

Bases de Datos OO y Objeto Relacionales

Bases de Datos Avanzadas
Curso 2010/2011

Sergio Ilarri Artigas

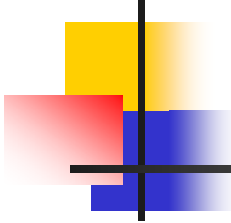
Repaso de OO

¿Es tu mascota orientada a objetos?



I have a cat named Trash... If I were trying to sell him (at least to a computer scientist), I would not stress that he is gentle to humans and is self-sufficient, living mostly on field mice. Rather, I would argue that he is object-oriented.

—Roger King “My Cat is Object-Oriented”



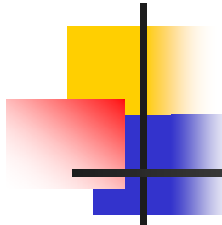
Conceptos básicos de OO (I)

■ Objetos complejos

- Un objeto es un elemento con una estructura (*attrs. de ciertos tipos*) (que pueden ser simples, listas, conjuntos,..) y un comportamiento (*operaciones o métodos*) que corresponden a la definición de la CLASE de la cual el objeto es una INSTANCIA.
 - CLASE define unos ATRIBUTOS y MÉTODOS
 - La clase agrupa a una serie de OBJETOS o INSTANCIAS

■ Identidad de objetos

- Cada objeto tiene un identificador único (OID)
 - Inalterable y dado por el sistema al crearse.
 - Ej: *25#cine* o bien sin indicar la clase: *#35*

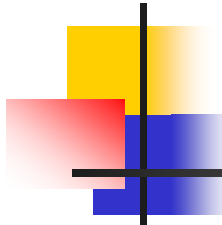


Ejemplo

- Ejemplo: la clase CINE
 - Atributos NOMBRE (string), AFORO (integer), ASIENTOS (array bidimensional de booleanos),...
 - Métodos PONER_LIBRES, RESERVAR_N, ...

NOMBRE: Golem1
AFORO: 250
ASIENTOS: TFTTT..
 FTTFF..
OID: 1#cene

NOMBRE: Golem2
AFORO: 350
ASIENTOS: FTF..
 TFF..
OID: 2#cene



Ejemplo

- O bien ASIENTOS tiene de tipo la clase ASIENTO con atributos ESTADO (booleano), FILA y COLUMNA y con métodos LIBERAR y RESERVAR.
- Por tanto, los valores para 1#ciné en ASIENTOS: 1#asiento 3#asiento serían del estilo de:

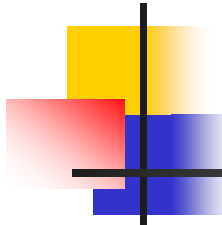
```
ESTADO: T
FILA: 3
COLUMNA: 3
OID: 1#asiento
...
```

```
ESTADO: F
FILA: 3
COLUMNA: 4
OID: 3#asiento
...
```



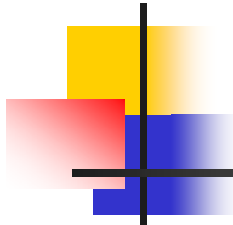
Conceptos básicos de OO (II)

- Identidad de objetos (continuación)
 - Puede haber objetos distintos con los mismos valores...
 - Predicados de igualdad
 - o1 idéntico a o2 si contienen el mismo OID.
 - o1 igual de manera superficial a o2 si el contenido de los objetos es idéntico.
 - o1 igual de manera profunda a o2 si el contenido de los valores escalares es igual y, si los valores son OIDs, si son iguales de manera profunda.
 - Operaciones de copia de objetos
 - Copia superficial: devuelve un nuevo objeto con los mismos valores (escalares y OIDs).
 - Copia profunda: crea un nuevo objeto con los mismos valores escalares y si son OIDs con copias profundas de dichos objetos.



Ejemplo

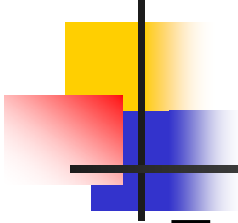
- Podríamos tener 2 cines con igual nombre, aforo y estado de los asientos y ser distintos...
- En Pamplona y en Logroño hay cines Golem. Supongamos que tengan igual aforo y que los identificadores de ambos sean 1#cine y 75#cine.



