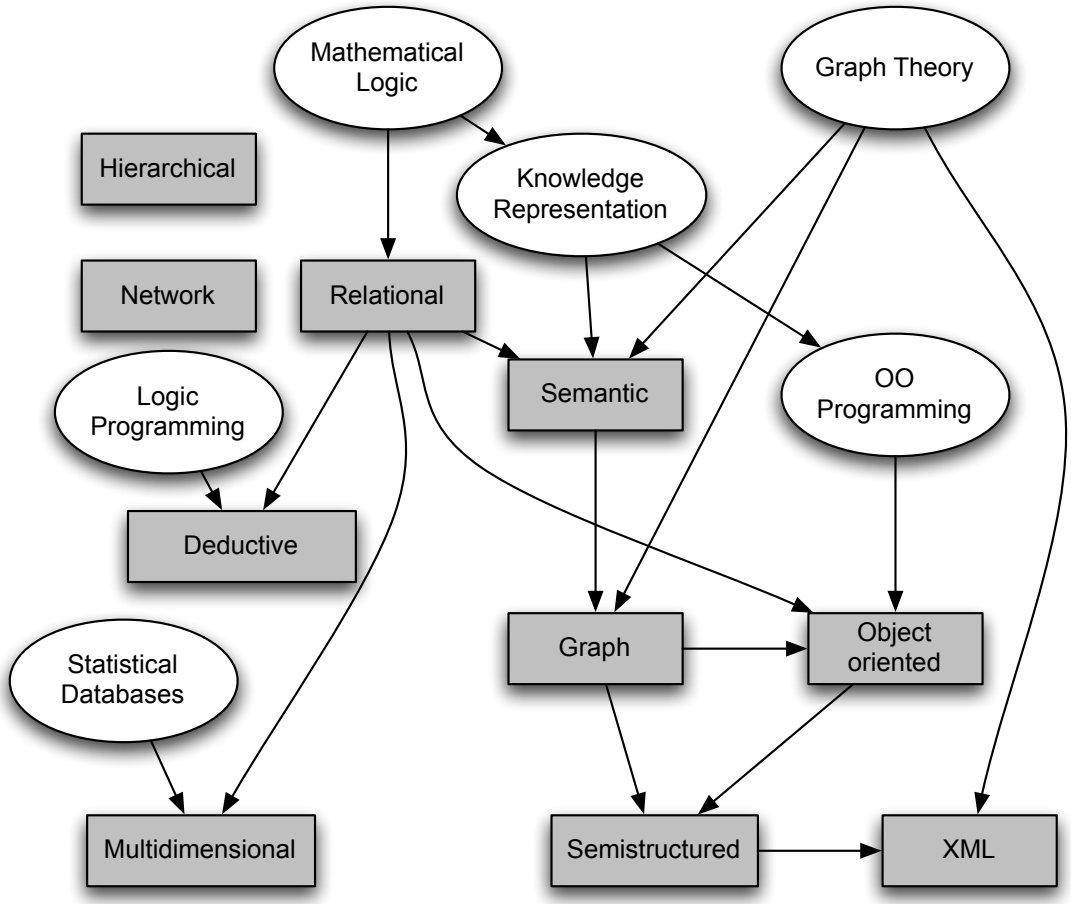
Modelo Orientado al Objeto y Modelo Objeto-Relacional

M. Andrea Rodríguez-Tastets

Universidad de Concepción,Chile
www.inf.udec.cl\ ∼andrea
andrea@udec.cl

II Semestre - 2013

Contexto de Evolución de Bases de Datos



Comparación básica de los Modelos

Modelo	Nivel Abstración	Structure base	Focus de nformación
Network	physical	pointers+records	records
Relational	logical	relations	data+attributes
Semantic	user	graph	schema+relations
Object-Oriented	physical/logical	objects	object+methods
Semistructured	logical	tree	data+components
Graph	logical/user	graph	data+relations

Bases de datos relacionales son buenas para

- Almacenamiento de gran volumen de datos
- Manejo de integridad a través de normalización y restricciones de integridad
- Manejo de transacciones
- Toda clase de optimizaciones dentro del sistema de gestión de bases de datos
- Numerosas características de actualización segura y recuperación de desastres

