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

Prof. Soraya Abad Mota

Actualización Octubre 2005

1. Introducción

Objetivos del curso:

1. Conocer 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 modelaje conceptual. Para ello es vital destacar la importancia y el impacto de esta fase en el éxito de un sistema de base de datos.

Contenido del curso:

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. Estrategias de Diseño de Base de Datos.
5. Calidad del esquema conceptual. Criterios y cómo lograrla.
6. Abstracciones. Clasificación, agregación y generalización.
7. Tecnología Objeto-Relacional. Definiciones, SQL-3, traducción de esquemas de objetos UML.
8. OMT: Modelo Dinámico. Conceptos básicos, diagrama de estados.
9. Bases de Datos Activas. Paradigma, triggers.
10. Bases de Datos Deductivas. Paradigma, historia. Otros Modelos.

2. Modelo de datos y proceso de diseño

Modelo (definición 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.

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, opcional o fijo.
- INTERRELACIÓN. Tipos: 1:1, 1:n, n:m. Restricciones de cardinalidad, notación (min,max).
- Interrelaciones especiales. Subclase (is-a). Generalización/Especialización. Categorías.
- Notación gráfica. Diagramas conceptuales.

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.

5. Estrategias de Diseño Conceptual

El diseño conceptual de una base de datos es un proceso iterativo de refinamientos sucesivos. Los fundamentos de una metodología de diseño consisten de primitivas de refinamiento del diseño y estrategias de diseño. Dentro de las primitivas tenemos dos grupos: las *top-down* (son 8) y las *bottom-up* (son 5). Los conjuntos de primitivas pueden tener algunas propiedades deseables como lo son la completitud y la minimalidad. En cuanto a las estrategias, tenemos cuatro: top-down, bottom-up, por conceptos centrales (de adentro hacia afuera) y mixta. (Ver copias del libro BCN 92 para los detalles.)

Adicionalmente, 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.

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.

6. 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.

6.1. 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.

¿ Qué tipo de transformaciones son las primitivas top-down y bottom-up ? Son transformaciones que aumentan el contenido de información. 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.

7. Abstracciones

Nuestro objetivo al elaborar un esquema conceptual era modelar aquellos aspectos relevantes de la realidad de un problema para luego incluirlos en una base de datos. De la *realidad* pasamos al *modelo* aquello que es importante para el uso que se le va a dar al modelo e ignora o desecha los detalles innecesarios. Para poder modelar necesitamos mecanismos que permitan destacar lo importante y desechar lo irrelevante. A estos mecanismos se les llama, *abstracciones*. Las *abstracciones* son procesos mentales que nos ayudan a seleccionar lo relevante de la realidad a modelar, las *abstracciones* más comunes son: clasificación, agregación y generalización.

Clasificación. Es un proceso mediante el cual un conjunto de objetos de la realidad, con propiedades comunes, se agrupan en una sola clase. Por ejemplo, teléfono describe la siguiente clase de objetos: el de mi oficina (que es beige), el rojo de Batman, el de múltiples líneas de la secretaria del departamento, entre otros. No importan las diferencias que existen entre éstos, hemos abstraído de ellos que es un aparato que se enchufa a una línea telefónica para comunicarse, tiene números y un auricular para oir y hablar, y tiene otras teclas. El color, el tamaño y la forma se pueden registrar para cada uno, pero las diferencias en esas características no importan y no se toman en cuenta para decir que todos pertenecen a la misma clase.