Ejemplo

- 1#ciné NO ES IDÉNTICO A 75#ciné
- 1#ciné NO ES IGUAL DE MANERA SUPERFICIAL A 75#ciné, si obviamente la lista de asientos se refiere a distintos objetos (con distinto oid)
- Es posible que en un momento determinado 1#ciné sea IGUAL DE MANERA PROFUNDA A 75#ciné

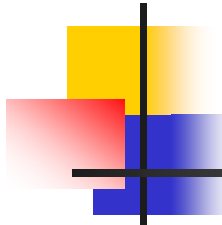
2 sillas distintas pueden ser iguales...



Conceptos básicos de OO (III)

- Encapsulación

- Cada objeto tiene una parte que constituye su INTERFAZ y otra que constituye su IMPLEMENTACIÓN.
- Sólo se puede acceder a cada objeto a través de su INTERFAZ, enviando órdenes para que ejecute MÉTODOS.
- El objetivo es encapsular los DATOS y los PROGRAMAS dentro de los OBJETOS.



Ejemplo

IMPLEMENTACIÓN

OID: 1#cine
NOMBRE: Golem1
AFORO: 250
ASIENTOS: 1#asiento,
3#asiento, ...

API

PONER_LIBRES()

RESERVAR_N(n) ->
lista_asientos

El objeto se crearía al ejecutar `CREAR_CINE('Golem1',250)`, que en realidad es un método de la CLASE. Dicha llamada devolvería el OID asignado, que en este caso sería 1#cine. Al definir dicho método habría que crear tantos objetos de la clase asientos como diga aforo. Para ello habría que llamar al método `CREAR_ASIENTO(...)`.



Conceptos básicos de OO (IV)

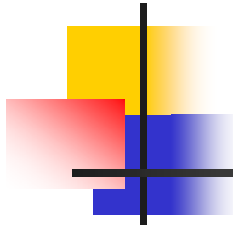
- Diferencia entre tipos y clases
 - Un tipo define una estructura que se utiliza para comprobar que no hay errores en tiempo de compilación. Todo valor de los que aparece en un programa DEBE SER de algún tipo.
 - Una clase está formada por un tipo(s), unas operaciones y un conjunto de instancias de dicho tipo(s).
 - TAD = tipo + operaciones + encapsulación
 - clase = TAD + herencia + cjto de instancias



Conceptos básicos de OO (V)

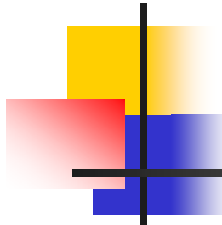
■ Herencia

- Una clase A se puede definir como subclase de otra clase B.
- En ese caso, todos los atributos y métodos de la clase B son heredados por la clase A.
- En algunos casos se permite herencia múltiple, esto es, que una clase sea subclase de más de una clase. En ese caso, hereda las propiedades de todas sus superclases (problemas...).
 - NOTA: Nos referimos a subclase directa en el árbol.



Ejemplo

- Clase *Persona* con atributos NOMBRE, DIR y métodos NACER, COMER, DORMIR, ...
- *Alumno* is-a *Persona* con atributos CURSO y método ESTUDIAR (y que hereda los de persona).
- *Profesor* is-a *Persona* con atributos ASIGNATURA y método ENSEÑAR (y que hereda los de persona).



Ejemplo

- *Alumno-Profesor* is-a *Alumno*, *Profesor* (y que hereda los de alumno y los de profesor).
- Si tuviéramos *Catedrático* is-a *Persona* NO SE CONSIDERA QUE HAY HERENCIA MÚLTIPLE (aunque sea subclase de profesor y de persona, ya que no es subclase directa de persona en el árbol).



Conceptos básicos de OO (VI)

- *Overloading* (sobrecarga), *overriding* (imposición), *late binding* (asociación retardada)
 - Un sistema soporta “overloading” si distintas clases pueden tener propiedades (o métodos) con el mismo nombre.
 - Si se producen conflictos con los nombres de una subclase y sus superclases entonces prevalece el de la subclase (“overriding”).
 - Cuando se invoca un método de un objeto, en tiempo de ejecución, se busca el código en la clase a la que pertenece y si no se encuentra, entonces se va buscando transitivamente por sus superclases (“late binding”).