Motivación: OODBMS

- Confieren al diseñador la posibilidad de especificar estructura de objetos complejos
- Confieren al diseñador la posibilidad de especificar operaciones sobre objetos
- El creciente uso de lenguajes de programación orientados a objetos permite la integración más natural entre lenguajes y bases de datos

Conceptos de orientación al objeto

- Objecto: Tiene dos componentes: estado (valor) y comportamiento (operaciones)
- La estructura interna de los objetos incluye la especificación de variable de instancia, lo que es similar al concepto de atributo
- Algunos sistema de OO exigen que todas las operaciones que se puedan definir sobre un objeto deben estar predefinidas. Esto hace obligatorio el encapsulamiento completo
- Las operaciones se definen en dos partes: signatura o interfaz y método o cuerpo
- La herencia es otro concepto importante que permite la reutilización de definiciones
- Las relaciones binarias explícitamente se representan por referencias inversas
- El polimorfismo de operadores permite aplicar una operación a diferentes tipos de objetos y la sobrecarga permite que implementaciones distintas existan de una operación dependiendo del tipo de datos

Identidad de Objetos

- Un OODBMS proporciona una identidad única a cada objeto almacenada independientemente en una base de datos
- Esta identida suele implementarse con identificador de objeto único (OID), el cual es inmutable
- Por lo tanto el OID no puede depender de ningún atributo
- A veces la dirección física se usa como OID pero implica usar un redireccionamiento en caso de cambio de posición
- Los OODBMS representan tanto objetos como valores, los cuales no tienen OID y se representan así mismos. Un valor se almacena usualmente dentro de un objeto y no se puede referenciar desde otro objeto a menos que sea por una interfaz o método

Estructura de Objetos

- En un OODBMS el estado de un objeto complejo se puede construir a partir de otros objetos (o de valores) mediante ciertos constructores de tipos
- Formalmente, un objeto $o = (oid, constructor\ de\ tipo,\ estado\ del\ objeto)$
- El modelo de datos usualmente incluye varios constructores de tipos: átomo, conjunto,lista, bolsa,array
- El estado de un objeto se interpreta a partir del constructor. Por ejemplo, si el objeto es del tipo conjunto, entonces los valores son un conjunto de identificadores de objetos del mismo tipo
- El modelo permite una anidación arbitraria de los contructores de conjuntos, listas, tuplas, etc.
- El único caso en el que aparece realmente un valor es en estado de un tipo atómico
- Los constructores de conjuntos, lista, array y bolsa se denominan tipos de colección o tipos masivos. Los contructores de tupla se denominan a menudo tipo estructura

Object Definition Language (ODL): classes

Una clase se define como:

```
class Movie {
        attribute string title;
        attribute integer year;
        attribute integer length;
        attribute enum Film { color, black&white} filmType; };
class Star {
        attribute string nombre;
        attribute struct addr
           {string street, string city } address; };
```

ODL: Relaciones

Una clase se define como:

```
class Movie {
        attribute string title;
        attribute integer year;
        attribute integer length;
        attribute enum Film { color, black&white} filmType;
        relationship Set <Star> stars
           inverse Star: starredIn; };
class Star {
        attribute string nombre;
        attribute struct addr
           {string street, string city } address;
        relationship Set <Movie> starredIn
           inverse Movie: stars; };
```

Operaciones de clase

- La idea principal es definir el comportamiento de un tipo de objeto basado en las operaciones
- En general la implementación de las operaciones se puede hacer en un lenguaje de programación de propósito general
- Los usuarios externos perciben solo la interfaz del tipo de datos, la cual es la signatura de las operaciones
- Normalmente un método se invoca enviando un mensaje al objeto
- Para flexibilizar la implementación en OODBMS, la estructura de un objeto se divide en atributos visibles y ocultos
- El término clase se usa para definir un tipo de objetos junto a sus operaciones de tipo

ODL: Métodos

Se pueden agregar la declaración de los métodos en la definición de las clases.

```
class Movie {
    attribute string title;
    attribute integer year;
    attribute integer length;
    attribute enum Film { color, black&white} filmType;
    relationship Set <Star> stars
      inverse Star: starredIn;
    void starNames(out Set <String>);
```

ODL: Tipos

Los tipos básicos son:

- Tipos atómicos
- Nombres de clases

Los constructores de tipos son:

- Set<T>
- List<T>
- Bag<T>
- Array<T,i>
- Dictionary<T,S>
- Structures

ODL: Tipos (cont.)

