Paradigmas de Modelado de Base de Datos I. CI5311. Apuntes de clase

Prof. Soraya Abad Mota

Actualizaciones: Octubre 2005, Mayo 2008, Abril 2009, Mayo 2012

1. Introducción

Este documento es una guía de los tópicos de la materia CI-5311. No existe un libro de texto que cubra todos los aspectos de esta materia; el material de apoyo consiste de un conjunto de artículos y capítulos de libros donde se abordan los diferentes temas y con estos apuntes se trata de enlazar todos los tópicos. Comenzamos con los objetivos del curso y su contenido, lo cual nos ocupa en el resto de esta sección.

El documento contiene una sección por cada tema; las secciones son más o menos escuetas, dependiendo del material que se encuentre disponible para cada uno. Por ejemplo, el tema de la tecnología objeto-relacional es muy amplio y el material de apoyo está disperso en muchas fuentes; de modo que para tratar de dar coherencia al tema y guiar al estudiante por todo el material, se cubre en mucho detalle en estos apuntes. Sin embargo, el tema del modelo entidad-interrelación extendido, que se cubre a modo de repaso y para contrastarlo con otros modelos conceptuales, apenas se trata como una lista de conceptos; los cuales se suponen conocidos y se encuentran en una sola fuente, que es el texto utilizado en la materia de introducción a los sistemas de base de datos.

1.1. Objetivos del curso

Este curso contempla los objetivos que se enumeran a continuación.

1. Descubrir y aplicar diversos modelos y métodos para modelar conceptualmente bases de datos.
2. Adquirir una visión global de la tecnología asociada a los modelos cubiertos.
3. Adquirir una "actitud" de modelado conceptual. Para ello es vital destacar la importancia y el impacto de esta fase en el éxito de un sistema de base de datos.

Los temas contenidos en el curso son los siguientes.

1. Introducción. Modelos de datos. Paradigmas. Proceso de Diseño de una base de datos. Diseño conceptual. Recolección y definición de los requerimientos de datos.
2. Modelo Entidad-Interrelación (ER) y la extensón de Elmasri/Navathe.
3. Object Modeling Technique (OMT). Modelo de objetos (estático). Notación UML.
4. Estrategia general de Diseño de una Base de Datos.
5. Calidad del esquema conceptual. Criterios y cómo lograrla.
6. Tecnología Objeto-Relacional. Historia de los DBMS. OODBMS vs ORDBMS. Definiciones, SQL-3, traducción de esquemas de objetos UML.
7. OMT: Modelo Dinámico. Conceptos básicos, diagrama de estados.
8. Bases de Datos Activas. Paradigma, triggers.
9. Bases de Datos Deductivas. Paradigma, historia. Otros Modelos.

2. Modelo de datos y proceso de diseño

El aspecto central de este curso son los modelos de datos, de modo que es muy pertinente comenzar por las definiciones de modelo y modelo de datos y en general un poco de terminología.

Modelo
(definiciones de diccionario)

1. Representación en pequeño de alguna cosa.
2. Esquema teórico, generalmente en forma matemática, de un sistema o una realidad compleja que se elabora para facilitar su comprensión y el estudio de su comportamiento.

Modelo de Datos
Colección de conceptos que pueden utilizarse para describir la estructura de una base de datos. Por estructura entendemos: los datos, las interrelaciones entre ellos y las restricciones que éstos deben cumplir. Algunos modelos incluyen también una serie de operaciones que actúan sobre los conceptos del modelo y permiten describir acciones sobre los datos.