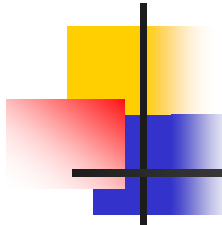
BDOO: Motivación

- Aplicaciones “tradicionales” de BD donde
 - Existen muchos datos almacenados en registros (pocos tipos de registros), generalmente de longitud fija con campos atómicos (en 1FN) y de tamaño pequeño
 - Esquemas de BD casi no cambian y están en 1FN
 - Las transacciones son generalmente cortas
- Existen nuevas aplicaciones de BD
 - Aplicaciones de diseño: CAD, CASE,...
 - Ofimática
 - Sistemas de Información Geográfica (GIS)
 - Repositorios multimedia
 - Sistemas expertos de BD (manejar conocimiento)
 - Gestión de documentos



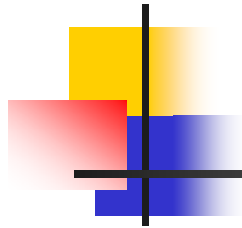
BDOO: Motivación

- Las nuevas aplicaciones de BD necesitan:
 - Esquemas dinámicos
 - Más entidades distintas con probablemente menos datos (ocurrencias) en dichas entidades
 - Campos de longitud variable y que contengan más tipos de datos: gráficos, sonidos, textos ...
 - Realizar operaciones complejas
 - Distintas versiones de los datos
 - Manejar transacciones largas



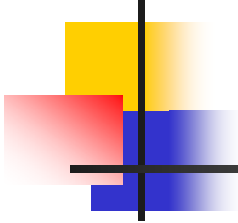
BDOO: Motivación

- Diversas tendencias en BD:
 - Modelos semánticos de datos
 - Bases de datos históricas (temporales)
 - Bases de datos que no están en 1FN (atributos multivaluados)
 - Sistemas expertos de BD (integración IA - BD)
 - Lenguajes de programación de BD: DB + LP
 - BD deductivas: fusión lenguajes de programación lógica + SGBD
 - BD funcionales: programación funcional + SGBD
 - BDOO: programación orientada a objetos + persistencia



BDOO: Motivación

- En relación con la OO, dos aproximaciones:
 - Bases de datos orientadas a objetos
 - $\approx$ Intento de añadir funcionalidad de BD a un lenguaje de programación
 - Alternativa a los SGBD relacionales
 - Bases de datos objeto-relacionales
 - $\approx$ Intento de extender los SGBD relacionales con funcionalidades de OO



Algunas Herramientas de los SGBD Relacionales

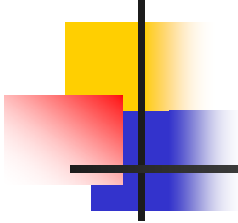
- Los SGBD Relacionales proporcionan:
 - Formularios para hacer entrada de datos
 - Interfaces (similares a hojas de cálculo) para ver datos
 - Generadores de informes
 - Facilidades para escribir SQL embebido desde LP
- SQL embebido (a utilizar desde un Lenguaje de Prog.)
 - Definir tipos, conocidos por el SGBD distintos a LP
 - Conexión a la BD
 - Captura de excepciones (errores)
 - Seleccionar tuplas simples desde una BD
 - Seleccionar varias tuplas (uso de cursores)
 - Ejecutar sentencias SQL dinámico (en tiempo de ejecución)



Problemas del Modelo Relacional (I)

- SQL no es computacionalmente completo. Se necesitan LP -----> *Mismatch impedance*
 - Con **los tipos**: el SGBD y el LP tienen distintos tipos
 - Los LP no ofrecen un tipo primitivo relación que tome como valor un CONJUNTO DE VALORES.
 - Es necesario utilizar CURSORES para tratamiento secuencial dentro de un programa.
 - Los tipos abstractos de datos que se pueden crear con LP habría que guardarlos en tablas para darles persistencia.
 - Con **la estrategia de evaluación**
 - El LP hace una pregunta SQL, el SGBD obtiene la respuesta y la guarda en un lugar intermedio para dársela al LP (traducida). Tal vez éste no pida más datos...

