orientado a objetos (OOA), el diseño orientado a objetos (OOD), la programación orientada a objetos (OOP) y el modelo de datos orientado a objetos (OODB). 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.

[16] Lo anterior tomando en cuenta 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).

De ahí que existan muchos productos de bases de datos cuya interfaz de aplicación (las pantallas de captura, llamada *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.

Definiciones básicas

Objeto: Ente con estructura constituida por propiedades o atributos que definen el estado asociado a esa estructura, un comportamiento definido y una forma única de identificarlos [Booch91].

Clase: "Conjunto de objetos cuya estructura (estado) y comportamiento es el mismo" [Booch91]. La definición anterior es incorrecta[17], ya que en realidad es la definición de *tipo de dato*. La clase es el conjunto de atributos, operaciones y reglas que un conjunto de objetos comparten (ver definición de Tipo de Dato Abstracto más adelante). Se puede decir que la clase es como un *molde* del cual se crean los objetos. Ejemplos:

```
PERSONA = { Juan, Pedro, Ramón, Arturo, José, Manuel }
persona = { Ojos, Naris, Boca, Cabello, Caminar(), 0 ≤ Edad ≤ 150 }
```

En este ejemplo, *PERSONA* denota un tipo de datos, mientras que *persona* denota la clase correspondiente a dicho tipo de datos. Es importante establecer la diferencia, ya que resultará un fundamento esencial para la discusión que en el capítulo III se presenta con respecto a la ecuación ***clase = dominio*** propuesta por [Date 95]

Instancia: Cada elemento de un conjunto de objetos de una misma clase. Es decir: instancia y objeto son términos intercambiables [Booch91].

Identidad: La propiedad de un objeto que permite distinguirlo en forma única de cualquier otro objeto. Este concepto es diferente de la igualdad entre objetos. Dos objetos pueden tener los mismos valores (el mismo estado) en cada una de sus propiedades diferentes de la identidad, y se dirá que son iguales; pero nunca la misma identidad, por que en este caso, en lugar de decir que son iguales, se dice que son *el mismo objeto*.

Estado: Estructura de un objeto. Conjunto de valores que adoptan las propiedades o atributos de un objeto en un momento dado en el tiempo [Booch 91].

---

[17] Aunque según comentó [Jazayeri 96], para efectos prácticos de implementación en C++ no se remarque la diferencia de conceptos (aunque evidentemente existe).

Comportamiento: Conjunto de operaciones de un objeto. Cómo actúa (métodos) y reacciona un objeto (eventos) en términos de sus cambios de estado y paso de mensajes [Booch 91].

Principios del modelo

Abstracción: El conjunto de características (atributos o propiedades) esenciales de un objeto, además de sus operaciones, que lo distinguen de otros tipos de objetos y por tanto proporciona los límites conceptuales definidos desde la perspectiva de quien lo visualiza [Booch 91].

Encapsulamiento: El ocultamiento de los detalles de un objeto que no contribuyen a sus características esenciales. Típicamente, la estructura de un objeto se oculta, así como la implementación de sus métodos. La *interfaz* es la parte de una clase que permite comunicar las características públicas de la misma a otras clases o a otras operaciones de la misma clase [Booch 91].

Jerarquía: Taxonomía o rango de ordenación de las abstracciones. Asociación (relationship) entre clases. Existen dos tipos básicos de jerarquía en un sistema complejo:

* **Herencia**: (jerarquía "es del tipo" o "is a kind of"). Dada cuando una clase comparte la estructura y/o comportamiento de una (herencia simple) o varias (herencia múltiple) otras clases. La clase heredada se convierte en *superclase* y la que hereda llega a ser *subclase*. Esta última puede re-definir o revocar algunas de las características o comportamiento heredados. También se le llama *relación de contención*.

* **Agregación**: (jerarquía "es parte de" o "is part of"). La relación que existe entre dos clases cuando una es componente de la otra. Se conoce también como *relación de uso*.

Tipos: Conjunto de valores y operaciones que un objeto puede tener. Se dice que un lenguaje de programación está fuertemente tipificado si garantiza que todas las expresiones son consistentes en su tipo (que no hay traslape o cambios dinámicos de tipo). Es importante mencionar dos conceptos relacionados con la tipificación:

Enlazado estático: Proceso en el que se verifican los tipos de los atributos definidos en los objetos en tiempo de compilación.

Enlazado dinámico: Proceso en el que la verificación de tipos se lleva en el momento de ejecución, asignando el tipo de cada dato al momento de construir dinámicamente el objeto.

Tipo de dato abstracto (TDA): Modelo ideal de un conjunto de datos en el cual están asociados dicho conjunto, el conjunto de operaciones entre esos datos, y las estructuras de datos que permitirán implementarlos [Aho Ullman Hopcroft 88]. Más ampliamente, el TDA es la especificación formal que permitirá construir una clase[18].

Concurrencia: Capacidad que distingue a un objeto activo de otro que no esté activo [Booch 91] en un momento dado *t*. Con *activo* en un momento *t* nos referimos a un objeto que esté ejecutando algún proceso en dicho momento.

Persistencia: Característica de un objeto de trascender en el tiempo y el espacio. Se implementa como procesos de almacenamiento secundario en una máquina, y es concepto fundamental para la implementación de una base de datos modelada bajo el paradigma de orientación a objetos.

Re-usabilidad: Característica del modelo que permite reutilizar clases predefinidas para crear nuevas clases sin necesidad de re-implementar todas las propiedades, sino sólo aquellas particulares de la nueva clase. Esto es posible gracias al concepto de *herencia*.

Modularidad: Propiedad de un sistema de poder descomponerse en un conjunto de piezas (módulos) cohesivas vinculables a través de una interfaz; pero independientes en su estructura y proceso interno.

Polimorfismo: Capacidad de definir operaciones con la misma nomenclatura para diferentes clases de objetos, con la consecuente diferencia de implementación en cada clase. De acuerdo al objeto al que se haga referencia, será la operación con la que responda. Por ejemplo: la operación "avanzar" existe tanto para el ser humano como para los vehículos, sólo que la forma de "avanzar" de uno y otro es totalmente diferente. En los lenguajes modernos de programación este concepto es llamado *sobrecarga de operadores*.

Existen diferentes tipos de clases, a saber:

Abstracta: Una clase que no tiene instancias y algunos de sus métodos no están completamente implementados. Se define con el fin de ser superclase de nuevas clases no

[18] En el capítulo V, al hablar del estándar ANSI-SQL3 se retomará el concepto del TDA.

abstractas a las que se les agregarán o modificarán propiedades y operaciones. La clase "persona" (con atributos como nombre, edad, domicilio, etc.) puede ser una clase abstracta a partir de la cual se definan clases como "empleado" para una base de datos de nómina, "cliente" para un sistema de ventas, "alumno" para un sistema escolar, etc.

Contenedora: Clase cuyas instancias son colecciones de otros objetos. Estas clases pueden denotar colecciones homogéneas (todos los objetos de una colección son de una misma clase) o heterogéneas (cada uno de los objetos en una colección puede ser de diferente clase, aunque todos deben compartir la misma superclase). Ejemplos típicos son las listas, los arreglos, los conjuntos únicos, etc.

Genérica (parametrizada): Clase que sirve de plantilla para otras clases, en la cual la plantilla se parametriza por otras clases, objetos y/o operaciones. Una clase genérica debe instanciarse (llenar sus parámetros) antes de que se puedan crear objetos de ella. Este tipo de clases se utilizan normalmente como clases contenedoras.

Metaclase: Clase de clases. Una clase cuyas instancias son clases. Una plantilla que indica las propiedades y operaciones que toda clase debe contener. Por ejemplo, el método de creación de objetos (:Create(), :New(), :Instantiate(), :Build(), :Initiate(); según el lenguaje) es una operación de una metaclase.

Invariantes, precondiciones y postcondiciones

Adicionalmente a la identidad, el estado y comportamiento de los objetos, existen una serie de condiciones que son parte intrínseca de ellos y que deben estar encapsuladas[19].

Estas condiciones permitirán conservar la integridad y consistencia de los objetos. Son expresiones lógicas (predicados) que validan o verifican la ejecución de algunas de las operaciones propias del objeto. Bertrand Meyer [MeyerPHI88] les llama *invariantes*. Otro tipo de predicados lógicos que garantizan la estabilidad de un objeto al ejecutarse un método, son las pre y post condiciones asociadas a dichos métodos:

- *Precondiciones:* La expresión o expresiones lógicas a evaluar antes de ejecutar una operación, validando que los parámetros propuestos para dicha operación cumplan con

[19] Este concepto no es generalmente establecido como fundamento del modelo orientado a objetos. Sin embargo, es una característica adicional que todos los objetos poseen, y que un modelo de datos en general [Date 95] debe considerar. Su equivalente directo con el modelo relacional son las *reglas de integridad*, y consideramos que debería ser parte fundamental del modelo.

las restricciones definidas conceptualmente en el objeto.

- *Postcondiciones:* Conjunto de reglas con las que se asegura que el objeto regrese al objeto de quien recibió el mensaje, el resultado consistente requerido.

Al ejecutarse una operación (método), se evalúa una intersección de condiciones (un AND) entre las invariantes propias del objeto y las pre y post condiciones del método.

Entre los objetos existe una interacción a través de operaciones a las que se les llama *envío de mensajes*. Un objeto de una clase envía un mensaje a otro objeto de otra clase esperando una respuesta o resultado. Meyer indica que estas operaciones son del tipo *contrato Cliente - Proveedor*; refiriéndose a que una clase asegura regresar un resultado válido (postcondición) siempre que la clase que envía el mensaje original proporcione las condiciones y los parámetros válidos (precondición).

Un ejemplo típico de lo anterior es la precondición en una clase *pila*, donde la operación *pop* debe verificar que no esté vacía antes de tratar de regresar un valor. De otra forma, el valor entregado por la operación *pop* sería inválido o *nulo*.

Administración de versiones

Aunque no es un aspecto definido universalmente en el modelo, se hará hincapié en un aspecto que la mayoría de los sistemas orientados a objetos contemplan de diferentes formas: la administración de versiones de los objetos. Esta administración consiste en conservar durante un cierto tiempo diferentes versiones de un mismo objeto. Es decir, la capacidad de que al efectuar alguna actualización, los valores originales del estado de un objeto sean conservados en algún lugar del sistema. Esto permitirá analizar la historia del cambio de estados de un objeto y eventualmente poder corregir errores o guardar consistencia y validez en la base de datos.

Varía de sistema a sistema la forma de administrar este concepto (los más eficientes conservan los valores anteriores de sólo aquellos atributos modificados, agilizando la actualización y ocupando menos espacio) ; pero en los sistemas basados en el modelo relacional lo más que puede hacerse es recuperar un respaldo y reejecutar *(redo)* las transacciones almacenadas en la bitácora (log) hasta un punto dado.

Bibliografía

[Booch91] Object Oriented Design with Applications, Grady Booch, *1991, The Benjamin/Cummings Publishing Company, Inc.*

[Meyer88] Object-oriented Software Construction, Bertrand Meyer, *1988, Prentice Hall International.*

[Weitzenfeld94] Paradigma orientado a objetos, Dr. Alfredo Weitzenfeld, Marisol González Lozano, ITAM, Marzo de 1994.