Arquitectura de los tres niveles de un Sistema de Base de Datos:

I Modelo Externo (visiones de usuario)

II Modelo "Lógico"

III Modelo Interno (nivel físico)

El modelo externo debe ser independiente del DBMS a utilizar, el interno si depende del manejador y el lógico depende del modelo de datos utilizado. Esta arquitectura no sólo describe al sistema de base de datos una vez implantado sino que también se utiliza para diferenciar las fases del diseño de una base de datos, nosotros le hemos agregado un nivel a la arquitectura, se trata del *nivel conceptual*, es la visión de la base de datos al más alto nivel de abstracción. Si el modelo de datos utilizado para diseñar conceptualmente la base de datos tiene una implementación, es decir, tiene un DBMS basado en su modelo, entonces el nivel conceptual y el nivel lógico coinciden y se puede diseñar e implementar la base de datos utilizando el mismo modelo. El modelo relacional es un modelo de datos a nivel lógico pues existen implementaciones de DBMSs del mismo, pero el modelo ER extendido en un modelo a nivel conceptual y no existen implementaciones de DBMSs basadas en él.

El diseño conceptual de una base de datos es un proceso iterativo de refinamientos sucesivos.

En este curso nos concentramos en dos modelos de datos a nivel conceptual, a saber: el modelo Entidad-Interrelación Extendido (ER-E) y el Modelo de Objetos de OMT (Object Modeling Technique). Consulte las fuentes indicadas en la bibliografía del curso y en las lecturas semanales sugeridas para estudiar estos modelos. En las dos secciones siguientes sintetizamos los conceptos fundamentales de cada uno de estos modelos.

3. Modelo Entidad-InterRelación Extendido (ER-E).

- Clase. ENTIDAD, superclase, subclase, entidad débil.
- Atributos: simples o compuestos, un sólo valor o multivaluados, primitivos o derivados, opcionales o fijos.
- INTERRELACIÓN. Tipos: 1:1, 1:n, n:m. Restricciones de cardinalidad, notación (min,max).
- Interrelaciones especiales. Subclase (is-a). Generalización/Especialización. Categorización.
- Notación gráfica. Diagramas conceptuales.

El modelo ERE ofrece algunas heurísticas de diseño, para decidir entre las siguientes alternativas:

- ¿Entidad o Atributo?
  En el ejemplo de la empresa de servicios eléctricos, el titular de pago puede representarse como un atributo del contrato de servicios o como una entidad asociada al contrato.
- ¿Generalización o Atributo?
  Si en el ejemplo anterior se decide representar al titular de pago como una entidad, entonces para indicar que los titulares de pago pueden ser una persona natural o una persona jurídica, se puede utilizar una generalización o simplemente colocar un atributo que indique el tipo de titular de pago en esa entidad.
- ¿Atributo compuesto o Conjunto de atributos simples?
- ¿Entidad o Interrelación?
  Si el concepto a representar es un concepto muy importante en el esquema y va a estar relacionado con otros conceptos, es preferible representarlo como una entidad. La noción de pago, por ejemplo, puede ser la interrelación entre un cliente y un producto o puede ser una entidad que se asocia con el cliente, pues es quien paga y se asocia con el producto pues es lo que se está pagando.

4. OMT. Modelo de Objetos.

- OBJETOS. Tienen *identidad*. Contienen datos y operaciones, proveen abstraccoón y encapsulamiento.
- CLASES. Son conjuntos de objetos similares en cuanto a datos y operaciones. Con la definición de una clase se definen atributos que describen a los objetos contenidos en la clase, y se definen operaciones que se ejecutan sobre los objetos o que los objetos ejecutan. Las clases permiten definir *herencia* y *operaciones polimórficas*.
- Los objetos son instancias de una clase.
- ENLACES. Relacionan a dos o más objetos.
- ASOCIACION. Conjunto de enlaces entre los objetos de dos clases, no necesariamente distintas.
- Los enlaces con las instancias de las asociaciones.
- Las asociaciones pueden tener atributos definidos y tienen restricciones de multiplicidad (cardinalidad), también se les puede asignar nombres de rol a las clases que participan en la asociación.
- En OMT se puede modelar una *asociación como una clase* y se tiene el concepto de *asociación calificada*.
- Existen dos restricciones implícitas del modelo que se pueden especificar para las asociaciones, estas son: el ordenamiento (*ordered*) de las instancias de una asociación y la noción de subconjunto (*subset*) aplicada a dos asociaciones diferentes.
- Existen dos asociaciones especiales: Generalización y Agregación.
- Hay otras nociones especiales importantes en OMT: descriptores de clase, clases abstractas y metadatos.
- El modelo de objetos tiene una notación gráfica.
- El modelo de objetos de OMT permite modelar aspectos estáticos, estructurales y de datos.