Problemas del Modelo Relacional (II)



- Poder limitado de modelado
 - El modelo relacional tiene como único tipo de datos a las TABLAS con ATRIBUTOS.
 - Sin embargo nos gustaría poder definir
 - Entidades con sus propiedades (multivaluadas)
 - Generalización / Especialización de entidades
 - Relaciones entre entidades (con restricciones)
 - ...

Problemas del Modelo Relacional (III)

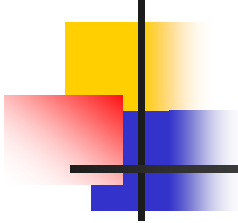
■ Complejidad del entorno

- Los programadores deben saber programar con el SGBD (SQL) y con el LP.
- Hacer que programas ya existentes que trabajan con ficheros lo hagan con BD es duro.
- Para añadir interfaces a los programas de BD hay que manipular otros tipos de objetos gráficos que no pueden ser guardados en la BD.
- Para cambiar de una plataforma a otra, todos los distintos componentes usados deben ser soportados de manera consistente en la nueva.



Qué son las Bases de Datos Orientadas a Objetos (BD OO)

- $BD OO = BD + OO$
- Características BD
 - Persistencia + Concurrencia + Transacciones + Recuperación + Lenguajes de Interrogación / Definición / Manipulación + Integridad + Seguridad + Eficiencia (+ Versiones)
- Características OO
 - Tipos Abstractos de Datos (TAD) + Herencia + Identidad de Objetos
[TAD = Tipos + Operaciones + Encapsulación]



Añadiendo Persistencia a los Lenguajes de Programación

- Java Persistence API (JPA)
- C++ con persistencia
- ...

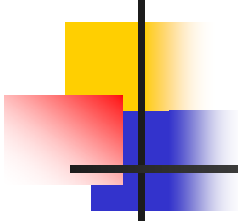


Ejemplo: C++ Persistente (I)

- Es posible escribir programas C++ con todas las características de OO comentadas.
- Problema: las características de BD no se conseguirían.
- Intento de solución: extender C++ para que permita definir clases persistentes.

- persistent class A: public B,..D {}

NOTA: Las clases persistentes no deben contener punteros a clases no persistentes !



Ejemplo: C++ Persistente (II)

- Sin embargo, el resto de características BD siguen sin obtenerse
 - Lo peor: el lenguaje de interrogación es navegacional o procedural.
 - Lo mejor: no hay mismatch impedance ni el resto de problemas del modelo relacional comentados.



Diseño de BD OO: Método 1 (I)

- Para diseñar BD generalmente se usa un modelo de BD semántico llamado Entidad-Relación (extendido) de Chen.
- Los pasos que se pueden seguir son:
 - Obtener el esquema E-R extendido
 - Normalizar dicho esquema E-R
 - Obtener las tablas correspondientes al E-R
 - Normalizar las tablas relacionales
 - Obtener el E-R al que corresponderían dichas tablas
 - Traducir el esquema E-R a un esquema OO



Diseño de BD00: Método 1 (II)

- Las entidades y relaciones del E-R se pueden traducir a clases y atributos OO.
 - Entidad -----> Clase
 - Entidad especialización -----> Subclase
 - Relación 1:1, 1:N -----> Atributo de Clase
 - Relación N:M -----> 2 atributos sobre Clases
 - Relaciones de grado mayor que 2 o de tipo N:M con atributos -----> Clase
 - También se pueden definir métodos
 - Para atributos calculados, realizar tareas (imprimir..)