Otra forma de definir a la clasificación es que según ella, se define un concepto como una clase de objetos, caracterizada por sus propiedades comunes. Los objetos se pueden clasificar de varias maneras. Son instancias específicas que se agrupan en una clase. Se dice que el objeto es miembro de la clase. Por ejemplo, si tenemos los siguientes objetos: bicicleta, barco, carro, camión, avión, moto y caballo, todos son medios de locomoción y se pueden agrupar en la clase *vehículo* y se dice que la bicicleta es un miembro de la clase *vehículo*. Por otro lado se puede definir la clase de *vehículos mecánicos* como aquella que contiene a los medios de locomoción que no usan combustible derivado del petróleo sino sólo energía mecánica para su funcionamiento, en este caso a esta clase pertenecen solamente la bicicleta y el caballo, de la lista an-

terior. Si le agregamos un burro a la lista, lo podemos incluir en ambas clases, pero si agregamos un tren, éste sólo pertenece a la clase de *vehículos*. Otro ejemplo sería, si tenemos los objetos: corolla, samurai, blazer, jeep, y leganza, los dos primeros son carros japoneses, la samurai y el blazer con vehículos de tracción en las 4 ruedas, corolla y samurai son carros tipo sedan, con lo cual hemos clasificado los objetos de la lista en tres clases distintas.

Agregación. Define una nueva clase a partir de las otras clases, las cuales representan sus componentes. Cada clase componente forma parte de la nueva clase. Hay varias formas que puede adoptar la agregación:

- una clase de objetos físicos está compuesta de otras clases de objetos físicos, por ejemplo: una bicicleta está compuesta de un manubrio, pedales y ruedas;
- un concepto y los atributos que lo describen, por ejemplo, una persona está descrita por su nombre, su cédula de identidad y su dirección de habitación;
- un concepto y los conceptos que lo definen, un ejemplo de esta variante es un propietario se define como la persona que es dueña de un edificio, por lo tanto propietario se define como la agregación de persona y edificio.

En otras palabras, en su forma más general, una agregación es una correspondencia que se establece entre clases, e indica que los miembros de las clases componentes forman parte de los miembros de la clase agregada.

Generalización. Define una interrelación de subconjunto entre los elementos de dos o más clases. Por ejemplo, la clase vehículo es una generalización de la clase bicicleta. Este es otro tipo de correspondencia entre clases que representa la noción de subconjunto o de interrelación is-a.

Por ejemplo, considere la tabla que se muestra a continuación donde se incluye la secuencia y duración de los períodos (tiempos) geológicos.

Era	Período	Época	Duración
Cenozoico	Cuaternario	reciente	últimos 10.000 años
		Pleistoceno	2.5
	Terciario	plioceno	4.5
		mioceno	19
		oligoceno	12
		eoceno	16
		paleoceno	11
Mesozoico	Cretáceo		71
	Jurásico		54
	Triásico		35

Utilizando las abstracciones descritas anteriormente para representar todos los conceptos encerrados en esta tabla (inclusive las líneas divisorias y los encabezados de la tabla), tenemos que los conceptos de *Era, Período* y *Época* se pueden representar en primera instancia con tres clasificaciones, la que construye el conjunto *Era* a partir de los elementos *Cenozoico* y *Mesozoico*, la que construye al conjunto *Período* a partir de los elementos *Cuaternario, Terciario, Cretáceo, Jurásico* y *Triásico*, y la que construye *Época* a partir de los elementos *reciente, Pleistoceno, plioceno, mioceno, oligoceno, eoceno* y *paleoceno*. En otras palabras estamos diciendo que: "*el Cenozoico y el Mesozoico son Eras*", y algo análogo estamos diciendo para el Período y para la Época.

Por otro lado, podemos decir que "las épocas tienen un nombre y una duración", por lo tanto podemos representar *Época* como una agregación de nombre y duración. También podemos representar a las *Era* como un agregado de períodos y a *Período* como un agregado de épocas. El aspecto más complejo de la información contenida en esta tabla, es la representación de la secuencia de eras, períodos y épocas. Una manera de representar esto es definiendo *secuenciaEpoca* como un agregado de *Época* y *Época*, con lo cual se establece cuál época es anterior a otra de a pares, y las propiedades antireflexiva, antisimétrica y transitiva de la relación "es anterior" describen todo el orden total existente entre las épocas. Existen otras maneras de representar esta secuencia, ¿puede ud. sugerir alguna otra?