Los creadores de la metodología OMT también dan algunas sugerencias para construir el modelo de datos. Estas son:

1. No se deben escribir clases, asociaciones y herencia "a lo loco". Primero hay que entender el problema y el contenido del modelo de objetos se obtiene de acuerdo a la relevancia en la solución al problema.
2. Haga un modelo simple, evite complicaciones innecesarias.
3. Elija nombres apropiados con mucho cuidado. Los nombres tienen una poderosa carga de connotaciones implícitas para cada quien que los lee. (Este es uno de los aspectos más difíciles de lograr.)
4. No esconda las referencias a otros objetos en apuntadores. Modele estas referencias con asociaciones.
5. No trate de colocar las cardinalidades perfectas muy temprano.
6. No coloque los atributos de la asociación en una de las clases.
7. Use asociaciones calificadas cuando sea posible.
8. Construir un modelo de objetos correcto, requiere de revisiones y de varias iteraciones, nunca sale bien a la primera.
9. Trate de que otros revisen su modelo.
10. Documente siempre su modelo de objetos, no basta con el diagrama. Hace falta una guía del modelo y la explicación de porqué se tomaron ciertas decisiones. Estas explicaciones clarifican el modelo.
11. No sienta que debe usar todas las primitivas de modelación de objetos. No todos los problemas las necesitan todas. Use sólo lo que necesite y haga a su modelo expresivo y autoexplicativo (ver criterios de calidad más abajo).

Una vez que se haya producido un esquema conceptual refinado, que cubra todos los requerimientos de datos e información del universo de discurso, para efectos de documentación es muy útil producir una *lectura del diagrama*, esta es un recorrido narrativo por el esquema conceptual, donde se explican las entidades y las interrelaciones, pero no se dan detalles de los atributos. Generalmente es preferible hacer un recorrido top-down del esquema, se comienza por el concepto central se recorren generalizaciones y subconjuntos, se identifican grupos (clusters) de información dentro del esquema y se describe cada uno, y finalmente se describen las propiedades individuales relevantes de cada entidad y cada interrelación.

5. Criterios de Calidad de un Esquema Conceptual

Se definen varios criterios para evaluar la calidad de un esquema conceptual. Se dan transformaciones para mejorar la calidad. Lo ideal es que el esquema conceptual sea mínimo, expresivo y auto explicativo. Ahora veremos los criterios y las transformaciones en detalle.

Los criterios de calidad de un esquema conceptual son los siguientes:

Completitud. Este criterio tiene dos aspectos duales: cada requerimiento debe estar en el esquema conceptual y cada concepto del esquema está en los requerimientos.

Correctitud. Un esquema es correcto si usa apropiadamente el modelo de datos. Hay dos tipos de correctitud: sintáctica y semántica.

- Sintáctica: por ejemplo, no poner una entidad con otra sin que haya una interrelación entre ellas o poner juntas dos interrelaciones.

Semántica: son incorrectos todos los siguientes.

1. Usar un atributo en lugar de una entidad.
2. Olvidar colocar una generalización o un subconjunto.
3. Olvidar la herencia en las generalizaciones.
4. Usar una interrelación con un número incorrecto de entidades.
5. Usar de una entidad en lugar de una interrelación.
6. Usar el mismo nombre para dos entidades o dos interrelaciones.
7. Olvidar algún identificador de una entidad.
8. No especificar alguna cardinalidad o especificarla incorrectamente.

Minimalidad. Un esquema es mínimo cuando cada aspecto de los requerimientos aparece una sola vez en el esquema, o en otras palabras, cuando no se puede eliminar ningún concepto del esquema sin perder información. Un esquema no es mínimo cuando es redundante, es decir, cuando tiene alguna información repetida. Ejemplos de redundancia son: atributos derivados, algunos ciclos en el esquema. La redundancia no siempre es mala, pero hay que documentarla muy bien. Para los atributos derivados debe especificarse que lo son y cuál es la fórmula para calcularlos.