[Yourdon93] Análisis y diseño estructurados modernos, Edward Yourdon, *1993, Technology Training*

[Jazayeri 96] Programming Languages: Data types, Medhi Jazayeri, *1996, VII Escuela Internacional en Temas Selectos de Computación. La Paz. B.C.S.*

[Aho Ullman Hopcroft 88] Estructuras de datos y algoritmos, Alfredo V. Aho, Jeffrey D. Ullman, John E. Hopcroft, *Addison Wesley Iberoamericana, 1988*

III.- DOMINIOS O RELACIONES, EL DILEMA

En los siguientes párrafos se presenta una serie de razonamientos que intentan explicar o contestar a la problemática resaltada por [Date Darwen 94] en la *no* convergencia del modelo relacional con el orientado a objetos. Este capítulo constituye el corazón de la tesis por contener la aportación que pretendo ofrecer en un nuevo y emergente modelo de datos, tratando de proporcionar un criterio de cómo se da esta convergencia (criterio que es resultado de diferentes experimentaciones en la práctica personal y se refuerza con los criterios de muchos otros autores) ; en contraposición con la *no convergencia* que aducen los primeros autores mencionados. En el artículo [Camps 96] existe un planteamiento similar al expuesto en esta tesis; sólo que su publicación resultó anticipada a la conclusión de este trabajo. El orden en que son presentados no tiene ninguna jerarquía en particular, ya que cada uno es independiente del otro en términos generales; y su presentación lleva un orden similar a lo expuesto por [Date 95] y [Won 95]

Encapsulamiento

[Date Darwen 94] dicen que el principio de encapsulamiento en las bases de datos orientadas a objetos es violado con las sentencias de actualización de SQL. Esto porque el modelo orientado a objetos establece que no debe ser accesado atributo alguno de una instancia salvo a través de métodos. De esta manera, es posible conocer (consultar) el valor de un atributo dado; y deben implementarse métodos que permitan actualizar y validar las reglas para actualizar atributos (precondiciones y postcondiciones). Esto es, en lugar de ejecutar un proceso de la siguiente forma:

```
objeto.atributo_x := nuevo_valor
```

deberá llevarse a cabo de esta manera:

```
objeto.fija_valor_atributo_x ( nuevo_valor )
```

En donde estará implementado el método *fija_valor_atributo_x()* con las invariantes necesarias para guardar la consistencia de la información. No obstante, es cierto que en un lenguaje orientado a objetos resultará problemático, como menciona [Date 95], codificar un método diferente para la actualización de cada campo. Sin embargo, este problema se resuelve parcialmente al tener definido en las metaclases un método de actualización que

pueda editarse para los casos particulares que requieran reglas adicionales.

Integridad referencial

Considero que así como la integridad referencial en una base de datos relacional se almacena en el catálogo, en una base de datos de objetos puede ser asociada a cada una de las clases a las que afecta, definiendo las reglas directamente en dichas clases que equivalen a las entidades relacionales. El caso de las validaciones para la actualización de atributos que se mencionaba en la sección de encapsulamiento, es un ejemplo perfecto de las reglas de integridad.

Consultas ad-hoc

Existirán consultas preprogramadas en las diferentes clases de objetos. Sin embargo, las consultas ad - hoc no son plenamente posibles en el modelo orientado a objetos [Date Darwen 94] (este tipo de consultas son las no programadas y que resultan de una necesidad específica que algún usuario de la bases de datos tenga en un momento dado ; donde la información requerida está ahí pero no existe el método o procedimiento encapsulado en los objetos o clases prediseñados para dicho esquema de información a consultar) . Una posibilidad que personalmente he experimentado es la creación de objetos *consulta*, los cuales consideren la interacción de una o más clases, además de un conjunto de reglas (invariantes) equivalentes a la cláusula WHERE del SQL. Un reporte, un listado en pantalla , etc.; finalmente son objetos que toman como parámetro colecciones de otros objetos. Una vista relacional, de hecho, es un objeto también.

Este enfoque resulta interesante; pero otros autores (y el modelo de objetos estándar ODMG así lo plantea) proponen el uso de un lenguaje de manipulación de datos paralelo al lenguaje de las bases de datos relacionales, con extensión de capacidades y que opere sobre colecciones de objetos en lugar de relaciones. Esto nos proporcionará diversas ventajas, como la desaparición de cierto tipo de joins, la anidación de cláusulas **SELECT**, etc. [Loomis 94]. En el capítulo de estándares y lenguaje OQL se analizarán con más detalles estos beneficios.

Sin embargo, es importante notar que la adopción de esta alternativa no constituye

un *mapeo* del álgebra relacional al modelo orientado a objetos. Realmente las operaciones del álgebra relacional constituirían un conjunto de operaciones propias de los objetos definidos en la base de datos.

Clases como dominios, relaciones anidadas

[Date Darwen 94] consideran en su documento *The Third Manifesto*, que el mapeo de las relaciones a los objetos constituyen una aberración y son incorrectas completamente. Afirman que la única ecuación válida entre estos dos modelos es la denotada por:

dominio = clase

Atendiendo a la definición de dominio en el modelo relacional, y el de clase en el modelo orientado a objetos; se puede encontrar que no son iguales dado que uno es un conjunto de datos y la otra un conjunto de propiedades. Por ende, la afirmación de [Date Darwen 94] de que la relación no es mapeable a la clase no está completamente explicada en el documento en cuestión.

Lo que sucede es que el mapeo que no puede darse a la clase es :

clase = relación (1)

sino más bien :

clase = esquema de la relación. (2)

y cada una de las tuplas de una relación corresponderá a cada instancia de la clase correspondiente.

Sin embargo, la afirmación anterior no implica que siempre sea mapeada *exactamente* una clase al encabezado de cada tabla. Digamos, una base de datos con ocho tablas no tendrá que ser equivalente a un esquema de ocho clases en un diagrama de objetos. Al tener un conjunto de datos originalmente modelado en términos de tablas y redefinirlos en clases, se pueden tener los siguientes casos:

- Alguna tabla corresponderá a una colección de objetos de una clase equivalente.
- Alguna tabla corresponderá a *más* de una colección del mismo número de clases.
- Algún número de tablas (mayor a 1) corresponderá a una sola clase.

Bajo el esquema anterior, efectivamente la ecuación (2) es incorrecta como afirman [Date Darwen 94] si generalizaramos en todas las tablas; pero hay que considerar que eso no

significa que no pueda darse el caso en al menos algunas de las tablas.

En [Ambler 95] se muestra un ejemplo sencillo pero ilustrativo de cómo un esquema de cinco tablas queda convertido en un esquema de clases (incluyendo clases abstractas) con seis clases interrelacionadas. Sin embargo, dado que su explicación resulta larga para citarla completamente en este trabajo, simplemente tomemos como ejemplo una base de datos escolar en donde tenemos las siguientes tablas :

```
ALUMNOS( MATRICULA, NOMBRE, FECHA_INGRESO , GRADO)
MATERIAS( CLAVE_MATER, DESCRIPCION, PROMEDIO_MINIMO_APROBATORIO )
CURSO( MATRICULA, CLAVE_MATER )
```

En esta base de datos se han modelado tres tablas, de las cuales dos son tablas base y la tercera es una tabla que establece la asociación entre las otras dos. Al modelar esta pequeña base de datos en un esquema orientado a objetos, podremos tener las siguientes clases :

```
Clase alumnos { Matricula : string(n1), nombre : string(n2),
                fecha_ingreso : date, grado :int,
                materias_inscrito :collection_of materias }

Clase materias { Clave_materia :int, descripción : string(m1),
                 promedio_mínimo_aprobatorio :real,
                 alumnos_inscritos : collection_of alumnos }
```

Como se puede observar, el esquema de tres tablas en el modelo relacional pudo representarse con un esquema de dos clases en el orientado a objetos, utilizando el concepto de colección (clase contenedora -ver pág. 30-), que representa un subconjunto de objetos de la clase referida. Las propiedades `materias_inscritos` en la clase alumnos y `alumnos_inscrito` en la clase materias hace referencia a objetos de otras clases, cuya declaración es totalmente transparente de la implementación. Si ésta última se lleva a cabo con apuntadores, índices o cualquier otra técnica ; es responsabilidad explícita del implementador.

Así mismo, en [Won 95] se afirma que existe la creencia de que el modelo orientado a objetos no es otra cosa que regresar al modelo jerárquico de bases de datos; y que entonces es retroceder, pues una de las metas del modelo relacional fue atacar las ineficiencias del

jerárquico (y del de red). Sin embargo, el modelo orientado a objetos considera también las operaciones entre entidades (comportamiento), además de modelar las relaciones de herencia entre entidades; que en el jerárquico no eran consideradas.

Lo que sí es cierto es que el modelo relacional, a través de la normalización ataca la complejidad de los objetos y busca la descomposición de éstos en elementos más simples de una manera consistente, hasta llegar a entidades cuyos atributos sean atómicos. En el modelo orientado a objetos, esta normalización ya no se da hasta ese nivel, y las entidades pueden llegar a tener atributos compuestos.

Sin embargo, el que una entidad contenga atributos compuestos no contradice el modelo relacional en términos de tablas, atributos y asociaciones. Si se considera a una relación con un conjunto de atributos no atómicos (compuestos) como otro dominio; se puede tener que otra relación que utilice a la primera contiene un atributo cuyo valor está tomado del dominio de los elementos de la segunda relación. Formalmente:

Sean R_1 y R_2 relaciones asociadas entre sí:

- En el modelo relacional se tendrá un atributo A tal que cada valor de A en R_1 se puede encontrar como valor de A en alguna instancia de R_2.
- En el modelo orientado a objetos, se tendrá que el atributo A toma valores del dominio C_1 (siendo C_1 un dominio de valores compuestos NO ATÓMICOS equivalente a R_1).

Bajo esta formalización, las instancias de una *clase* y la clase misma constituyen un *dominio* porque el *mapeo* que pueda existir entre la relación R_2 y una clase C_2 equivalente, resulta en un mapeo de dominio al convertirse R_2 en un dominio compuesto.

Sin embargo, la redundancia que la normalización pretende desaparecer existe si el modelador de bases de datos no está suficientemente capacitado para evitarla. Esto quiere decir que aún definiendo tablas, es posible que un modelo de datos contenga elementos redundantes si no son plenamente identificadas las llaves primarias, las foráneas y las dependencias funcionales. Esto sucede a menudo en la práctica con los sistemas relacionales más robustos que existen en el mercado cuando el modelador no es capaz de normalizar correctamente.

Dado lo anterior, para que la redundancia efectivamente desaparezca, es necesario que el modelador comprenda perfectamente la semántica de sus datos, lo que conlleva

precisamente un buen diseñador de base de datos podrá especificar en términos de clases y objetos la estructura de su modelo sin redundancia. El ejemplo de los alumnos y materias no tiene redundancia alguna, ya que la declaración `collection_of` no significa una *copia* de los objetos de la clase en cuestión, sino una referencia indirecta (que en términos de implementación puede lograrse con apuntadores ; pero como se mencionó anteriormente, estos detalles no necesitan ser conocidos por el modelador).