8. La Tecnología Objeto-Relacional

8.1. Motivación

La introducción a esta tópico podía haber sido la introducción a esta materia, pues en ella vamos a motivar la necesidad de *modelar datos complejos* que requieren de otros paradigmas de modelaje. Sin embargo, el curso está naturalmente dividido en dos partes, por un lado los aspectos metodológicos del modelaje 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 amniocentécis, 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.

8.2. 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 accesados y manipulados por estos lenguajes (C++, Smalltalk, LISP, Java).

A continuación listamos ordenadamente los OODBMS más importantes del mercado.

- 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:

- Objetos
- 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: 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.* Extent de una clase: todos los objetos que pertenecen a ella.
- Herencia: por subtipos, por implementación y por inclusión.

8.3. 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 OO.

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 OO 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.

8.4. SQL3 (SQL:1999)

Después de la definición oficial del modelo relacional por parte del Dr. Codd (1970) había que especificar un lenguaje de definición y manipulación de datos para el modelo. El primer lenguaje de esta naturaleza se llamó *SEQUEL* y sus siglas significaban, en inglés, Structured English Query Language, este lenguaje era el *API (application program interface)* del Sistema R, que era el manejador relacional que estaba desarrollando IBM a principios de los años 70. El primer prototipo de SEQUEL salió entre 1974 y 1975, después hubo un par de versiones más y se le cambió el nombre a SQL, pero se seguía pronunciando "sequel". El proyecto que desarrollaba el Sistema R estuvo activo desde 1971 hasta 1979 y después evolucionó en el Sistema R* que era un sistema de base de datos distribuidos.

Pero el Sistema R era un sistema de investigación, después IBM desarrolló un manejador comercial, llamado SQL/DS, que introdujo al mercado en 1981 y en 1983 liberó la primera versión de su manejador DB2. En particular, otra empresa: Relational Software, Inc. sacó otro producto al mercado que le ganó en ventas al producto de IBM, esta empresa luego se convirtió en Oracle Corporation, Inc. En paralelo, otras empresas estaban desarrollando productos relacionales.

En 1986 SQL se convirtió en un estándar legal establecido por el *ANSI (American National Standards Institute)*, la especificación de este estándar se llamó *SQL-86*, después vino otro en 1989: *SQL-89*, y uno en 1992, que fue una revisión importante del anterior, llamado *SQL-92*. A partir de este estándar se sabía que existían muchos aspectos por resolver para que realmente SQL pudiera ofrecer las capacidades de base datos requeridas y para poder satisfacer las necesidades de los usuarios. Durante siete años se produjeron varios estándares que atacaban aspectos específicos de SQL, hasta que en 1999, se produjo el siguiente estándar importante de SQL, el SQL:1999 o SQL3. Estos estándares intermedios fueron:

CLI-95 El estándar SQL/CLI (CLI por *Call Level Interface*), cuya implementación más conocida es el estándar *ODBC (Open Database Connectivity)*.

PSM-96 El estándar SQL/PSM (PSM por *Persistent Stored Modules*), el cual especifica la sintaxis de la lógica procedimental de los módulos del servidor de SQL. A pesar de que los manejadores comerciales tenían la capacidad de usar "stored procedures" no había ningún estándar al respecto.

OLB-98 El estándar SQL/OLB (OLB por *Object Language Bindings*) provee la habilidad de incluir comandos de SQL en programas de Java y está basado en el paradigma de JDBC, el cual utiliza iteradores (iterators) que son "fuertemente tipeados".