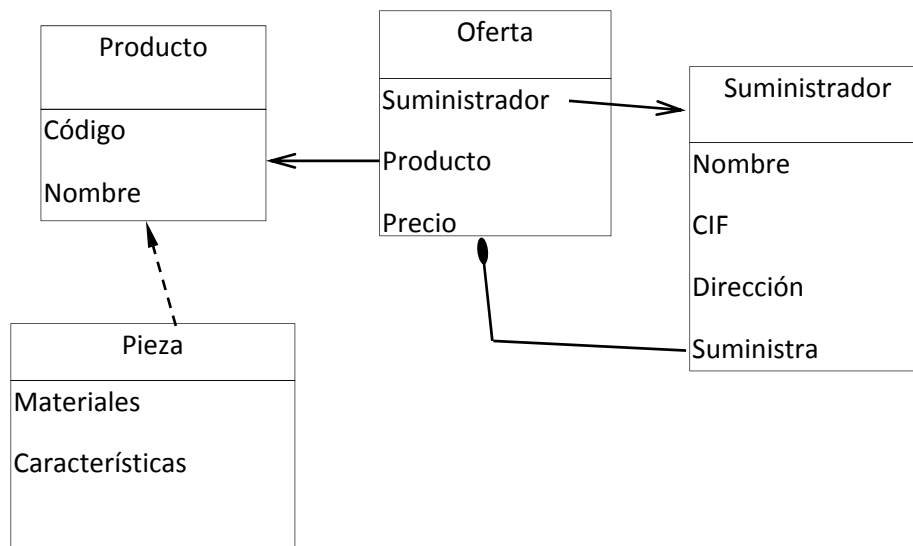
Diseño de BD00: Método 2

- Otra aproximación:

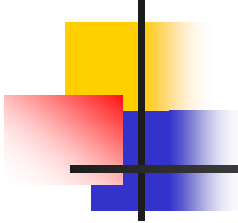
- Diseño conceptual utilizando diagramas de clases UML extendidos con estereotipos
- Diseño lógico utilizando SQL:1999 (o alguna notación gráfica similar a los diagramas relacionales)

Diseño de BD00: Propuesta de Notación Sencilla

Representación de un conjunto de clases y su interrelación a través de atributos:

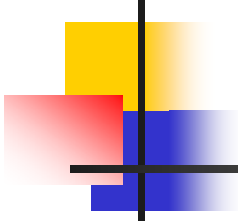


- > Especialización
- > Atributo monoevaluado
- Atributo multievaluado



Ejemplos de SGBD OO “Puros”

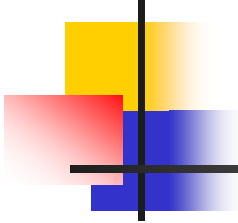
- ObjectStore (Progress Software)
- Versant Object Database
- db4objects (Versant) → open source
- GemStone
- O2
- Iris
- Orion
- ODE



Ejemplos de SGBD Objeto-Relacionales

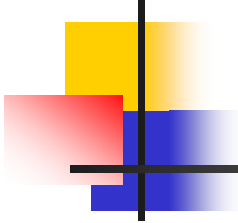
- Oracle (Oracle Corporation)
- Universal Server (Informix)
- DB2 (IBM)
- Cloudscape (IBM)
- PostgreSQL (open source)

Crítica a los SGBD00: limitaciones (I)

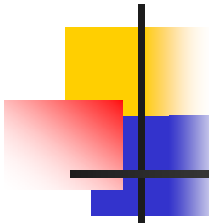


- Lenguaje de preguntas:
 - No son compatibles con ANSI-SQL
 - No incluyen preguntas anidadas, unión, intersección, funciones de agregación, *group by*, ...
- No soportan creación de vistas
- No permiten que los usuarios controlen privilegios
 - En SQL se puede hacer GRANT, REVOKE,...
- No dejan cambiar clases dinámicamente (añadir atributos...)
 - En SQL se puede hacer ALTER TABLE ...

Crítica a los SGBDOO: limitaciones (II)

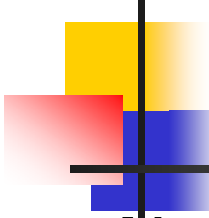


- Generalmente los usuarios deben manejar los *locks* (transacciones)
- Capacidades limitadas para hacer “tuning” de la BD
- Distintos OID en distintas BDOO
- Relacional: operaciones cerradas (resultados son rel.)
OO: operaciones sobre clases dan conjuntos de OID



Crítica a los SGBDOO: mitos (I)

- Los SGBDOO son mucho más rápidos que los relacionales.
 - En realidad sucede si la aplicación navega entre objetos (OID) que están cargados en memoria principal.
- Se elimina la necesidad de ejecutar joins.
 - No eliminan. Reducen el nº de joins (al navegar por atributos)
- Se elimina la necesidad de usar claves.