Modelo matemático subyacente

El modelo orientado a objetos no cuenta con un modelo matemático formal aceptado universalmente. Esto lo pone en franca desventaja contra el modelo relacional, el cual tiene como fundamento la teoría de conjuntos y los conceptos matemáticos de relación; además de tener perfectamente definidos un conjunto de operaciones (álgebra relacional).

Existen diversas propuestas de modelos formales para el paradigma de orientación a objetos; pero hasta la fecha no se ha publicado oficialmente ninguna. No obstante, es importante considerar que estas propuestas están suficientemente fundamentadas matemáticamente; pero han surgido como resultado de la necesidad del modelo formal cuando el paradigma surgió de problemas de implementación. Esto es: las características del modelo orientado a objetos se han definido y se han ido refinando a través del tiempo, resolviendo problemas prácticos sin que hubiera existido la preocupación de establecer una definición formal. Hoy día se está trabajando en esa definición pero aún no se ha unificado.

Sin embargo, las propuestas de modelos formales para el modelo orientado a objetos están fundamentadas principalmente en la teoría de conjuntos y el cálculo de dominios, al igual que el modelo relacional. Se puede decir que solamente falta que estas propuestas se hagan *oficiales*. En efecto, la posibilidad de visualizar una relación como un dominio cuando va a ser utilizada como elemento foráneo de otra relación, implica que las operaciones de consulta entre objetos de una clase y otra den como resultado *colecciones* de objetos. Una colección de objetos representa un dominio dado, y de esta manera toda operación entre clases e instancias dará como resultado dominios. Conclusión: el cálculo de dominios resulta una formalización válida para el modelo orientado a objetos.

A continuación se enunciarán y comentarán los aspectos generales de algunos

modelos formales propuestos para este paradigma:

Una teoría de objetos

[Cardelli Abadi 95] ponen a consideración una interesante aportación con respecto a la formalización de objetos, que probablemente llegue a ser la aceptada universalmente. Introducen el concepto de cálculo de objetos (object calculli) que está fundamentado en el cálculo lambda. Proporcionan un cálculo simple basado en teorías de ecuaciones que permitan comprender y modelar mejor la herencia cuando se topa uno con la problemática de refinar un método en una subclase, o redefinirla completamente.

Describe un cálculo de objetos de primer orden sin tipo, así como sistemas de segundo orden donde se habla del tipo Self (es decir, referencias al mismo objeto, que en subclases heredadas puede ocasionar problemas al constituir un dato de tipo *dinámico*).

Los autores de este modelo han creado un lenguaje, llamado *Obliq*, cuyas principales características incluyen el soporte de objetos distribuidos, muy necesarios en la actualidad en que el procesamiento de datos a través de Internet es cada vez más frecuente. Definen una colección de operaciones simples para crear y manipular estructuras de registros. En este modelo los registros son asociaciones finitas de valores a etiquetas. Un sistema de tipos de segundo orden soporta tanto subtipificación como polimorfismo.

Modelo ontológico

Propuesto por [Yair88] basado en el concepto filosófico de la ontología[20]. Él establece inicialmente que la falta de universalidad en la definición de un modelo formal de objetos es resultado del hecho de que este paradigma surgió de aspectos de programación e implementación, de manera que su propuesta se desplaza de una orientación de implementación a una orientación de modelado formal. [Yair88] propone que el beneficio del paradigma va más allá de las mejoras en el control de datos, portabilidad, encapsulamiento o reusabilidad. Más bien, dice, los objetos son importantes por que reflejan una visión "natural" del mundo y la realidad que se está modelando. Por tanto, propone un enfoque ontológico cuyos principios[21] son:

[20] La ontología es la rama de la filosofía que trata el modelado de la existencia de las cosas en el mundo.

[21] Según Bunge, (M. Bunge, Treatise on Basic Philosophy: Vol. 3: Ontology I: The Furniture of the World, Reidel,

- El mundo se compone de cosas (naturales o sociales).
- Las formas son propiedades de las cosas.
- Las cosas se agrupan en sistemas o agregados de componentes que interactúan entre sí.
- Todas las cosas cambian.
- Ninguna cosa viene de la nada, y ninguna cosa se reduce a la nada.
- Todas las cosas están regidas por reglas (naturales, sociales, etc.)

Describe lo que son las cosas y propiedades (estas últimas como el conjunto de partes que conforman las cosas) ; estableciendo la diferencia entre las consas sustanciales y las conceptuales (equivalente a las relaciones *base* y las *vistas*).

Hace hincapié en que el mundo está formado fundamentalmente por entidades, las cuáles son percibidas a través de sus propiedades, y las propiedades se materializan en términos de atributos (abstracción). Introduce el concepto de *esquema funcional*, que constituye el conjunto de reglas que validan que una cosa se encuentre en un estado válido. Describe un evento como el proceso a través del cual una cosa cambia su estado (cambia los valores de sus propiedades). Finaliza redefiniendo los conceptos expresados en términos de cosas, pero ahora especificándolos como objetos. El paralelo entre cosa y objeto es evidente ; pero parte de las cosas porque éstas son el fundamento de la ontología.

Modelo matemático para objetos.

[Pons Giandini Baum 95] hacen un recuento de las diferentes aportaciones de modeloes teóricos para objetos; y resumen un análisis propio dividiendo la construcción de un sistema en tres niveles :

- Nivel de especificación de objetos
- Nivel de implementación
- Nivel de programas de aplicación.

Para estos autores argentinos, un objeto se modela como un par <v,B> donde v es el valor del objeto (el conjunto de valores del estado) y B es el conjunto de todos los métodos que el objeto puede ejecutar. A su vez, B está particionado en tres subconjuntos : métodos creadores, observadores y mutadores. Los primeros son los que instancian una clase e

---

Boston, 1977)

inicializan valores del estado de un objeto. Los segundos son los que permiten inspeccionar el valor de las propiedades de un objeto, y los últimos son aquellos que modifican los valores de dichas propiedades (en el capítulo donde se analiza ANSI-SQL3 en la página 79 se habla de los métodos visualizadores *Get_* y los modificadores *Set_* para los campos de una tabla, correspondientes a estos dos últimos subconjuntos de métodos; este concepto refuerza también los aspectos de encapsulamiento analizados en la página 33 de este mismo capituo) .

Ahonda en el concepto de la autoreferencia de un objeto (self), que puede tener lugar en cualquier método definido en la clase a la que dicho objeto pertenece. Presenta una interesante discusión respecto a cómo esta referencia puede dar problemas al programar la herencia. Plantea la pregunta : cuando se hereda una clase, al ejecutar un método modificado en la sub_clase, debe ejecutarse después de procesar la porción de la superclase, debe ejecutarse en lugar de, o debe haber mecanismos que permitan hacer referencia al método definido en la superclase bajo ciertas condiciones? Esta pregunta no está formalmente contestada y cada implementador aplica su propio criterio.

Resumen

Como puede observarse, las diferentes propuestas presentadas, así como otras existentes ([AlhajjArkun 93]) ofrecen más o menos conceptos expresados en términos formales; lo que habla de un interés por ofrecer un modelo tan robusto como el relacional. La realidad es que el fundamento del modelo orientado a objetos es, al igual que el relacional, la *teoría de conjuntos*. Quizá el mayor problema es que han surgido varias propuestas pero no se ha publicado alguna de ellas como la universal. Esto llegará tarde o temprano, ya que todas estas propuestas contienen una serie de elementos en común y no carecen del formalismo necesario.

Joins no necesarios

En [Won 95] se habla de la creencia generalizada de la posibilidad de eliminar Joins[22] en un OODMS. Existen propuestas de que la declaración de atributos como OIDs[23] de atributos de otra clase constituyen la eliminación de joins. (Por ejemplo, al definir una ciudad en una

[22] En este texto se utiliza la palabra Join o Juntado para no confundirlo con la unión de conjuntos, ya que la unión es una operación totalmente diferente al join, según se explicó en el primer capítulo.

clase "cliente", donde el atributo ciudad es un apuntador a una clase "ciudad" que contiene todas las ciudades del país). En el modelo relacional efectivamente sería necesario ejecutar una operación JOIN para obtener el dato asociado a esta llave foránea (ciudad). En el modelo de objetos, por otro lado, puede directamente consultarse la ciudad sin ejecutar el join:

Relacional:

```
SELECT cliente.nombre , ciudades.ciudad
  FROM cliente , ciudad
 WHERE cliente.cve_ciudad = ciudad.cve_ciudad
```

Objetos:

```
SELECT cliente.nombre, cliente.ciudad.nombre
  FROM cliente
```

Sin embargo, esto no significa que los JOIN ya no tengan sentido en las consultas de bases de datos de objetos. Simplemente existirán muchos joins que podrán suprimirse (en el modelo relacional todos los joins son necesarios); pero habrá otros casos en la misma base de datos de objetos que el join *sí* tenga sentido.

Pese a ello, sí es importante que este planteamiento no sólo mejora la legibilidad de una expresión de consulta (el **SELECT** de objetos es más claro que el de relaciones); sino que en cuestiones de desempeño, el no ejecutar un join trae consigo una serie de ventajas en rapidez, ya que no se ejecuta un producto cartesiano sino que los OIDs implícitos permiten accesar directamente los elementos que entran en el resultado de la consulta.

OIDs sustituyen llaves foráneas

También en [Won 95] se hace referencia a esta errónea creencia y se explica porqué es errónea. El manejo de OIDs permite manipular asociaciones (relationships) entre entidades; pero frecuentemente es necesario un apuntador de regreso, en primer lugar para poder ejecutar consultas sin necesidad de recorrer todos los elementos de la primer entidad.

Retomemos nuevamente el ejemplo la base de datos con una clase *alumno* sobre la cual está declarado un conjunto de *materias*. Este conjunto es una *colección* que contiene una serie de referencias a instancias de la clase *materia*. En el modelo relacional este hecho, como se vio anteriormente, estará materializado con una tercer tabla que incluya la llave de la

[23] Object IDentificators (identificadores de objetos)

tabla de alumnos y la llave de la tabla de materias. El conjunto de materias de un alumno X estará dado por el conjunto de tuplas en las que la llave del alumno corresponda al objeto X.

Con esta tercer tabla, en el modelo relacional es posible consultar todos los alumnos que cursan una materia M1 dada simplemente seleccionando todas las tuplas en las que la clave de la materia sea M1 y ejecutando un join con la tabla de alumnos.

En el modelado de clases de objetos, para conocer esta información, será necesario recorrer cada una de las instancias de alumnos y de forma iterativa inspeccionar los objetos a los que hace referencia la colección *materias* definida, extrayendo las referencias de aquellas ocurrencias correspondientes a la materia M1 consultada.