En 1999, después de esta secuencia de estándares intermedios, finalmente fue aceptado y publicado el SQL3 o SQL:1999. En él participaron las dos organizaciones oficialmente activas en la estandarización de SQL, estas organizaciones son: ANSI e ISO (*International Organization for Standardization*), la primera es una organización estadounidense, la segunda está basada en Europa. Dentro de estas organizaciones hay comités específicos que trabajan en este desarrollo. Dentro de ISO está el ISO/IEC JTCI (Joint Technical Committee 1), que depende del International Electrotechnical Commission, y cuya responsabilidad

es el desarrollo y mantenimiento de estándares relacionados con las Tecnologías de la Información. Dentro del JCT1, el subcomité SC32 se formó recientemente su título es "Data Management and Interchange" y tiene que ver con estándares relacionados a base de datos y a metadatos, desarrollados por otras organizaciones. El SC32 además formó grupos de trabajo (Working Groups) que son los que realmente hacen el trabajo técnico, en particular el WG3 de "Database Languages", es el responsable por el estándar SQL, y el WG4 es el encargado de SQL/MM (SQL Multimedia).

En los E.E.U.U. los estándares de tecnología de la información están a cargo del comité acreditado por ANSI que es un Accredited Standards Development Committee llamado *NCITS (National Committee for Information Technology Standardization)*, que antes se conocía como "X3". Dentro de NCITS el Comité Técnico H2 (antes "X3H2") es el responsable de varios estándares relacionados con el manejo de datos, en particular de SQL y SQL/MM.

A partir de 1999, la concepción del estándar se dividió en porciones, cada una de las cuales aborda un aspecto específico de SQL, estas porciones son:

- *Foundation*: constituye el corazón del estándar.
- *SQL/CLI*: Call Level Interfaced.
- *SQL/PSM*: Persistent Stored Modules.
- *SQL/Bindings*.
- *SQL/XA*.
- *SQL/Temporal*.

En el año 2003 salió un nuevo estándar de SQL, SQL:2003, el cual introdujo la noción de XML y estandarizó los generadores de secuencias o valores auto generados, esto incluye columnas que se utilizan como identificadores.

8.5. Características Objeto-Relacionales

A SQL3 se le ha llamado informalmente "SQL orientado por objetos", pero SQL3 va más allá de SQL-92 más la tecnología de objetos. Este nuevo estádar provee muchas mejoras sobre SQL-92 que no se limitan a la inclusión de los conceptos OO. Aparte de algunos nuevos tipos de datos y nuevos predicados, hay tres aspectos fundamentales de SQL:1999 que vamos a reseñar en este curso:

- Orientación por objetos, aquí se concentran las características denominadas objeto relacionales y es donde nosotros hacemos énfasis. Veremos en detalle estas características.
- Bases de Datos Activas, lo cual se refleja en la especificación de triggers y assertions en el estándar. Este aspecto lo cubrimos luego, cuando veamos el modelo dinámico y el paradigma de base de datos activas.

- Nueva semántica, nos referimos específicamente a la habilidad de SQL3 de definir consultas recursivas, con lo cual se puede computar la clausura transitiva, eliminando así la restricción que existía en el álgebra relacional. El paradigma envuelto en esta nueva habilidad de SQL es el de Bases de Datos Deductivas del cual se incluye una visión global al final de este curso.

Comenzamos nuestra cobertura de los conceptos de OO en SQL3 con las estructuras orientadas por objetos, las cuales son extensiones de los tipos. Estas estructuras son:

1. Tipos definidos por el usuario (UDT's, user-defined types), en esta categoría se encuentran:

   a) *Distinct types*: declaran tipos diferentes basados en un tipo ya definido, formalmente, a pesar de que el tipo base es el mismo, los nuevos tipos declarados son diferentes y las operaciones que los combinan dan error.
   CREATE DISTINCT TYPE usDollars AS decimal(9,2)
   CREATE DISTINCT TYPE canadianDollars AS decimal(9,2)
   A pesar de que los dos tipos definidos en este ejemplo son números reales, no se pueden sumar valores en usDollars con valores en canadianDollars, pues da error.
   Se pueden definir funciones para hacer las conversiones de tipo con la opción de CAST dentro de la definición del tipo (dentro del CREATE TYPE).
   CAST (SOURCE AS DISTINCT) WITH usd_to_real
   especificado dentro de la creación del tipo usDollars, dice que para convertir un valor en usDollars a un real se utilice la función usd_to_real, la cual debe estar definida previamente. Cuando se quiera operar sobre dos tipos diferentes, primero se hace cast de uno al otro o de cada uno a un tipo base y luego si se puede operar sobre los valores. Si se declaran las siguientes variables:
   DECLARE VARIABLE X decimal(9,2)
   DECLARE VARIABLE y usDollars
   No se puede sumar directamente X + Y, sin embargo si se puede hacer
   X + CAST(Y AS decimal(9,2)).