Expresividad. Un esquema es expresivo cuando representa los requerimientos de una manera natural.

Legibilidad. Legible significa que se puede leer, esta es una propiedad particular de cada esquema conceptual donde se consideran criterios estéticos. Los esquemas deben ser dibujados sobre una rejilla, todas las líneas deben ser verticales u horizontales, no se usan diagonales ni curvas; se deben minimizar el número de cruces en todo el esquema, lo ideal es tratar de lograr un esquema "planar"; cuando se dibujen generalizaciones, se debe tratar en lo posible de colocar la superclase arriba y las subclases debajo de ésta, análogamente en las interrelaciones de subconjunto, también se debe destacar la simetría, colocando todas las subclases simétricamente con respecto a la superclase.

Autoexplicación Cuando se logra expresar las propiedades del problema con los conceptos del modelo de datos, sin recurrir a otras cosas, se dice que el esquema conceptual es autoexplicativo. Las otras cosas pueden ser explicaciones en lenguaje natural o restricciones explícitas.

Capacidad de Extensión. (Flexibilidad) Describe la facilidad de un esquema conceptual de adaptarse a cambios por nuevos requerimientos. En la medida en que el esquema sea modular y use los conceptos más generales para representar los requerimientos, será más flexible y adaptable a cambios.

5.1. Transformaciones para mejorar la calidad

Si un esquema conceptual no tiene una buena calidad de acuerdo a alguno de los criterios vistos, lo deseable es conseguir alguna transformación del esquema que lo transforme en uno con mayor calidad, pero que preserve el contenido de información del esquema original. El problema está en cómo medir el contenido de información de un esquema conceptual. Es muy difícil definir este concepto formalmente, sin embargo podemos comparar dos esquemas en base a las consultas que cada uno es capaz de contestar. Se dice entonces que dos esquemas conceptuales S_i y S_f son equivalentes a nivel de contenido de información si para toda consulta Q sobre S_i, existe una consulta Q' sobre S_f que expresa lo mismo que Q y da la misma respuesta, y por otro lado, para todo Q' sobre S_f, existe un Q sobre S_i que expresa los mismo que Q' y da la misma respuesta.

Definido así el contenido de información, también podemos hablar de que un esquema S_a tiene mayor contenido de información que otro esquema S_b si existe una consulta Q sobre S_a que no tiene una consulta correspondiente en S_b, pero lo contrario no ocurre.

Con estas definiciones podemos clasificar las transformaciones en:

1. Transformaciones que preservan el contenido de
2. Transformaciones que alteran el contenido de información:
   a) Transformaciones que lo aumentan.
   b) Transformaciones que lo reducen.
   c) Transformaciones que no son comparables.

Con respecto a la *calidad* queremos transformaciones que preserven el contenido de información, pero que mejoren la organización de los conceptos. Nos concentramos en transformaciones para lograr:

- Un modelo mínimo, eliminando ciclos de interrelaciones, atributos derivados y subconjuntos implícitos, y
- un modelo expresivo y auto-explicativo, eliminando subclases "colgantes" en generalizaciones o eliminando entidades "colgantes" en general; creando generalizaciones donde sea apropiado para obtener un esquema más compacto, y creando nuevos subconjuntos, cuando hay una identidad clara y es significativo para el diseño.

Es importante aclarar que, aún cuando la decisión de eliminar la redundancia le corresponde tomarla al diseñador y éste puede justificar dejarla, como la redundancia es una fuente indiscutible de anomalías en la administración de los datos, es fundamental que se indique claramente en el esquema donde existe redundancia.