Para evitar lo anterior, es necesario definir en la entidad de materias un atributo adicional de regreso que sea del tipo colección que contenga el conjunto de referencias de los alumnos inscritos en esa materia. Formalmente tenemos:

Modelo relacional:

Estructura:

```
ALUMNOS( matricula: char, nombre: varchar, edad: int, ... )
MATERIAS( cve_materia: int, descripcion: varchar, ... )
ASOC_ALUM_MATER( matricula, cve_materia )
```

Consultas:

- Materias cursadas por un alumno X

```
SELECT materias.descripcion
  FROM materias, asoc_alum_mater
 WHEREmaterias.cve_materia = asoc_alum_mater.cve_materia
   AND asoc_alum_mater.matricula = "alumno_X"
```

- Alumnos que cursan la materia M1

```
SELECT alumnos.nombre
  FROM alumnos, asoc_allum_mater
 WHERE alumnos.matricula = asoc_alum.mater.matricula
   AND asoc_alum_mater.materia = "M1"
```

Modelo orientado a objetos:

Estructura:

```
ALUMNOS( matricula:char, nombre:varchar,
         edad:int, Materias_Inscrito:set of MATERIAS, ... )
MATERIAS( cve_materia:char, descripcion:varchar,
          Alumnos_cursantes:set of ALUMNOS, ... )
```

Consultas:

- Materias cursadas por un alumno X

```
SELECT alumnos.materias.descripcion
  FROM alumnos
 WHERE alumnos.matricula = "alumno_X"
```

- Alumnos que cursan la materia M1

```
SELECT materias.alumnos.nombre
  FROM materias
 WHRE alumnos.materia = "M1"
```

Como puede observarse, es posible ahorrarse una operación de join (el ODBMS debe ser capaz de actualizar en forma transparente las referencias tanto de *Materias_Inscritos* en **alumnos** como de *Alumnos_Cursantes* en **materias**); pero finalmente es necesario en ambas entidades declarar atributos que sirvan como llave foránea.

Así mismo, [Date 95] afirma que el manejo de referencias (que en la implementación corresponden a OIDs) redunda en problemática para el usuario final, ya que lo obliga a meterse en detalles de implementación. El ejemplo mostrado en esta página demuestra también que el concepto de OIDs es parte del modelo orientado a objetos; pero no tiene porqué reflejarse su implementación para el usuario final. Una declaración del tipo:

```
... Materias_Inscrito: SET OF materias ...
```

deberá ser administrada en forma totalmente transparente por la implementación del ODBMS, no teniendo el usuario que lidiar con los OIDs físicos (como afirma Date que tendría que hacerlo).

Conclusión: las llaves (tanto primarias como foráneas) no son eliminadas en el modelo orientado a objetos; simplemente existirán algunos casos en que las llaves puedan ser sustituidas por OIDs manejados transparentemente por el ODBMS, pero habrá otros casos en que simple y sencillamente la llave constituya un atributo primordial del objeto a modelar (# de seguro social en una tabla de empleados, folio de una factura, etc.). Al mismo tiempo, la eliminación de redundancia depende más de la habilidad del modelador que del modelo utilizado.

Datos multimedia en OODBMS

En la misma referencia [Won 95] se explica la afirmación falsa por parte de algunos fabricantes de OODBMS de que los sistemas orientados a objetos manipulan los datos multimedia mejor que los relacionales. Efectivamente los sistemas orientados a objetos manipulan este tipo de información; pero no resuelven en forma directa los problemas de almacenamiento, recuperación o actualización de los mismos. Finalmente cada producto tiene implementados estos procesos de manera propia. No existe *aún* una definición formal para el manejo de datos BLOb (Binary Large Objetct) ni para el modelo orientado a objetos ni para el relacional.

¿Sustituirá el modelo de objetos al relacional ?

Existe una creencia generalizada de que las bases de datos relacionales que durante más de 25 años han existido, jamás podrán ser redefinidas en un nuevo modelo (sea el de objetos o sea cualquier otro) ; pues dada la inversión de las grandes compañías en este tipo de bases de datos, no permitirán experimentos con un nuevo modelo sobre aplicaciones de misión crítica cuyas bases de datos son lo suficientemente robustas en el modelo relacional.

La realidad es que el modelo relacional tiene algunas lagunas en el manejo de la información, que el modelo orientado a objetos ataca de una manera más eficiente. Sin embargo, también es cierto que el modelo relacional ataca una serie de aspectos de una forma más efectiva que el orientado a objetos. De ahí que poco a poco ambos modelos se fusionarán (mucho de esto en base a los estándares que actualmente se están definiendo y que poco a poco convergen en conceptos) y generarán un nuevo modelo que no será relacional totalmente, sino que permitirá visualizar y diseñar la información desde enfoques *ortogonales* pero *complementarios*. (Aunque sí existen algunos aspectos que desaparecerán del relacional y se manejarán 'a la OO).

Por otro lado las grandes inversiones en sistemas de modelos relacionales no se perderán con la introducción de nuevos productos orientados a objetos, ya que en todo proceso de cambio o transición es necesaria una etapa de adaptación. En este caso será necesaria la reingeniería de la información en las empresas basándose en el modelo orientado a objetos; pero utilizando aquellas herramientas que proporcionen una *capa* de diseño que permita en forma gradual llevar a cabo esta transición, que internamente opere sobre tablas

relacionales aunque el resultado final lo exprese en términos de objetos. Prueba de esta *ortogonalidad* es el hecho de que el lenguaje OQL (por definir en el capítulo V), soportará la instrucción **SELECT** y el álgebra relacional sobre objetos sin ignorar la definición de *tipos de datos abstractos*.

De hecho, existen importantes empresas que actualmente están desarrollando bases de datos utilizando los diferentes productos orientados a objetos que existen hoy en el mercado (y que algunos se mencionan en el siguiente capítulo). AT & T entre otras. Así mismo, compañías fuertes tradicionalmente en las bases de datos relacionales, están incorporando cada vez más características de orientación a objetos en sus productos: Sybase (con su OBJECT-CONNECT), Informix (a través de UNIVERSAL-SERVER que está basándose en el lenguaje SQL3 - a explicar más adelante -), Oracle (a través de un concepto Universal-Server también), DB2 de IBM (con su DATABASE 2 que ya soporta tipos de datos y funciones definidos por el usuario), etc.

En lo que respecta a los más de veinticinco años de robustez del modelo relacional; la realidad es que este modelo no ha sido implementado en forma totalmente pura, sino que los fabricantes se han visto en la necesidad de incluir características que no pertenecen al modelo (caso concreto, los procedimientos almacenados :***stored procedures***). Además, los modelos prerelacionales también tienen muchos años trabajando, incluso existen todavía bases de datos (principalmente en *main frames*) basadas en el modelo de redes o en el jerárquico, y no por eso se consideran mejores que el relacional. Sin embargo, siguen utilizándose precisamente porque la transición no es tan fácil ni aunque el modelo a querer implementar esté lo más *matemáticamente fundamentado*.

Incluso, el mismo modelo relacional ha sufrido de malas implementaciones; además de haber salido el primer producto que pretendía ser relacional (DB2 de IBM), catorce años después de definido el modelo original (1983) [Date 95]; y ciertamente diferentes productos a partir de entonces y hasta la fecha no han implementado en la forma totalmente teórica como está definido dicho modelo.

Podemos resumir los puntos en en discusión bajo la siguiente tabla :

Problema	Solución
Violación del encapsulamiento	No se da si la ctualización es a través de métodos
Carencia de integridad referencial	Esta se da a través de invariantes y apuntadores
No son posibles las consultas ad-hoc	Es posible la creación "ad-hoc" de objetos consulta
Se regresa al modelo jerárquico al considerar clases como dominios, relaciones anidadas	No es verdad, el modelo jerárquico no contemplaba comportamiento. El de objetos sí. El modelo relacional
Carece de modelo matemático subyacente	Existen varias propuestas, y pronto se tendrá una aceptada universalmente
Joins no necesarios	Falso, sólo algunos joins son sustituidos por anidación de relaciones o clases
OIDs sustituyen llaves foráneas	Falso. Surge el concepto de OIDs de reversa.
Datos multimedia en OODBMS se manejan mejor	No necesariamente. Siempre será necesaria la implementación de métodos que manejen este tipo de datos ; pero una vez definidos, pueden ser caja negra
¿Sustituirá el modelo de objetos al relacional?	No. Se complementarán ambos surgiendo un nuevo modelo que no será ni completamente orientado a objetos ni completamente relacional

Bibliografía

[Date 95] An introduction to Database Systems C.J. Date, *Addison Wesley 1995*

[Won 95] Modern Data Base Systems, Won Kim, *ACM Press 1995*

[Loomis 94] Querying object databases Mary E. S. Loomis, *Journal of Object Oriented Programming, Vol. 7, No. 3, June 1994, pp. 56 - 60, 78*

[Ambler 95] Mapping Objects to Relational Databases Scott Ambler, *Software Development, Vol. 3, No. 10, pp. 63 - 72.*

[Frank 95] Object - Relational Hybrids Maurice Frank , *DBMS, Vol. 8, No. 8, July 1995, pp. 46-54*

[Date Darwen 94] The third manifesto Hugh Darwen & C.J. Date *1994.*

[Loomis 95] Object Databases, The Essentials Mary E.S. Loomis , *1995, Corporate and Professional Publishing Group*

[Camps 96] Domains, relations and religious wars R. Camps. Escola Universitària Politècnica. Vilanova i la Geltrú. España , *1996, SIGMOD Record, Comunications of the ACM, Vol. 25, No. 23, Septiembre 1996*

[AlhajjArkun 93] An Object Algebra for Object-Oriented Database Systems, Reda Alhajj and M. Erol Arkun, *1993, Bilkent University, Faculty of Engineering, Ankara, Turkey*

[Pons Giandini Baum 95] Modelos Matemáticos para Objetos, Claudia Pons, Roxana Giandini, Gabriel Baum, *Panel 1995, XXI Conferencia Latino-Americana de Informática.*

[Yair88] A Proposal for a Formal Model of Objects, Yiar Wand, 1988

[Cardelli Abadi 95] A Theory of objects, Luca Cardelli, Martin Abadi, 1985

IV.- OODBMS ACTUALES

Generalidades

En este capítulo se describirán las principales características de los más importantes productos de software que actualmente existen en el mercado que intentan ser considerados como manejadores de bases de datos orientadas a objetos, algunos adicionalmente aprovechando también los beneficios de los conceptos del modelo relacional. El primero de ellos es GemStone; pero de los diferentes productos que han surgido, en los últimos meses han sido poco a poco acoplados al modelo orientado a objetos ODMG - 93. Para comprender cómo se diferencían los diferentes sistemas reseñados a continuación, es necesario revisar en forma generalizada los problemas que los sistemas de administración de bases de datos deben atacar. Existen fundamentalmente cuatro tecnologías diferentes para administrar datos : sistemas de archivos, sistemas de administración de bases de datos relacionales (RDBMS), sistemas de administración de bases de datos orientadas a objetos (OODBMS) y sistemas de administración de bases de datos de objetos-relacionales (ORDBMS). Cada uno tiene sus propias capacidades que lo hacen apropiado para administrar un conjunto de datos dado. El siguiente diagrama muestra la correlación que existe entre ellos dividiendo el universo de los datos en cuatro clases a través de dos ejes : el eje de la complejidad de los datos y el eje de la complejidad en las consultas [Informix96]:

Consultas	DBMS relacionales	DBMS relacionales-objetos
No consultas	Sistemas de archivos Servidores multimedia	DBMS orientados a objetos
	Datos simples	Datos complejos

Datos simples, sin consultas

Los sistemas de archivos funcionan perfectamente cuando se trabaja utilizando nombres de archivos dado que el número de éstos es manejable y la navegación la ejecuta el usuario directamente. A este grupo corresponden tareas como procesadores de textos, hojas de cálculo, programas de gráficas, etc. Las operaciones de búsqueda de archivos las lleva el

usuario al conocer de antemano la organización jerárquica de su sistema, o bien puede utilizar herramientas especiales de búsqueda de archivos. Los servidores de archivos multimedia (Video Servers, Audio Servers) trabajan de una manera en que el usuario (o la interfaz de software) hace referencia directamente al *nombre de archivo* que contiene el video o el audio requerido.