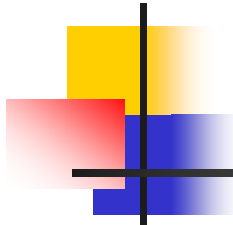


Crítica a los SGBDOO: mitos (II)

- No se necesitan lenguajes asercionales. No, eso viene porque al principio no ofrecían dichos lenguajes...
- El procesamiento de preguntas puede violar la idea de encapsulación
 - Acceder atributo [pepe.nombre] vs. método [pepe.get_nom()]
- Pueden soportar mejor versiones y transacciones de larga duración. No, en BD relacionales también se podría (aunque quizá no se ha tratado lo suficiente).
- Soportan datos multimedia. En principio mejor que con relacionales. Quedan cuestiones que resolver...

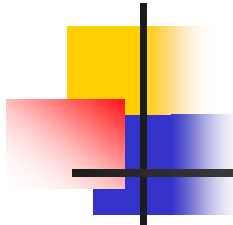


- OQL = Object Query Language
- ODL = Object Definition Language
- Estándar del Object Database Management Group (ODMG) <http://www.odbms.org/odmg/>
 - Finalizó su trabajo en 2001
- En Febrero de 2006, el Object Management Group (OMG) decide trabajar en un estándar de BD de objetos de 4ª generación
- Hoy en día: JDO (Java Data Objects) y JPA (Java Persistence API).



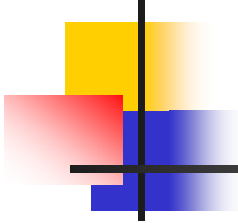
SQL:1999 (I)

- Estándar para el modelo objeto-relacional y la base para diversos SGBD objeto-relacionales
 - Introduce User-Defined Types (UDTs)
 - Posibilitaría almacenar objetos (por ejemplo, objetos Java) como stream de bytes en un BLOB o mapeándolos en filas de una tabla tipada
 - Extiende SQL-92 y, por supuesto, SQL-86 y SQL-89



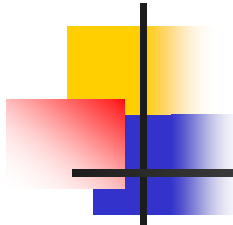
SQL:1999 (II)

- Soporte para nuevos tipos de datos:
 - LOB: BLOB, CLOB
 - BOOLEAN (true, false, unknown)
 - ARRAY, ROW
 - ¿Violación de la 1FN?
 - Tipos de datos estructurados definidos por el usuario
 - Atributos (anidad.) y métodos (vs. funciones)
 - Jerarquías de tipos (no herencia múltiple): supertipos y subtipos



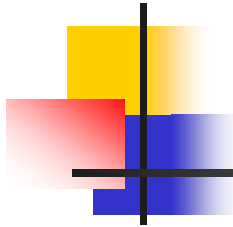
SQL:1999 (III)

- ¿Y dónde están los objetos?
 - Tablas tipadas
 - Tipos REF:
 - Identifica filas de una tabla tipada



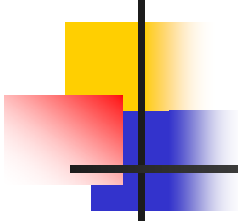
SQL:1999 (IV)

- Nuevos predicados:
 - “Distinct types”
 - Basados en el mismo tipo de dato primitivo
 - SIMILAR (expresiones regulares)
- Otros:
 - *Savepoints*
 - Preguntas recursivas
- Posteriormente: SQL:2003, SQL:2006, SQL:2008



Oracle...

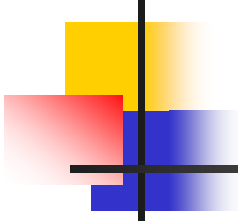
- Es un ejemplo relevante de SGBD objeto-relacional
- Vamos a ver algunos aspectos de las características objeto-relacionales de Oracle



Oracle: Definición de Tipos (I)

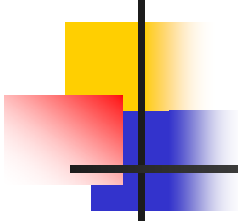
```
CREATE TYPE MiTipo AS OBJECT  
(lista de atributos y métodos);
```

```
CREATE TYPE TipoPunto AS OBJECT  
(x NUMBER,  
y NUMBER);
```



Oracle: Definición de Tipos (II)

```
CREATE TYPE TipoTriangulo AS OBJECT  
(vertice1 TipoPunto,  
 vertice2 TipoPunto,  
 vertice3 TipoPunto);
```