Hay otra situación donde se puede dar redundancia en un esquema conceptual. Se trata del caso cuando en un mismo elemento del modelo de datos (entidad/interrelación o clase/asociación), se concentran propiedades con semánticas diferentes. Por ejemplo, si se mezclan en una sola entidad, los datos de un contrato de servicio y los datos del cliente que establece el contrato, puede haber redundancia en los datos del cliente, si el mismo cliente puede establecer varios contratos. Esta situación se detecta en el modelo de datos relacional, con la teoría de normalización. Si tuviésemos un mecanismo de aplicar la teoría de normalización a los modelos ERE y OMT, podríamos utilizar la normalización para validar algunos aspectos importantes de calidad, como este de la redundancia producida por la mezcla de diferentes semánticas en el mismo elemento.

6. La Tecnología Objeto-Relacional

La introducción a este tópico podía haber sido la introducción a toda la asignatura, pues en ella vamos a motivar la necesidad de *modelar datos complejos* que requieren de otros paradigmas de modelado. Sin embargo, el curso está naturalmente dividido en dos partes, por un lado los aspectos metodológicos del modelado conceptual donde se hace énfasis en esa etapa del diseño de una base de datos, y por el otro la tecnología que puede apoyar la implementación del esquema conceptual.

El modelo relacional ha sido suficiente para las aplicaciones tradicionales comerciales o de negocios, estas aplicaciones se caracterizan por manejar datos muy simples en grandes volúmenes. Datos simples generalmente se refieren a datos alfanuméricos que, con bastante precisión y facilidad, pueden ser representados en un computador. Estas aplicaciones han evolucionado en cuanto a sus necesidades de manipulación, y de almacenamiento y análisis de datos históricos en lo que se ha denominado "data warehouses" y "OLAP (on-line analytical processing)".

Supongamos una aplicación donde se quieren llevar las historias médicas de los pacientes de una clínica o de un centro integral de medicina. Si se quieren almacenar datos alfanuméricos de los pacientes, como por ejemplo, sus datos personales, lista de las alergias que sufre, historia familiar de enfermedades, tratamientos quirúrgicos recibidos, y resultados de exámenes de sangre, el modelo ER-E es suficiente, y el esquema conceptual se puede implementar eficientemente en un manejador que siga el modelo relacional. Sin embargo, si la historia médica incluye también: imágenes de resonancia magnética de la rodilla del paciente, video de alguna operación, fotos de algún órgano o de los cromosomas encontrados como resultado de una amniocentésis, con anotaciones de los médicos, los modelos ER-E y OMT dejan de ser suficientemente expresivos, y el modelo relacional es completamente inconveniente, pues no tiene estructura ni operaciones apropiadas para manejar esos datos.

Existen muchas aplicaciones con necesidades similares a la aplicación médica descrita anteriormente, por ejemplo, las de CAD/CAM, los sistemas de información geográficos, las bases de datos multimedia, entre otras.

Una primera aproximación a este problema fue la de extender el modelo relacional (en teoría) para dar cabida a estos nuevos datos y sus necesidades de manipulación, lo cual dió lugar a las bases de datos extensibles, que planteaban modelos con extensiones al relacional para albergar otros tipos de datos y sus operaciones. Una de las primeras implementaciones de estas ideas fue el manejador *Postgres* (Stonebraker 1987), el cual todavía existe, con el nuevo nombre de PostgresSQL y es uno de los productos de software libre más robustos en el área de base de datos.

En paralelo, estaban los investigadores del área de orientación por objetos definiendo las bases de datos orientadas por objeto, pues el paradigma de OO es naturalmente extensible, y con ello se podían representar esos datos e incluir sus operaciones encapsuladas en la misma noción de objeto. De estos esfuerzos nacieron varias generaciones de software para manejar bases de datos orientadas por objeto, conocidos como OODBMS.

Otra corriente de pensamiento, propuso liberar al modelo relacional de la restricción de que los dominios de los atributos fuesen atómicos, con lo cual las relaciones no tenían que estar en 1NF (primera forma normal) y se llamaron "non-first normal form relations"

o NFNF o NF^2. Llevando esta noción hasta el extremo, se podía definir una tabla con atributos cuyos valores pudieran ser tablas completas. Por ejemplo, se puede definir una tabla DEPARTAMENTO con cinco atributos: codigo-D, nombre-D, jefe-D, Empleados y Proyectos, donde el atributo Empleados toma como valores, tablas con dos atributos nombre y direccion del empleado, y el atributo Proyectos es otra tabla, con atributos nombre-Proyecto y presupuesto.