Datos simples, con consultas : DBMS relacionales

Los cinco principales vendedores de RDBMS (IBM, Informix, Microsoft SQL-Server, Oracle y Sybase) han penetrado en el mercado de administración de datos después de varios años y con fuertes inversiones de miles de millones de dólares. Un RDBMS moderno es escalable, robusto y manejable, proporcionando acceso a los datos pos su contenido, utilizando el lenguaje SQL estándar para consultas. Es una plataforma excelente para desarrollar aplicaciones para transacciones que requieran ejecutarse velozmente y con un conjunto reducido de tipos de datos.

Sin embargo, datos más complejos (BLOBs, servicios financieros, series de tiempo financieras, objetos en el World Wide Web de Internet, etc.) no pueden indexarse ni manipularse con el álgebra relacional pura.

Datos complejos sin consultas estructuradas : DBMS orientados a objetos

En recientes años varias empresas han desarrollado productos cuyo mercado aún no es suficientemente robusto (no cabe comparación con las empresas de RDBMSs) ; principalmente implementando métodos de persistencia de datos con protocolos para definición de esquemas de modelos de datos utilizando lenguajes como C++ o SmallTalk. Sin embargo, se están desarrollando cada vez más, capas que permitan consultar datos relacionales desde un sistema OODBMS o bien estructuras formales de consulta y manipulación de objetos complejos en dichos sistemas.

Datos complejos con consultas : DBMS relacionales de objetos

El enfoque que algunos productos han tratado de dar al nuevo modelo de datos que aprovecha las bondades del relacional y el orientado a objetos, es un conjunto de objetos relacionales ; donde los objetos pueden ser de cualquier complejidad, manejar jerarquías de herencias y de uso, encapsulamiento, polimorfismo, etc. ; pero al mismo tiempo considerando

el fundamento matemático del modelo relacional, considerando a una colección de objetos de una clase dada como una relación con dominios compuestos. De 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 [Stonbraker96].

Clasificación de productos

En las siguientes secciones se presentará un análisis de algunos productos clasificándolos de acuerdo a su naturaleza:

- Productos orientados a objetos con persistencia implementada.
- Productos relacionales con extensiones a objetos.
- ORDBMS diseñados desde el principio.

Para llevar a cabo dicho análisis, tomaremos en consideración cómo son soportadas las siguientes características (basado en el criterio de [Zand Collins Caviness 95]; aunque se han incluido productos que no aparecen en el documento de referencia, pero se ha obtenido información al respecto en otras fuentes:

- Principales aplicaciones
- Soporte o no de administración de versiones de objetos
- Recuperación
- Soporte o no de transacciones
- Manejo de objetos compuestos
- 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).
- Soporte de 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,

según la información que se obtuvo de Internet y de algunos manuales y libros consultados ; pero estas características no están definidas en un esquema global para todos los productos, sino que se mencionan conforme los manuales las describen :

I. 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. Algunos ejemplos son:

1.- Primer ODBMS: 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 bloqueo optimista y pesimista (no más detalles).

Soporte de base de datos distribuida: 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: Ya es un 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 OODBMS 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 las variables 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 "Class Defining Object" (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 DBMS debe tener, tales como (todas estas previamente definidas en los dos primeros

capítulos):

- 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.

Clases colección predefinidas. Existe un conjunto de subclases predefinidas heredadas de la clase colección para manipular conjuntos 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 intensional[24]. 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).

[24] Definida en el capítulo del modelo relacional

- 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 de la base de datos. Están implementados como mensajes para las instancias de las clases del tipo *colección*:

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

```
Cuadros select: {:s | ( 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]: Similar a **select**, sólo que con la operación contraria.

A detect: [x | predicado lógico]: Se ejecuta la búsqueda hasta encontrar al primer elemento que cumpla la condición.

Arquitectura del sistema

El sistema está 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 del usuario. Constituye un servidor para los programas de aplicación.

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 (no más detalles).

Herencia múltiple: Sí.

Concurrencia/bloqueo: Sí, 2 tipos de bloqueos (no más detalles).

Soporte de base de datos distribuida: 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 base de datos para aplicaciones escritas en C++. Dirigido a desarrolladores con soluciones interactivas, proporciona funcionalidad completa para ambientes distribuidos que corran en super micros de PCs en áreas locales (a través de Windows).

Este DBMS 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 (además de en PC), 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 :];
```

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 (no más detalles).

Soporte de base de datos distribuida: 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. Pero por otro lado, existe verdadera encapsulamiento (en forma de variables privadas y protegidas de C++), y herencia múltiple. Ontos serializa todas las transacciones concurrentes. Esto reduce la 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.

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, cuatro tipos de bloqueo.

Soporte de base de datos distribuida: 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.

Características 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. Éstas son atómicas y trabajan con un protocolo de grabación en dos fases (two phase commit)[25].

5.- O_2

Usos principales : CAD/CAM, GIS, OIS, Procesamientos financieros y transacciones

Administración de versiones : Limitada.

Recuperación : Sí.

Administración de transacciones: Sí.

Herencia múltiple: Sí.

Concurrencia/bloqueo: Sí, optimista.

Soporte de base de datos distribuida: 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 variable persistente. Incluye un lenguaje de consultas similar al select de

[25] Este tipo de actualización a dispositivo de almacenamiento consiste en un ambiente concurrente en enviar una señal a todos los usuarios de la base de datos solicitando se confirme si se puede grabar el dato del búfer al disco (fase 1) ; y cuando la respuesta es afirmativa por parte del software agente de cada usuario, el servidor poseedor de

SQL; últimamente adoptando los estándares definidos por la versión OQL establecida en el documento ODMG-93[26]. Dado que esta es la principal característica de este producto, será hasta el capítulo de estándares cuando se estudie con detalle su modelo de datos y los lenguajes de manipulación y definición que soporta.

II. Sistemas relacionales con extensión de orientación a objetos

Como se especificó en capítulos anteriores, 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 (Informix, por ejemplo, incluso ha solicitado recientemente personal con doctorado en ciencias de la computación y con experiencia y conocimiento de los estándares que están casi terminados en estas fechas, y que en el siguiente capítulo se reseñan ; de hecho sus próximos productos estarán basados en el estándar ANSI-SQL3 cuando éste esté completamente definido).

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 orientadas a objetos, se ha considerado 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.

---

la base de datos ejecuta el commit (actualización a disco, fase 2).

[26] Que son analizados con detalle en los siguientes capítulos

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 DBMS. Las columnas en una definición de tabla pueden declarase 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 se regrese un valor que sea utilizado como columna. Incluye también indexación B-Tree.

Interfaz: La interefaz de Illustra está dada con aplicaciones (llamadas DataBlades) puestas a disposición del usuario como API. Como 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 (próximas a ser liberadas) del lenguaje SQL3 de ANSI.

2.- UniSQL

Al igual que Illustra, en el documento fuente donde se normalizan las características de los productos analizados, UniSQL no está presente ; pero también es uno de los más importantes, por lo que a continuación es reseñado.

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, optimación 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.

III. Nuevos ODBMS diseñados desde cero

Como se especificó al principio de este capítulo, 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.

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 base de datos distribuida: No.

Evolución dinámica del esquema: Sí (limitada).

Datos multimedia: No.

Interfaz con lenguajes de programación: C, LISP, OSQL.

Estado actual de desarrollo: Prototipo de investigación aún...

Características especiales:

Desarrollado por Hewlett-Packard. Es resultado de la investigación cuyo prototipo desarrollado fue 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 (definen una asociación: por ejemplo: *TrabajaPara()*)

- 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 funcional que constituye un superconjunto semántico de SQL. QSQL 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.
- Intefaz de programación para C.
- Aplicaciones y herramientas de usuario.
- Manejador de objetos.
- Manejador de almacenamiento relacional y Funciones externas.

2.- Itasca

Usos principales : CAD/CAM, Int.Artif., 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 base de datos distribuida: 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 base 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. Está 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 intensionales y extensionales. Un identificador de objeto no se llama OOP como en GemStone, sino OID. Adicionalmente, este modelo incorpora nuevas características del modelo orientado a objetos:

- Objectos compuestos (estructurados recursivamente desde otros objetos).
- Los valores de los atributos son apuntadores a referencias de objetos.
- Contempla la *referencia compuesta*, que captura la semántica *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 o desigualdad, 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.

Conclusión.

El documento fuente, base de este capítulo, incluye algunos otros productos de bases de datos orientadas a objetos. Sin embargo, no se han incluido todos por contemplar características no estándares. Los productos aquí mencionados tienen más tiempo de desarrollo y muchos de ellos comienzan a adaptarse a los estándares ODMG-93 y ANSI-SQL3 (a estudiar en el próximo capítulo).

Bibliografia

[Larsen 92] A test evaluation procedure for object oriented and relational database management systems, Master's Thesis, A.B. Larsen, Institute *of Informatics, University of Oslo, Norway, 5 February 1992.*

[Kemper Moerkotte 94] Object-Oriented Database Management (Applications in Engineering and Computer Science), Alfons Kemper, Guido Moerkotte, *1984, Prentice Hall.*

[Ullman 88] Database and knowledge - base systems (Volume I: Classical Database Systems), Jeffrey D. Ullman, *Computer Science Press.*

[Chorafas Steinmann 93] Object - Oriented Databases, Dimitris N. Chorafas / Heinrich Steinmann, *Prentice Hall, 1993.*

[Date 95] An introduction to Database Systems, C.J. Date, *Addison Wesley, 1995, 6th. edition.*

[GemStone 96] Introduction to GemStone (Chapter 2), GemStone Systems, Inc. *14 - Dec - 95, http://www.gemstone.com/Products/TechOver/chapter2.htm*

[Ontos 95] ONTOS OIS, Ontos, Inc., *1995 http://www.ontos.com/ontos-ois.html*

[Versant 95] Versant - Development Solutions and Architecture, Versant, *http://www.versnat.com/versant/products/datasheet*

[Kim 94] UniSQL/X Unified Relational and Object-Oriented Database System, Won Kim, *SIGMOD Conference 1994*

[O2 95] Object Query Language manual, O2 Technology, *1995.*

[Frank 95] Object-Relational Hybrids, Maurice Frank, DBMS, *July 1995, Vol. 8, No. 8, pp. 46 - 56*

[Calderón 95] Una visión general de los sitemas manejadores de bases de datos orientadas a objetos, Marcos Calderón Macías, Soluciones *Avanzadas, Junio 1995, No. 22, pp. 37 - 43*

[Stonebraker 96] Object-relational DBMS, The next great wave, Michael Stonebraker with Dorothy Moore *Morgan Kauffman Publishers, Inc 1996.*

[Informix 96] Informix and Illustra Merge to Create Universal Server, Informix web site: `http ://www.informix.com/informix/whitpprs/illustra/ifxilus.htm`, 1996.