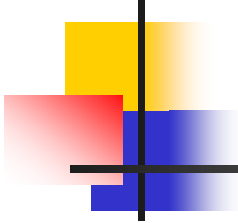


Oracle: Definición de Tipos (III)

Colecciones de objetos

```
CREATE TABLE Triangulos  
(idTriangulo INT,  
  triangulo TipoTriangulo);
```

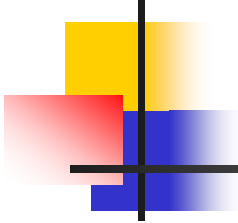
Tipos de columna



Oracle: Definición de Tipos (IV)

Inserción de objetos

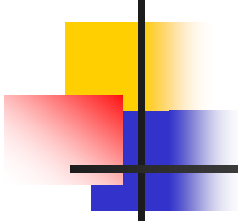
```
INSERT INTO Triangulos  
VALUES (1, TipoTriangulo (  
    TipoPunto(1.0, 0.0),  
    TipoPunto(1.0, 3.0),  
    TipoPunto(4.0, 2.0)  
)  
);
```



Oracle: Definición de Tipos (V)

Eliminación de tipos

```
DROP TYPE TipoPunto;  
DROP TYPE TipoTriangulo;
```

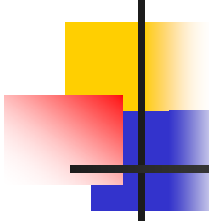


Oracle: Definición de Tipos (VI)

Asociación de métodos

```
CREATE TYPE TipoTriangulo AS OBJECT
(vertex1 TipoPunto,
vertex2 TipoPunto,
vertex3 TipoPunto
MEMBER FUNCTION perimetro (escala IN
NUMBER) RETURN NUMBER,
PRAGMA RESTRICT_REFERENCES(perimetro,
WNDS)
);
```

→ Write No Database State (para usarla en consultas)



Oracle: Definición de Tipos (VII)

Definición de métodos

```
CREATE TYPE BODY TipoTriangulo AS  
  MEMBER FUNCTION perimetro(escala NUMBER) RETURN NUMBER IS  
  BEGIN  
    ...  
  END;  
END;
```

SELF.vertice1.x, SELF.vertice1.y, ...

Utilización de métodos en consultas

```
SELECT idTriangulo, tri.triangulo.perimetro(1.0)  
FROM Triangulos tri;
```

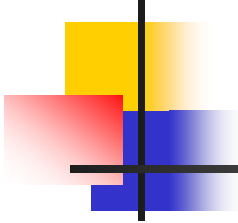


Oracle: Definición de Tipos (VIII)

Colección de objetos

CREATE TABLE Triangulos **OF** TipoTriangulo

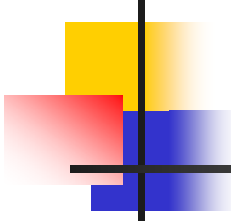
Tipos de fila



Oracle: Definición de Tipos (IX)

Utilización de REF en el DDL

```
CREATE TABLE Triangulos2 AS OBJECT  
(vertice1 REF TipoPunto,  
  vertice2 REF TipoPunto,  
  vertice3 REF TipoPunto);
```

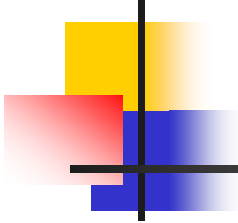


Oracle: Definición de Tipos (X)

```
CREATE TABLE Puntos OF TipoPunto
```

Crea una referencia
a partir de un valor

```
INSERT INTO Triangulos2  
  SELECT REF(p1), REF(p2), REF(p3)  
  FROM Puntos p1, Puntos p2, Puntos p3  
  WHERE ((p1.x <> p2.x) OR (p1.y <> p2.y)) AND  
         ((p2.x <> p3.x) OR (p2.y <> p3.y)) AND  
         ((p1.x <> p3.x) OR (p1.y <> p3.y));
```

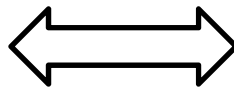


Oracle: Definición de Tipos (XI)

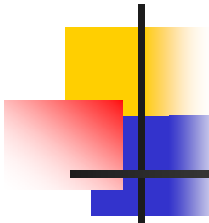
```
SELECT tri.vertice1.x, tri.vertice2.x, tri.vertice3.x  
FROM Triangulos2 tri;
```