Las bases de datos extensibles evolucionaron en lo que hoy en día se llama la *tecnología objeto relacional (OR) (object-relational)*, en la cual se combinan las nociones de extensibilidad, orientación por objetos, y en algunos casos hasta tablas anidadas.

En este curso vamos a hacer una breve revisión histórica de los avances que dieron lugar a la tecnología OR y nos concentramos en esa tecnología para implementar nuestros esquemas conceptuales, como una alternativa actual, práctica y válida al uso de un OODBMS.

6.1. Definiciones

El paradigma de orientación por objetos se desarrolló en las áreas de lenguajes de programación e ingeniería de software. En un lenguaje de programación orientado por objetos, los objetos existen sólo durante la ejecución del programa que los crea. Por otro lado, en una base de datos orientada por objetos, los objetos se crean, pueden ser persistentes y se pueden compartir entre varios programas. Por lo tanto, las bases de datos orientadas por objeto almacenan objetos persistentes en almacenamiento secundario y soportan el compartir objetos entre diferentes aplicaciones.

Los OODBMS son el resultado de integrar la tecnología de base de datos con el paradigma OO. Para soportar los objetos persistentes hace falta agregarle a este paradigma, mecanismos de manejador de base de datos, como por ejemplo, indización de los datos, control de concurrencia y manejo de transacciones.

Desde finales de la década de las 80 se han construido OODBMS, en muy pocos años se desarrollaron tres generaciones de estos productos. La primera generación de OODBMS data de 1986, cuando la empresa francesa Graphael introdujo G-Base al mercado. En 1987, la compañía estadounidense Servio Corporation introdujo GemStone. En 1988, Ontologic introdujo VBase y Symbolics introdujo Statice. El objetivo común de todos estos desarrollos era darle persistencia a los lenguajes de objetos utilizados en inteligencia artificial. Estos eran sistemas independientes, basados en lenguajes propietarios, que no usaban ninguna plataforma industrial estándar. Los sistemas construidos con estos productos residían en los departamentos de investigación de grandes empresas y se construyeron unos 500.

La segunda generación de OODBMS comenzó en 1989 con la liberación del producto Ontos por la empresa del mismo nombre. A Ontos le siguieron los productos: ObjectStore (de Object Design), Objectivity/DB (de Objectivity), y Versant ODBMS (de Versant Object Technology). Estos sistemas ya usaban la arquitectura de cliente/servidor y tenían una plataforma común: C++, X Windows y estaciones UNIX.

Itasca fue el primer producto de la tercera generación de OODBMS y fue liberado en agosto de 1990 (apenas unos meses después de los productos de la segunda generación). Este producto era la versión comercial de Orion, proyecto de MCC (Microelectronics and

Computer Corporation), instituto de investigación financiado por empresas estadounidenses fabricantes de hardware. Los otros productos de esta generación eran O2, producido por la empresa francesa Altair, y Zeitgest, desarrollado internamente por Texas Instruments.

Los productos de la tercera generación de OODBMS tenían características más avanzadas y tenían lenguajes de definición y manipulación de datos que eran orientados por objetos y computacionalmente completos.

En este ambiente, de desarrollo de muchos sistemas por parte de empresas pequeñas, estos sistemas fueron revolucionarios con respecto a los DBMS's existentes, pues se construyeron desde cero con una base muy diferente y con modelos de datos distintos. En este momento se sintió la necesidad de por lo menos definir lo que era un OODBMS y Atkinson en 1989 escribió lo que se llamó el Manifiesto de los Sistemas de Base de Datos Orientados por Objeto, el cual describía lo que eran las características principales de un sistema para poder calificar como OODBMS.