   b) *Row type*: se utiliza para definir un registro que contiene campos de varios tipos, es un tipo compuesto por otros tipos. Por ejemplo, se puede definir el tipo compuesto telefono de la siguiente forma:

```
create row type telefono_t (
       codigoArea    varchar(3),
       numTelefono   varchar(7));

create table TablaTelefonos of type telefono\_t;
```

   Como se ilustra en el ejemplo anterior, una vez definido un row type éste se puede usar para definir una tabla con filas del tipo de ese row type. De hecho, esta es la forma de darle persistencia al tipo. Los *row type* también se pueden utilizar para definir el tipo de una columna de una tabla y que esa tabla tenga otras columnas.
   Adicionalmente, se pueden definir funciones que devuelvan valores del tipo de un *row type*.

Con los *row type* se puede utilizar la notación de punto (dot notation) para acceder a elementos del registro compuesto.

c) *Abstract data type (ADT)* (tipo abstracto de dato definido por el usuario). La definición del ADT incluye la estructura del tipo, las operaciones que definen igualdad y ordenamiento, y las operaciones que definen el comportamiento del ADT. Las operaciones se implementan con procedimientos llamados "rutinas".

```
create type empleado_t as object (
              nombre VARCHAR(100),
              apellido VARCHAR(100),
              telefE   telefono_t,
              direccion direccion_t,
              salario decimal(9,2));

create table tablaEmpleados of empleado_t;
```

La manera de almacenar persistentemente un ADT en una base de datos es declarándolo como el tipo de una de las columnas de una tabla. Los atributos y las operaciones de un ADT pueden ser públicos o privados, los públicos son los únicos que pueden ser accedidos fuera del ADT. Adicionalmente, para cada atributo se definen automáticamente dos funciones: una observador y una "mutator". Se pueden definir atributos virtuales dentro de un ADT, para los cuales no se almacenan valores, sino que se calculan cuando se necesitan a través de las funciones definidas por el usuario de observador y "mutator". Las instancias de un ADT se crean con las funciones constructoras del sistema. Se puede utilizar la notación de punto o la funcional (solo para las funciones) para acceder las partes de un ADT.

En el ejemplo de empleado, sólo definimos la estructura del tipo, pero podemos definir allí las funciones apropiadas para manipular instancias del tipo.

2. *REFerence Type*. Es un tipo especial de datos para definir identificadores de objetos. En términos de lenguajes de programación imperativos es un apuntador a un tipo. Todo tipo complejo o estructurado tiene la posibilidad de definirle un ref para los apuntadores a instancias de ese tipo.

```
create type empleado_t as object (
              nombre VARCHAR(100),
              apellido VARCHAR(100),
              telefE   telefono_t,
              direccion direccion_t,
              salario decimal(9,2),
              trabajaEn REF(depto_t));
```

3. *Collection Types*: conjuntos, listas y multiconjuntos. En SQL:2003 es donde se han definido ampliamente los multiconjuntos.

4. BLOB y CLOB. Objetos grandes de los tipos binario o caracter.

Herencia y delegación.

8.6. Oracle 8i

El ejemplo de manejador objeto relacional que veremos en esta edición del curso es Oracle 8i. Primero cubrimos las características de objetos de ese manejador y luego nos concentramos en cómo traducir esquemas de objetos en UML a Oracle 8i.