- Para seguir una referencia, se utiliza la notación ".", de igual forma que si accediéramos directamente al valor.
- Alternativamente, se puede utilizar el operador Deref:

x.jefe.nombre



y=Deref(x.jefe), y.nombre



Oracle: Definición de Tipos (XII)

Utilización de la herencia

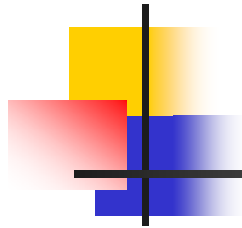
```
CREATE TYPE corporate_customer_typ_demo UNDER customer_typ  
( account_mgr_id NUMBER(6)  
);
```

Definición de una jerarquía

```
CREATE TYPE person_t AS OBJECT (name VARCHAR2(100), ssn NUMBER)  
    NOT FINAL;
```

```
CREATE TYPE employee_t UNDER person_t  
    (department_id NUMBER, salary NUMBER) NOT FINAL;
```

```
CREATE TYPE part_time_emp_t UNDER employee_t (num_hrs NUMBER);
```



Oracle: Tablas Anidadas (I)

Para poder tener una relación como tipo de un atributo...

```
CREATE TYPE TipoPoligono AS TABLE OF TipoPunto;
```

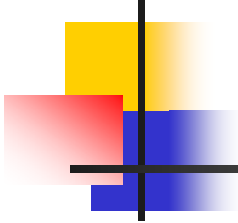
```
CREATE TABLE Poligonos (  
    id INT,  
    puntos TipoPoligono)  
NESTED TABLE puntos STORE AS TablaPuntos;
```

Tabla que almacena las
relaciones anidadas de tipo TipoPoligono



Oracle: Tablas Anidadas (II)

```
INSERT INTO Poligonos VALUES(  
    1, TipoPoligono(TipoPunto(0.0, 1.0),  
                    TipoPunto(1.0, 1.0),  
                    TipoPunto(1.0, 0.0),  
                    TipoPunto(0.0, 0.0)  
    )  
);
```



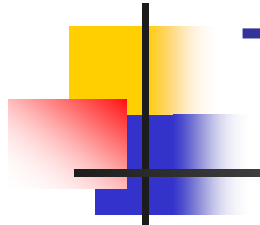
Oracle: Tablas Anidadas (III)

```
SELECT puntos FROM Poligonos  
WHERE id = 1;
```

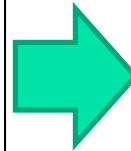
```
SELECT p.x  
FROM THE (SELECT puntos  
          FROM Poligonos  
          WHERE id = 1  
          ) p  
WHERE p.x > p.y;
```

Obtiene una
tabla anidada
concreta en la
cláusula FROM
(hay que aplicar
el THE a una
subconsulta
cuyo resultado
es una relación)

Oracle: Paso de Relaciones Tradicionales a Objetos

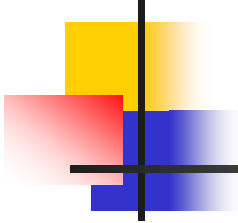


```
CREATE TABLE TrianguloPlano(  
  id INT,  
  x1 NUMBER,  
  y1 NUMBER,  
  x2 NUMBER,  
  y2 NUMBER,  
  x3 NUMBER,  
  y3 NUMBER  
);
```



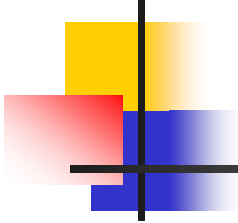
```
INSERT INTO Triangulos(  
  SELECT id,  
  TipoTriangulo(TipoPunto(x1, y1),  
                TipoPunto(x2, y2),  
                TipoPunto(x3, y3))  
  FROM TriangulosPlano;
```

-Basta utilizar los constructores de tipos adecuadamente...



Como Indicamos, Hay Otros Sistemas

- Os propongo echar un vistazo a:
 - PostgreSQL
 - Versant Object Database
 - db4objects (Versant) → open source
 - ...
- Posible tema de presentación en clase



Implicaciones del Uso de Tipos Complejos

- Almacenamiento
- Indexación
- Seguridad
- Optimización
- Gestión de transacciones
- ...