Otras particularidades de esta época eran: que estos sistemas no tenían un modelo de datos común y se habían utilizado sólo en aplicaciones experimentales. La falta de un estándar para las base de datos de objetos era una limitante muy importante de su aceptación en el mercado y por otro lado, uno de los secretos de los RDBMS era precisamente la existencia de un estándar. Como respuesta a ello, en el verano de 1991 se forma ODMG (Object Database Management Group) un consorcio de empresas (casi todas las que desarrollaban OODBMS), para tratar de proveer un estándar. ODMG estaba afiliado al OMG (Object Management Group), establecido en 1989 y cuya principal contribución era la arquitectura CORBA para la interoperabilidad de sistemas distribuidos de objetos. ODMG produjo los primeros estándares en 1993, ellos fueron: ODMG Object model, que define el modelo de datos de las bases de datos orientadas por objetos; ODL (Object Definition Language) que es el DDL para definir un esquema en este modelo de datos; OQL (Object Query Language) que es un lenguaje declarativo, inspirado en SQL, para hacer la correspondencia entre los conceptos del modelo de datos para OODB y los lenguajes considerados en el estándar y para definir cómo los objetos de ODMG podían ser accedidos y manipulados por estos lenguajes (C++, Smalltalk, LISP, Java).

A continuación se enumeran ordenadamente los OODBMS más importantes desarrollados.

- ORION/Itasca (nombre de la versión comercial a partir de 1990)
  MCC Austin, Texas. Won Kim lo desarrolló, el primer producto se construyó desde 1985 hasta 1989. Se han desarrollado tres generaciones de este producto.
- O2 (Consorcio Altair, 1986-1991)
  Está descrito en un libro: "The Story of O2" (Bancilhon 1992). Hacían más énfasis en los lenguajes de programación que en la arquitectura.
- GemStone (1987)
  Es el producto comercial más antiguo que está disponible hoy en día. Extiende Smalltalk en un sistema de base de datos. Ha sido el más visible de todos, hoy existen dos versiones GemStone/S (versión Smalltalk) y GemStone/J (versión Java).
- IRIS/OpenDB (1989)
  Prototipo de investigación de HP, sigue el modelo funcional.

- VBase/Ontos (1988)
  De lenguajes propietarios pasaron a C++. Es totalmente distribuido.
- ObjectStore (1989)
  Tiene soporte para documentos XML.
- POET (1999)
  C++ y Java. Con su Content Management Suite soporta documentos XML y SGML.

Los conceptos fundamentales del paradigma de orientación por objetos son:

- Noción de Objeto y clase a la cual pertenecen que encapsula estructura y comportamiento.
- OID (object id). Igualdad por identidad (idénticos): si son el mismo objeto, es decir, tienen el mismo identificador. Igualdad por valor (iguales): dos objetos son iguales si sus estados son recursivamente iguales.
- Estado de un objeto: valores de las propiedades del objeto, pueden cambiar en el tiempo. Propiedades son atributos e interrelaciones con otros objetos.
- Comportamiento: especificado por las operaciones que pueden ser ejecutadas por el objeto o sobre el objeto, y posiblemente actúan sobre el estado del objeto.
- Clases: todos los objetos del mismo tipo tienen el mismo conjunto de propiedades y de operaciones. *Un tipo de objeto se define a través de una clase.* Extensin de una clase: todos los objetos que pertenecen a ella.
- Herencia: por subtipos, por implementación y por inclusión.

6.2. Manejadores Objeto Relacionales

A diferencia de los OOBDMS que tienen un enfoque revolucionario, los Manejadores de Base de Datos Objeto Relacionales (ORDBMS) son una evolución de los RDBMS, pues integran los objetos al modelo de datos relacional y extienden los RDBMS existentes con características del paradigma orientado por objetos. A principios de los 90, los dos enfoques entraron en conflicto, los OO puros proponían extensiones a los lenguajes de programación OO, y los del otro bando, el enfoque híbrido, proponían partir de las bases de datos relacionales y agregarle extensiones orientadas por objeto.