[Zand Collins Caviness 95] A Survey of Current Object-Oriented Databases, Mansour Zand, Val Collins, Dale Caviness ; *Computer Science Department, University of Nebraska at Omaha ; DataBase advances, February 1995 (Vol. 26, No. 1)*

V.- 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 como:

- OMG (Object Management Group)
- ODMG (Object Database Management Group)
- EXPRESS (Lenguaje)
- Open Distributed Processing
- Management Information Model
- SQL 3 (X3H2 ANSI)
- Matisse
- + (Lenguaje) (X3J16 ANSI)
- OO COBOL (Lenguaje) (X3 J4 ANSI)
- SmallTalk (Lenguaje) (X3 J20 ANSI)
- Eiffel (Lenguaje)
- System Object Model
- OLE 2 Component Object Model
- OSQL (HP Open ODB)

... entre otros.

En este capítulo, de los modelos listados, se analizarán aquellos cuya aplicación sean las bases de datos (ya que algunos otros tienen aplicación para lenguaje nada más, para proceso distribuido, etc.). Concretamente el modelo del grupo ODMG y el modelo de ANSI

para el nuevo lenguaje SQL 3. Se muestra también un resumen de las características del modelo OMG porque son la base tomada por el modelo ODMG. El modelo OSQL de HP no ha sido elaborado en términos de los conceptos manejados por el comité X3H7, así que sólo se cuenta con las características de este modelo especificadas en capítulos anteriores.

El reporte elaborado por el comité se denomina *Matriz de características del modelo de objetos*, porque es presentado precisamente como una matriz teniendo como renglones los diferentes modelos propuestos y como columnas los diferentes aspectos analizados, a saber:

- 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, por ejemplo).
- Polimorfismo
- Encapsulamiento (qué tan oculta se maneja la información en cada modelo).
- Identidad, igualdad y copia de objetos.
- Cómo se definen los tipos y las clases.
- Herencia
- Aspectos de los objetos (asociaciones, atributos, literales, contención, agregados, etc.)
- Extensibilidad
- Lenguajes definidos en el modelo (Si contempla DDL y DML y si son computacionalmente completos).
- Semántica de las clases base.

OMG

La idea básica del modelo de objetos OMG es el objeto, sus operaciones, los tipos de objetos con la capacidad de subtipificar. 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 del modelo (Core Model) comunes a todos los perfiles.

Los objetos se crean como instancias de tipos. Un tipo caracteriza 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 una *signatura*[27], 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. Formalmente, la *signatura* de una operación se denota por:

operación: $(par_1: tipo_1, par_2: tipo_2, ..., par_n: tipo_n) \rightarrow (ret_1: tipo_1, ret_2: tipo_2, ..., ret_m: tipo_m)$

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.

[27] El término no es muy común ; pero existe en la jerga formal computacional.

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 raíz 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[28]. Puede haber múltiples implementaciones para un tipo dado, dependiendo del dominio del problema.

ODMG

El grupo ODMG se inició a finales del 91 para atacar la carencia de un estándar en bases de datos de 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) y definieron el modelo que a continuación se detalla:

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

Por lo anterior, el primer documento ODMG-93 de especificaciones, 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 objetos.

[28] No es lo mismo clase que tipo estrictamente. La clase es la definición en una implementación de las características del tipo. Este concepto es paralelo en el modelo ODMG.

Modelo de datos

El modelo de objetos 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 permitie compartir datos entre lenguajes de programación. Los integrantes del grupo esperan que se convierta en un estándar *de facto*[29] a través del tiempo, y han acoplado sus productos y el lenguaje OQL para cumplir tal fin.

ODMG-93 difiere del trabajo actual ANSI SQL3 (a revisar en la página 79) en varios aspectos importantes. La arquitectura básica de ODMG-93 contempla extender los lenguajes de programación orientados a objetos existentes con persistencia y funcionalidad de bases de datos. En SQL el programador incorpora sentencias para convertir las estructuras de datos del lenguaje anfitrión a estructuras de bases de datos y viceversa. ODMG no extiende el SQL (como en el SQL3) para crear otro lenguaje de programación. Más bien combina el modelo OMG con la sintaxis de SQL. *En SQL3 debe haber colecciones de renglones, mientras que en ODMG las colecciones pueden ser de cualquier tipo y con cualquier nivel de composición.*

No obstante, los miembros de ODMG esperan poder trabajar en conjunción con el comité que aún está definiendo el SQL3 para llegar a un acuerdo y tratar de mapear lo más posible ambos modelos. Por el momento, el modelo ODMG-93 soporta SQL2 en sus productos comerciales. Sin embargo, dadas las necesidades mercadotécnicas, necesitaban tener definido un modelo previo (aunque ciertamente carente de algunas características como las mencionadas en párrafos anteriores); aunque están trabajando en refinarlo y esperando a que ANSI termine su definición (esperada para 1997) de manera que puedan unificarse.

El grupo ODMG es a la vez un miembro del grupo OMG; por lo que su modelo está basado en el núcleo del modelo general de objetos OMG, adicionando servicios de

[29] 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.

persistencia (incluso adoptado por OMG); sin embargo, los servicios de objetos del modelo OMG van más allá de los definidos en ODMG. Esta arquitectura proporciona más flexibilidad que el modelo ODMG; pero así mismo resulta más complejo. No obstante, ODMG contempla interfaces con las características no contempladas del modelo OMG similares a las definidas para lenguajes como C++ y Smalltalk.

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.

Literales: Son objetos inmutables, tanto atómicos como estructurados (enteros, booleano, char, etc.). Las literales estructuradas tienen dos tipos: colección inmutable (cadenas de bits, de caracteres, enumeraciones) y estructuras inmutables (fecha, hora, etc.). Se pueden definir subtipos adicionales; pero no se pueden definir las operaciones sobre tipos literales primitivas.

Objetos: Los objetos son instancias de un tipo, y como tales tienen 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. Todos los objetos son de tipo Objeto_Denotable. Un objeto puede ser mutable (instancia del tipo "objeto") o inmutable (instancia del tipo "literal").

Los objetos y literales pueden ser atómicos o estructurados. La identidad de objetos se representa a través de OIDs (object identificators); las literales se identificarán por su valor. Los objetos pueden también tener nombres significativos además de identidad (un objeto tendrá además las propiedades "tiene_nombre?" y "nombre").

Operaciones: Las operaciones se definen en los tipos. La interfaz de un tipo incluye *signaturas de operaciones*: esto es, nombres de argumentos y tipos, excepciones posibles, tipos de resultados. El primer argumento de una operación determina el proceso a ejecutar.

Los nombres de las operaciones pueden sobrecargarse; la ejecución de una operación se basa en el tipo más específico del primer argumento de la llamada a la operación. Las operaciones pueden tener efectos colaterales; una operación con efectos colaterales puede regresar valores nulos. En el modelo se asume que las operaciones se

ejecutarán en forma secuencial. Los optimadores, por tanto, deben ser conservadores puesto que una operación en una consulta puede tener efectos colaterales.

Especificación de la semántica del comportamiento: El comportamiento de un objeto se modela a través de las operaciones. Una *signatura* de una operación (parte de un tipo de especificación) especifica un tipo de argumento, un tipo de resultado y las excepciones que pueden dispararse.

Métodos: Las operaciones (declaradas en la interfaz) se implementan con métodos definidos en la implementación del tipo. Cada operación se implementa por un métodos, además de existir métodos adicionales definidos.

Estado: El estado se modela a través de las propiedades de un objeto. Una propiedad puede ser un atributo o una asociación. Los atributos y asociaciones de un objeto se definen como parte de la interfaz del tipo. Los atributos toman literales como valores; las asociaciones sólo pueden definirse entre dos tipos de objetos no literales.

Tiempo de vida de un objeto: El tiempo de vida de un objeto es ortogonal a su tipo, se especifica a la creación del objeto y, una vez especificado, no puede cambiarse. El tiempo de vida puede terminar al mismo tiempo que el procedimiento o manejado por la base de datos en tiempo de ejecución. El tiempo de vida no es aplicable a las literales (objetos inmutables). Las literales siempre existen implícitamente. La mayoría de las consultas regresan literales. El lenguaje de consulta contiene expresiones para construir objetos, pero no existe tiempo de vida de estos objetos.

Comportamiento: ODMG especifica un modelo de objetos clásico: cada operación tiene un argumento que la distingue.

Modelo de comunicaciones: El modelo asume que las operaciones se ejecutan en forma secuencial; pero se permiten operaciones paralelas concurrentes.

El modelo soporta transacciones anidadas Una transacción obtiene bloqueos de lectura-escritura estándares con control pesimista de concurrencia.

Ligado: El modelo soporta ligado en tiempo de ejecución de métodos a los objetos basados en el primer argumento.

Encapsulamiento: Todos los objetos son instancias de un tipo que especifica la interfaz para accesar al objeto. Existe una sola interfaz por cada tipo.

Identidad, igualdad y copia: Todos los objetos denotables tienen una identidad. Para las literales esta identidad es su valor. Para los objetos, la identidad es un valor que distingue en forma única a un objeto de otros objetos del mismo dominio donde se creó, y es independiente del estado del objeto. Recuérdese que son operaciones diferentes la asignación de un objeto O_1 a otro O_2, y la copia :

O_1 := O_2 hace que O_1 apunte al mismo objeto que O_2

O_1 := copy_of(O_2) crea un nuevo objeto y copia todos los valores de estado de O_1 sobre O_2 (salvo las propiedades de identidad).

Tipos y clases: Un tipo es una especificación; puede tener una o más implementaciones. Todos los tipos son instancias del tipo TIPO. Una clase es la combinación de la especificación de un tipo y una implementación específica. El modelo está fuertemente tipificado.

El conjunto de todas las instancias de un tipo es la 'extensión' del tipo. Los tipos abstractos no son instanciables - sólo tienen especificadas las características que deben heredarse pero no definen implementación alguna.

Dos objetos son compatibles si son instancias del mismo tipo declarado o si alguno es una instancia de un subtipo de otro. Dos estructuras literales tienen el mismo tipo si tienen la misma estructura a cualquier nivel y los tipos atómicos correspondientes son los mismos. El heredar subtipos para las literales requiere la misma estructura para cada nivel y que el tipo de cada subobjeto sea el mismo, o que sea un subtipo de supertipo del subobjeto correspondiente. No existen conversiones implícitas para objetos o literales estructuradas; sólo se dan en la especificación del lenguaje de consulta.

Herencia: El modelo define herencia de tipos. Si S es un subtipo de T, entonces S hereda todas las operaciones y propiedades de T, y S puede definir nuevas operaciones y propiedades aplicables a sus instancias. Un subtipo puede especializar las propiedades y operaciones que hereda; pero el modelo no proporciona reglas para indicar cómo llevar a cabo esta especialización. Existe herencia múltiple teniendo que renombrar las operaciones o propiedades que tuviesen el mismo nombre en los supertipos.