- El tipo de relaciones son tipos de clase o un tipo de colección (set, bag,list, array,dictionary)
- Un tipo de atributo se construye a partir de tipo atómico o structure types.

Jerarquía de tipo y herencia

- El concepto de subtipo es útil para definir un nuevo tipo que es similar a uno ya existente.

```
class Cartoon extends Movie {
      relationship Set <Star> voices;}
```

De relacional a Objeto-Relacional

El modelo relacional ha incorporado las siguientes características:

- Tipos estructurados para atributos: tipos construidos a partir de los atómicos y constructores para estructuras, conjuntos y bolsas. Así por ejemplo, un valor de un atributo puede ser una relación completa.
- Métodos.
- Identificadores por tuplas
- Referencias: se pueden usar referencias o punteros a tuplas.

Structured Types SQL99

- UDT User Defined Type:
  - A UDT can be the type of a table
  - A UDT can be the type of an attribute belonging to some table
  - A UDT can inherit from another UDT

```
CREATE TYPE T AS <attribute and method declarations>;
```

Nested Relations

```
CREATE TYPE AddressType AS (
  street CHAR(50),
  city CHAR(20)
);

CREATE TYPE AddressTypeTable
  AS TABLE OF AddressType;

CREATE TYPE StarType AS (
  name CHAR(30),
  address AddressTypeTable
);
CREATE TABLE MovieStar OF StarType;
```

Methods SQL99

- Special operations defined for a type
- In SQL, implementation defined with Presistent Stored Modules (PSM) language

```
METHOD m() RETURNS <TYPE>;
```

Methods SQL99: Example

```
CREATE TYPE AddressType AS (
  street CHAR(50),
  city CHAR(20)
);

METHOD houseNumber() RETURNS CHAR(10);

CREATE METHOD houseNumber() RETURNS CHAR(10) FOR
  AddressType
  BEGIN.... END
);
```

References SQL99 - I

A table which type is a UDT may have a reference column that serves as its "ID"
In `CREATE TABLE` statement, add
`REF IS <attribute name> <how generated>`

where `<how generated>` is either

- `SYSTEM_GENERATED` : DBMS generates unique IDs
- `DERIVED`: DBMS uses primary key of the relation for IDs

References SQL99 - Example

```
CREATE TYPE MovieType AS (
  title CHAR(30),
  year INTEGER
);

CREATE TABLE Movie OF MovieType (
  REF IS movieID DERIVED,
  PRIMARY KEY (title, year)
);
CREATE TYPE StarType AS (
  name CHAR(30),
  address AddressType,
  bestMovie REF(MovieType) SCOPE Movie
);
```

Inheritance of tables

```
CREATE SEQUENCE component_component_id_seq   START 1;


CREATE TABLE ‘‘component"(
    component_id int4 DEFAULT nextval(‘component_component_id_seq ’) NOT NULL,
  component_name varchar(50),
  ip inet,
  po varchar(20),
   mac macaddr,
  department varchar(20),
   CONSTRAINT component_pkey PRIMARY KEY (component_id) );

CREATE TABLE ‘‘computer’’ (
  os varchar(20),) INHERITS(component);
);
```

Objecto-relacional versus Object-oriented

El modelo relacional ha incorporado las siguientes características:

- Objetos y tuplas: Los objetos tienen sus atributos y relaciones. En el modelo relacional, una tupla es también una estructura compuesta solo de atributos. En el objeto-relacional se manejan también referencias como punteros directos a tuplas.
- Extensiones y relacionales: En orientación al objeto, todos los objetos viven en una extensión a una clase. Pero ODL permite definir interfaces con lo cual se puede definir un número de clases que heredan la misma interfaz, mientras cada clase tiene distinta extensión. En el objeto-relacional se permiten diferentes relaciones con el mismo esquema.
- Métodos: Tienen la misma funcionalidad en ambos modelos.
- Sistema de tipo: Son muy similares. Todas las variantes de estos modelos incluyen al menos como constructores set y bags.
- Referencias y ID de objetos: Un modelo orientado al objeto puro, esconde del usuario la generación del ID. El objeto-relacional permite referencias ser parte de un tipo y así un usuario puede saber el ID de una tupla.
- Compatibilidad con versiones previas. Los modelos objeto-relacional permiten usar la conversión desde modelos relacionales puros.