En Oracle 8i hay varios aspectos nuevos que implementan las nociones de objeto definidas en SQL3. Estos aspectos se describen a continuación.

- *Object types*, son tipos abstractos de datos (TADs) definidos por el usuario o por el sistema. Tienen dos componentes: los atributos, en cuya definición se pueden usar los tipos definidos en Oracle u otros TADs, y los métodos, que son funciones o procedimientos escritos en PL/SQL o en algún otro lenguaje soportado por Oracle. Un ejemplo sencillo de definición de tipo de objeto es:

```
CREATE OR REPLACE TYPE departmento AS OBJECT (
    Code          VARCHAR(20),
    Name          VARCHAR(40),
    MEMBER FUNCTION getStudents     RETURN personsArray,
    MEMBER FUNCTION getFaculty      RETURN personsArray,
);
/
< definicion de las dos funciones >
/
```

  Los *object types* se pueden usar para definir:

  - *object tables* (tablas objeto) cuyas tuplas tienen identificadores de objeto (OID), un ejemplo de esta definición es:
    CREATE TABLE departmentos OF departamento;
  - objetos contenidos (embedded objects), los cuales no tienen OID, definen estructuras de registro complejas y se usan para definir atributos de la tabla objeto;
  - el tipo de un atributo en otra tabla, en una tabla relacional las columnas pueden ser del tipo de un TAD o de un tipo complejo. Por ejemplo:

```
CREATE OR REPLACE TYPE persona AS OBJECT (
    Nombre          VARCHAR(40),
    Telefono        VARCHAR(20) );
CREATE TABLE contactos (
    contacto      persona,
    fecha         DATE) ;
```

  En Oracle 8i no hay jerarquías de object types; en Oracle 9i si, pero con *single inheritance*, es decir sólo se permite la herencia simple.

- *Object views*, para ver esquemas relacionales como objetos.
- Nuevos tipos, además de los tipos tradicionales, predefinidos en Oracle 7:

- VARRAYs y nested tables, dos nuevos tipos que permiten que las columnas de una tabla sean una colección estructurada de datos.
- REFs, son referencias a objetos, se utilizan para almacenar apuntadores lógicos a objetos.
- LOBs, arreglos muy grandes de bytes sobre los cuales hay una serie de operaciones predefinidas.

8.7. Traducción de esquemas de objetos en UML a Oracle 8i

La traducción propuesta en esta sección sigue los lineamientos presentados por Urban y Dietrich en el capítulo 2 del libro [OBJ DB] de la bibliografía del curso.

Las indicaciones generales para traducir esquemas conceptuales de objetos escritos en UML a esquemas objeto relacionales de Oracle 8i son las siguientes:

- Las *clases* se traducen en *object tables*, donde el tipo del object table debe definirse primero como un *object type*.
- Las asociaciones 1:1 o 1:n entre clases se pueden traducir como
  - referencias entre objetos, utilizando el tipo REF, o
  - atributos derivados, que se calculan a través de una función, el atributo derivado sólo puede utilizarse en un lado de la asociación.
- Las asociaciones 1:n o n:m entre clases se pueden traducir como
  - colecciones estructuradas, como VARRAYs o nested tables
  - atributos derivados, utilizados sólo en un lado de la asociación
- Para las generalizaciones se puede utilizar
  - una variable de superclase que aparece en las subclases o
  - un object type que representa la jerarquía aplanada.
- Si las referencias a objetos se mantienen explícitamente en ambos lados de una asociación, se pueden construir triggers para generar y mantener las asociaciones inversas automáticamente o se puede utilizar algún otro mecanismo automático para controlar la redundancia.
- Se pueden crear *object types* para representar estructuras definidas por el usuario u otros tipos complejos que se puedan necesitar en una aplicación. Esos tipos se pueden entonces utilizar para crear objetos contenidos dentro de tablas relacionales o tablas objeto.