Dos objetos tienen el mismo tipo *si y sólo si* son instancias del mismo tipo con nombre. Un objeto de un tipo dado puede asignarse a un objeto de alguno de sus supertipos.

La compatibilidad en las literales se da por su estructura (colecciones inmutables o estructuras). Se dice que tienen el mismo tipo sí y sólo si tienen la misma estructura a cada nivel y las partes atómicas tienen el mismo tipo.

Asociaciones: Las asociaciones son un tipo de propiedad definida entre dos tipos de objeto mutables. Las asociaciones NO son objetos. Una asociación puede ser uno a uno, uno a muchos o muchos a muchos. Las asociaciones están definidas in la interfaz del tipo de objeto. Una asociación que deba referirse en ambos objetos tendrá un nombre en uno y el nombre equivalente de regreso en el otro.

Atributos: Los atributos son propiedades definidas sobre un tipo de objeto simple. Los atributos toman literales como valores. Los atributos pueden accesarse con operaciones get_value() y set_value() (que pueden redefinirse) que estén definidas como parte de la interfaz. Los atributos no pueden tener propiedades (son de último nivel).

Agregados: Los objetos estructurados (agregados) puede ser del tipo *estructura* o *colección*. Las *estructuras* son registros con propiedades nombradas que pueden llenarse con objetos o literales de diferentes tipos. Las *colecciones* son agrupaciones homogéneas de elementos con o sin orden. Los tipos de colección primitivos son:

- Conjunto: Colección de elementos *únicos* sin orden.
- Bolsa: Colección de elementos sin orden que permite *duplicados*.
- Lista: Colección de elementos ordenados que permite *duplicados*.
- Arreglo: Colección de tamaño variable que permite valores *nulos*.

Las colecciones mutables tienen semántica intensional. Las inmutables tienen semántica extensional[30]. Pueden definirse iteradores para recorrer los elementos de la colección. El tipo *colección* define también operaciones `select` basadas en predicados. Las operaciones de consulta se aplican a cualquier colección (extendidas, definidas por el usuario). El resultado de una selección es una subcolección del mismo tipo que la colección consultada. Una colección debe tener definidas operaciones de actualización (insertar, reemplazar, suprimir). Una colección de literales no podrá tener este tipo de operaciones.

Un agregado no es un BLOB[31] (Bynary Large OBject) en el modelo. Un BLOB

[30] Definida en el capítulo del modelo relacional
[31] Este es un tipo de datos complejo : audio, video, imagen, etc.

puede almacenarse en una base de datos relacional; pero ésta desconoce la estructura interna del dato. Una aplicación deberá entonces tener codificado el acceso a dicho tipo de dato. Un tipo agregado especificado bajo los lineamientos de este modelo exime al desarrollador que lo reutilice, de recodificar la interfaz.

Debe definirse un tipo *excepción* que puedan subtipificarse para el manejo interno de las excepciones en transacciones.

El tipo *Tipo* es un subtipo y una instancia del tipo objeto_atómico. Los meta-datos pueden accesarse usando la interfaz para instancias de tipos, y pueden consultarse utilizan el lenguaje de consulta estándar.

Lenguajes: ODMG especifica un lenguaje de definición de objetos (ODL) que soporta a su modelo de objetos. Este lenguaje es compatible con el lenguaje IDL de OMG. Debe ser independiente de C++ o SmallTalk.

También se especifica un lenguaje de consultas (OQL) similar al SQL que da acceso declarativo a los objetos. Cualquier objeto denotable puede consultarse, especificando primero el nombre del objeto:

- Consultar elementos del conjunto pre-nombrado *Personas*

```
SELECT DISTINXT C.EDAD
  FROM X IN Personas
 WHERE x.nombre = "Pedro"
```

- Consultar sobre el resultado de otra consulta (consultas anidadas)

```
SELECT DISTINCT struct( a: x.edad, s: x.sexo)
  FROM x IN
      ( SELECT y
          FROM y IN Empleados
        WHERE y.salario > 10000
          AND y.nombre = "Pedro" )
```

ANSI - SQL3

Los comités ANSI X3H2 e ISO DBL actualmente (y en los últimos tres 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 :TDAs (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 TDAs pueden implementarse utilizando las extensiones procedurales de SQL3 o con lenguajes externos; así mismo pueden contener operadores unarios o binarios sobre sus instancias.

Los TDAs pueden ser tanto objetos con 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 TDAs parametrizados. Existen tipos de colecciones predefinidas como CONJUNTOS, MULTICONJUNTOS (bolsas en ODMG-93) 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 TDAs 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 TDAs.

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 proporcionaran compatibilidad tanto para el *tipo* como la *tabla*. Los TDAs pueden utilizarse como los tipos

preconstruidos; por ejemplo: las columnas de las tablas relacionales pueden definirse sobre dominios de tipos definidos por el usuario. Las operaciones que definen la igualdad y ordenación de los TDAs 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 no todavía por ISO) que permitirá mantener un identificador único y ser utilizado por las aplicaciones del DBMS; pudiendo utilizarse como valora para una columna o como llave foránea. Esta capacidad reducirá la *diferencia* entre los renglones SQL y los 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 propiedades en la misma posición y del mismo tipo[32]. 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*[33] y columnas en base a los *tipos renglón* de cada tabla.

Cláusulas adicionales

WITH IDENTITY: Un identificador de renglón es un tipo de dato utilizado para identificar renglones en tablas base[34]. El valor de este identificador es igual al número de renglón y por tanto es diferente en cada renglón de la tabla. Cada tabla con esta definición tendrá una columna implícita llamada `IDENTITY`, que no se incluirá en los resultados de la consulta `SELECT *`, aunque puede pedirse explícitamente.

UNDER: Se ha incorporado también el concepto de subtabla. Una tabla puede declararse como subtabla de una o más supertablas. De esta manera, una subtabla hereda todas las columnas de sus supertablas además de poder definir nuevas columnas. Este concepto es muy independiente del concepto de subtipo TDA.

---

[32] Equivalente a la *compatibilidad en la unión* del modelo relacional

[33] 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

[34] Tablas físicamente almacenadas en dispositivo.

TEMPLATE: Se utiliza esta cláusula para permitir la definición de familias de TDA con tipos parametrizados. Un parámetro de una plantilla de tipos puede ser cualquier valor conocido en el ambiente de SQL. Un tipo generado es un TDA que resulte de la especificación de un parámetro formal en un tipo plantilla. Se permiten los tipos anidados, ejemplo: secuencia(punto(FLOAT))

Objetos: Como se mencionó en párrafos anteriores, SQL3 permitirá la definición de TDAs (tipo de dato abstracto) utilizando tipos preconstruidos o reutilizando otros TDAs definidos por el usuario; y las columnas de las tablas relacionales pueden tomar sus valores de los dominios correspondientes a estos TDAs.

Existen dos tipos de TDAs, TDAs objetos y TDAs valores[35]. 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 OIDs 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 TDAs como partes de otros TDAs o como columnas de tablas; teniendo así mismo la capacidad de incorporarlos completamente o sólo referirse a ellos a través de sus OIDs. Para ello se incorpora la cláusula INSTANCE en la definición de atributos de un TDA. Esta facilidad puede originar alguna inconsistencia semántica si se usa irresponsablemente; por lo que aún está evaluándose por el comité. Por ejemplo, si se define un TDA persona dentro del cual exista un atributo del mismo TDA persona (digamos, cónyuge; o jefe inmediato); 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_i que incluye un atributo Y también de este tipo t_i; si se modifica directamente una atributo z (que ambos tienen) a través del primer dato:

X . Y. z := algo;

en lugar de hacerlo directamente:

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

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 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 TDAs 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, parámetros 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 actualizar 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 *signatura*. Se invocan utilizando una notación funcional, que también se utiliza para invocar rutinas asociadas a tablas.

Especificación de la semántica del comportamiento: Las rutinas pueden

encapsularse en la definición de los TDAs, teniendo acceso a sus atributos privados; o pueden definirse fuera de los TDAs. Existen algunas rutinas predefinidas para los TDAs, como las constructoras que generan una instancia de dichos TDAs.

Métodos (multimétodos y combinación de 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 el comportamiento de objetos totalmente.

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 TDAs 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 (como en el caso del modelo ODMG donde cualquier dato o colección de datos puede tener asociado un 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 TDAs sobre los que se desee hacer consultas.

Modelo de comunicaciones: Hasta el momento no existe

Eventos: Hasta el momento no se ha especificado el manejo de de excepciones y *triggers*[36].

Polimorfismo: SQL 3 soporta la definición de rutinas con el mismo nombre en diferentes TDAs, además de la redefinición de rutinas heredadas en un subtipo. Esto es cierto tanto para los TDAs 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: SQL 3 maneja tres tipos de encapsulamiento:

- PÚBLICO: Componentes que constituyen la interfaz del TDA y son visibles para todos

[36] Situación o condición que dispara un proceso eventual

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.

Identidad, igualdad y copia:

- Dos instancias de un TDAs con el mismo OID son el mismo objeto (identidad).
- Dos instancias de un TDA con diferente OID pero con el mismo valor en todos sus atributos son dos objetos iguales (igualdad).
- Dos valores atómicos no son distintos si son nulos.
- Dos renglones del mismo tipo son distintos si el valor en al menos uno de sus atributos es distinto.

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 pre definidos (char, int, float, etc.); así como colecciones predefinidas. Ahí mismo incorpora tres tipos de tabla: SET, MULTISET y LIST (explicadas en párrafos anteriores) . A continuación un ejemplo de la declaración de un TDA:

```
CREATE OBJECT TYPE persona
   (nombre VARCHAR NOT NULL,
    sexo CONSTANT CHAR(1),
    edad UPDATABLE VIRTUAL GET WITH edad SET WITH fija_edad,
   PRIVATE
    cumpleanios  DATE CHECK (cumpleanios <> DATE '1992-01-01'),
   PUBLIC
    EQUALS DEFAULT,
    LESS THAN NONE,

    ACTOR FUNCTION edad (:P persona) RETURNS REAL
       RETURN <... código para calcular la edad ... >
    END FUNCTION,
    ACTOR FUNCTION fija_edad (:P persona, ... ) RETURNS persona
       < ... código para actualizar la fecha ... >
       RETURN :P
    END FUNCTION,

    DESTRUCTOR FUNCTION remover_persona (:P persona)
       RETURNS persona
       < ... operaciones previas para liberar memoria ... >
       DESTROY :P;
```

```
        RETURN :P;
    END FUNCTION;
);
```

Herencia: La herencia de TDAs es soportada con la cláusula UNDER. Existe un *tipo mínimo específico* que corresponde al más bajo nivel de subtipos asignado 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, delete, update) se definen de manera que mantengan los renglones de las tablas en forma consistente entre subtablas y supertablas, a saber:

- Al insertar un renglón en una subtabla T, se inserta un renglón correspondiente (con el mismo identificador y los mismos valores de los atributos heredados) en cada supertabla de T.
- Al actualizar un renglón en una supertabla, se modifican todas las columnas de los renglones de las tablas heredadas.
- Al actualizar un renglón de una subtabla, cada renglón correspondiente de las supertablas es modificado.
- Al eliminar un renglón en una tabla que pertenece a una familia de subtablas, todos los renglones correspondientes (niveles superiores y subniveles) son eliminados también.