La idea de los ORDBMS data de 1990 con la publicación del "Third Generation Database System Manifesto" de Stonebraker. La premisa básica de este manifiesto es que la nueva (tercera) generación de manejadores de base de datos, deberían poder manejar objetos y reglas, y deberían ser compatibles con los manejadores de la segunda generación, es decir, los relacionales. Por ese mismo año, UniSQL, Inc., fundada por Won Kim, produjo un manejador objeto relacional, el UniSQL, que usaba SQL/X, una extensión de SQL-92 como lenguaje de base de datos. Durante esa década, otras empresas que se iniciaban, desarrollaron productos objeto relacionales, como Illustra y Omniscience. Ya para 1996, Informix compra Illustra y todos los grandes productores de manejadores de base de datos,

a saber, Sybase, IBM y Oracle, comenzaron a trabajar en productos objeto relacionales. Ese mismo año se logró un consenso en cuanto a las particularidades del modelo objeto relacional que se plasmaron en el estándar SQL-3 y todos los actores en el mundo de base de datos, lo adoptaron, cada uno de ellos ha liberado su producto objeto relacional.

Los sistemas OR son *relacionales* porque soportan SQL, y son orientados por objetos porque soportan datos complejos. Si consideramos la complejidad en los datos y la complejidad en las consultas y las combinamos en dos ejes perpendiculares, obtenemos una matriz con cuatro cuadrantes, a saber: (Navathe 2000 y Stonebraker 1999)

(I) datos simples, consultas simples (editores de texto, hojas de cálculo estilo excell, file system).

(II) datos simples, consultas complejas (RDBMS).

(III) datos complejos, consultas simples (algunos OODBMS, Khoros, sistema de procesamiento de imágenes).

(IV) datos complejos, consultas complejas (ORDBMS).

Al final de la década de los 90, los OODBMS y los ORDBMS dejaron de estar en conflicto, cuando mejoraron las funciones de DBMS ofrecidas por los OODBMS y mejoraron sus facilidades para soportar un lenguaje de consulta declarativo (con la definición de OQL que es muy parecido a SQL-3). Las principales diferencias en estos productos actualmente se concentran en los siguientes aspectos:

- Los OODBMS proveen persistencia para los objetos creados con los lenguajes OO, como Java, C++ y Smalltalk; los programadores definen clases y crean objetos de esas clases, y con los mecanismos de DBMS que tienen, permiten almacenar estos objetos y compartirlos, por lo tanto, la integración con estos lenguajes es transparente. Sin embargo, en los ORDBMS se introduce un API separado (basado en SQL) para manipular los datos almacenados, las definiciones de clase se deben "mapear" a los tipos de datos soportados por el sistema de base de datos.
- A nivel de arquitectura, los OODBMS están centrados en los clientes, manejan un "cache" de objetos en las máquinas cliente y permiten la navegación entre objetos relacionados, lo cual es muy apropiado para ciertas aplicaciones. Por otro lado, los ORDBMS tienen un enfoque mas centrado en el servidor, lo cual es más apropiado para satisfacer muchas consultas concurrentes a los datos.

En esencia, los conceptos de la tecnología objeto relacional son:

- Tipos abstractos de datos (TADs), definidos por el usuario.
- Tipos complejos (o estructurados, o agregados), definidos en base a los tipos atómicos con la utilización de constructores de tipo para crear: conjuntos, tuplas, arreglos, secuencias, entre otros.
- Herencia: definición de tipos de datos como subtipos de otros.

- Tipos referencia. Apuntadores a objetos de otro tipo.
- BLOBs (eran lo único que tenían los RDBMS).

Estos nuevos tipos de datos, necesitan nuevas funciones de manipulación. Estas funciones son:

- Métodos definidos por el usuario: métodos asociados a los TADs.
- Operadores para los tipos complejos (estructurados o agregados): por ejemplo, el constructor conjunto tiene los métodos: pertenece-a, subconjunto de, igualdad de conjuntos, intersección, unión y diferencia de conjuntos.
- Operadores para los tipos referencia.