Asociaciones: En SQL 3 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 SQL 3 existen dos tipos de atributos para los TDAs:

- Almacenados: Atributos de cualquier tipo (incluso otros TDAs). Cada atributo declarado genera implícitamente atributos para visualizar y modificar (get, set) su 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 también representan atributos.

Agregados: SQL3 soporta tipos renglón como estructuras literales cuyas instancias pueden utilizarse como valores de renglones en tablas. Así mismo pueden especificarse colecciones SET(<tipo>), MULTISET(<tipo>) o LIST(<tipo>) de diferentes tipos. Cuando cada elemento de una colección corresponda a un renglón en una tabla, puede utilizarse la colección para consultas simples. Las tablas se consideran como una simple columna cuyo tipo es el tipo de instancias de esa colección[37].

Extensibilidad: Se pueden definir nuevas tablas y nuevos TDAs en base a 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 SQL 3 en un lenguaje computacionalmente completo. A saber:

- DESTROY: Elimina los objetos TDA existentes y debe estar en una función DESTRUCTOR.
- ASSIGNMENT: Permite asignar el resultado de una expresión SQL a un variable local, una columna o un 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

[37] Nuevamente encontramos el mapeo clase = dominio cuando un tipo de renglón se convierte en un dominio.

repetitiva hasta cumplir la condición especificada en la cláusula WHILE.

Pueden 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 SQL 3 no contempla actualmente ningún tipo de interfaz a lenguajes como C++ o SmallTalk; pero se está trabajando en este aspecto, aunque C++ guarda más similitud con el modelo de SQL 3 (por ejemplo la visibilidad de los atributos: publico, protegido y privado) que SmallTalk (este último requerirá de más trabajo para poder ser enlazable a SQL3).

Problemas 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. A saber:

- Algunos OODBs requieren que el usuario maneje explícitamente los bloqueos del esquema de 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 optimador de consultas.
- Se requieren algoritmos de procesamiento de consultas que evalúen la complejidad de éstas (joins, 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 optimador 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.

Adicional a todos estos problemas enunciados, [Shiner 96] define un problema particular del ODMG-93 que sus implementadores han dejado de lado. El estándar ODMG 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 creada 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 (lo que en el modelo relacional equivale al catálogo de la base de datos) con interfaces individuales para cada lenguaje (aquellos diferentes de C++ y SmallTalk), y para cada biblioteca implementada en una plataforma en particular (aquellos compatibles con los dos lenguajes enunciados).

Esto redunda en una problemática mayor ; porque la definición de un estándar formal (como el SQL-92 que actualmente se utiliza mucho en las grandes bases de datos basadas en el modelo relacional) se hace necesaria para enriquecer al modelo y hacerlo independiente de la plataforma.

Particularmente, mi propuesta de un modelo formal para definir el esquema de la base de datos consiste en enunciar los atributos o características del modelo a través de una secuencia de estos atributos que resida en formato de texto plano (plain text) 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 (donde este dominio puede ser alguna clase definida en el mismo archivo plano, por ende se podrá resolver en el mismo archivo ; o bien algún tipo de dato primitivo de los que más comúnmente soportan los diferentes lenguajes hoy en día : enteros, reales, booleanos, caracteres, cadenas de caracteres, fechas, horas, etc.).

Esta idea resulta del desarrollo personal de un prototipo para bases de datos

deductivas cuyo lenguaje base fue el LISP (LISt Processing). En este prototipo los esquemas de bases de datos estaban declarados como listas reconocidas por el lenguaje :

```
ALUMNO( MATRICULA, NOMBRE, FECHA_INGRESO )
MATERIA( CLAVE, DESCRIPCION, PROMEDIO_MINIMO )
```

Con algunas modificaciones (utilizar llaves en lugar de paréntesis ; así como utilizar palabras claves que hagan referencia a símbolos típicos de la lógica : for_every, exists, greather_than, less_than, etc.) ; puede implementarse un protocolo de definición de bases de datos con notación formal de conjuntos que no ocupe demasiado espacio (esto es necesario por que el archivo plano regularmente es muy extenso ; pero al tener definido únicamente el esquema el archivo no crece tanto), y que sirva como canal de entrada para que cada implementador o cada fabricante de lenguajes construya su propio controlador (driver) capaz de reconocer el mismo archivo plano del esquema de la base de datos y haga las traducciones necesarias para definir sus propios objetos y clases dentro del lenguaje. Ejemplo de una pequeña base de datos en notación de conjuntos:

```
ALUMNO { MATRICULA :STRING(10),
         NOMBRE :STRING(30),
         FECHA_INGRESO : DATE,
         MATERIAS_CURSADAS [n:INT]: MATERIA,
         n LESS_THAN 5 }
        /* Un alumno puede cursar hasta 5 materias */
MATERIA { CLAVE :INT, DESCRIPCIÓN :STRING(30),
          PROMEDIO_MINIMO :INT,
          ALUMNOS_INSCRITOS[n :INT] :ALUMNO,
          n LESS_THAN 50 }
```

Obviamente esta notación es una idea no desarrollada completamente ; pero el fundamente de la misma es el hecho de que los conjuntos son la base formal tanto del modelo relacional como del orientado a objetos. Por lo anterior, resulta fácil percibir que esta notación representa un modelo semántico más claro. La forma de resolver los tipos de datos (dominios) según se encuentren definidos, ya es un aspecto de implementación y no de modelado.

Bibliografía

[Barry 95] On the road to standards, Douglas Barry, Object Magazine, *October 1995, SIGS publications*

[Loomis 93] Object database semantics, Mary E. S. Loomis, *Journal of Object Oriented Programming, Vol. 6, No. 4, July/August 1993, SIGS publications. pp. 26-33*

[Loomis 94] Querying object databases, Mary E. S. Loomis, *Journal of Object Oriented Programming, Vol. 7, No. 3, June 1994, SIGS publications. pp. 56-70, 78.*

[Cattel 94] A Standard for Object - Oriented DBMSs. R.G.G. Cattell, *SIGMOD Conference 1994. ACM Inc.*

[Kulkarni 94] Object - Oriented Extensions in SQL3: A Status Report Krishna G. Kulkarni, *SIGMOD Conference 1994, ACM Inc.*

[Jordan 95] OQL: The Object Query Language David Jordan, C++ Report, *SIGS Publications 1995*

[O_2 95] Object Query Language: OQL Manual O_2 Technologies, *Technical manual, March 1995.*

[X3H7 96] ANSI X3H7 Object Model Features Matrix, ANSI X3H7 comitee, *ANSI 1995-96 capitulo OMG, ODMG y SQL3*

[Shiner 96] What's missing from ODMG-93 ?, John F. Shiner, *Object Journal, SIGs publications, February 1996*

CONCLUSIONES

El modelo relacional existirá por mucho tiempo todavía. Sus objetivos principales originalmente contemplaban la independencia total entre la implementación y la conceptualización de la semántica de los datos. Ha cumplido ese cometido y por tanto difícilmente puede pensarse en su pronta o eventual desaparición.

No obstante, el modelo orientado a objetos otorga una serie de capacidades y beneficios (más allá de la simple definición de nuevos tipos de datos) que el modelo relacional no contempla. De esta manera, este último se enriquecerá considerablemente con las nuevas características que grupos y organizaciones de estándares han estado definiendo y que próximamente concluirán. Así, ambos modelos deberán convivir en los nuevos sistemas de administración de bases de datos que se están desarrollando. No desaparecerá ninguno de los dos, sino que se complementarán al constituir dos marcos de referencia diferentes (ortogonales) desde los cuales se ve la estructura conceptual de la información.

Así mismo, la constante discusión de la ecuación que mapea estos dos modelos tiene una respuesta final: No es cierto que, como [Date 95] lo sugiere, la única ecuación válida para el mapeo del modelo relacional al modelo orientado a objetos sea:

clase = *dominio*

Sino que *existe* una nueva ecuación adicional que es:

relación = *dominio (compuesto).*

De esta manera, la única alteración sobre los postulados originales del modelo relacional es que los dominios ya no son *nada más* valores atómicos o escalares, sino que pueden estar constituidos por elementos a cualquier nivel de descomposición. Así mismo, no es la clase la que se iguala al dominio; dado que ya quedó establecido que la clase es un modelo, un esquema, y no un conjunto de datos (pese a que en C++ así se considera ; pero este lenguaje ni siquiera es formal). La clase más bien se puede mapear al esquema de una relación o a un atributo. En [Camps 96] (publicado apenas en septiembre de 1996) se refuerza esta afirmación, haciendo hincapié en que C.J. Date reconoce que la investigación constante hace cambiar de opinión y evoluciona las ideas.

Esta última ecuación permite también establecer una base formal para el modelo orientado a objetos basado en conjuntos y cuyas operaciones son equivalentes al cálculo relacional de dominios del modelo relacional.

Son diferentes las ventajas que proporciona que ambos modelos interactúen, entre otras:

- Introducción del concepto de tipo de dato abstracto y tipos definidos por el usuario (atómicos y no atómicos), lo 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.

  Por ejemplo: Sea una tabla de derechohabientes del seguro social donde se registra a los cónyuges de cada persona: El modelo relacional requerirá la especificación de una llave foránea que haga referencia recursivamente a la misma tabla. El modelo orientado a objetos requerirá la declaración de un atributo cónyuge cuyo dominio es el conjunto de instancias de la misma clase asegurado. Consultar los cónyuges de la gente no soltera nos da las siguientes dos sentencias en cada uno de los modelos:

Relacional:

```
SELECT   t_conyuge.nombre
  FROM der_hab dh, der_hab t_conyuge
 WHERE dh.conyuge = t_conyuge.llave_asegurado
   AND dh.conyuge NOT NULL
```

Objetos:

```
SELECT der_hab.conyuge.nombre
  FROM der_hab
 WHERE dh.conyuge NOT NULL
```

En el primer caso se necesitó recurrir a los alias dado que está definida una operación *join* de una tabla consigo misma.

En estas sentencias notaremos tres ventajas concretas del modelo orientado a

objetos:

- ◊ Semántica más clara en el modelado de los datos
- ◊ Por ende, 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: léase join).
- ◊ Mejor desempeño al no ejecutar una operación join, sino un acceso directo al objeto al que apunta el tipo de dato *cónyuge* definido en el dominio *derecho_habiente*.

- 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 (cálculo relacional de tuplas y cálculo relacional de dominios) 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 (no sólo operaciones entre 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[38], ahorrando el trabajo de definir tablas con todos los atributos heredados. La ventaja real está dada en la *conceptualización* o *modelado* de elementos que heredan atributos, entre los cuales no sólo existen valores escalares, sino operaciones (comportamiento) que pudieran ser muy complejos y que el heredarlos facilitará el trabajo conceptual (no tanto el de implementación).

En resumen, ambos modelos proporcionan una serie de ventajas que se complementan entre sí. Por un lado continuará utilizándose el álgebra (pero ahora conceptualizada como álgebra de dominios); y por otro lado se aprovechan las características de tipos de datos abstractos, herencia, comportamiento, y modelos semánticos más ricos.

[38] Finalmente existen herramientas para el modelo relacional que contemplan este concepto en el modelo conceptual y generan un modelo físico