



BASES DE DATOS AVANZADAS

Tema 3, parte b

Modelo de Objetos

Objetos Puro

Univ. Cantabria – Fac. de Ciencias
Francisco Ruiz



Agradecimientos

- Este material ha sido preparado con la colaboración de:
 - Coral Calero (Univ. de Castilla-La Mancha)



Objetivos

- Presentar las diferentes propuestas **objetuales** en la tecnología de bases de datos.
- Conocer los principales **estándares** en este tópico: **SQL:2003** y **ODMG 3.0**.
- Manipular un **SGBD objeto-relacional** mediante el estándar SQL:2003.



Contenido

- Tercera Generación de BD
 - Debilidades de los SGBD Relacionales
 - Modelos de Objetos
 - Tipos de SGBD con Objetos
- Bases de Datos Objeto-Relacionales
 - Características
 - Aspectos de Objetos en SQL
 - Identidad de Objetos
 - Tipos de Datos en SQL:2003
 - Objetos Grandes
 - Tipos Definidos por el Usuario
 - Tipos Construidos
 - Módulos y Rutinas
 - Métodos
 - Jerarquías de Tablas y Vistas
 - Resumen
 - Ejemplo
 - SGBD Objeto-Relacionales
 - ORACLE
 - Comparación
- Bases de Datos Orientadas a Objetos Puras
- ODMG 3.0
 - Modelo de Objetos
 - ODL
 - OQL
- SGBD Objeto-Relacionales vs Orientados a Objetos
- Manejo de Objetos
 - Identidad
 - Persistencia



Bibliografía

- Básica
 - Piattini et al. (2006): Tecnología y Diseño de Bases de Datos.
 - Cap. 19.
 - Connolly y Begg (2005): Sistemas de Bases de Datos.
 - Caps. 27.
- Complementaria
 - Elmasri y Navathe (2007): Fundamentos de Sistemas de Bases de Datos.
 - Caps. 21.



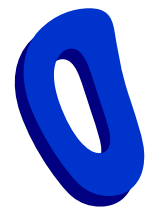
BD Orientadas a Objetos Puras

MANIFIESTO DE LOS SGBD-OO

ATKINSON et al. (1989)

Reglas de oro:

- Objetos complejos
- Identidad de objetos
- Encapsulación
- Tipos o clases
- Herencia de tipos o clases (jerarquías)
- Vinculación dinámica (sobrecarga de métodos)
- LMD computacionalmente completo
- Extensibilidad de tipos de datos





BD Orientadas a Objetos Puras

MANIFIESTO DE LOS SGBD-OO

ATKINSON et al. (1989)

Reglas de oro:

- Persistencia
- Gestión de BD grandes
- Concurrencia
- Recuperación
- Facilidad de consulta “ad hoc”



BD Orientadas a Objetos Puras

MANIFIESTO DE LOS SGBD-OO

ATKINSON et al. (1989)

Características opcionales:

- Herencia múltiple
- Verificación e inferencia de tipos
- Distribución
- Transacciones de diseño
- Versiones

Opciones abiertas a los fabricantes:

- Paradigma de programación
- Sistema de representación
- Sistema de tipos
- Uniformidad



BD Orientadas a Objetos Puras

- **MITOS sobre los SGBD-OO**
 - Son entre 10 y 1000 veces más veloces que los SGBDR.
 - Eliminan la necesidad de un LMD específico.
 - Eliminan la necesidad de las combinaciones.
 - La identidad de los objetos elimina la necesidad de claves.
 - No pueden tener un lenguaje de consulta no procedimental.
 - El procesamiento de consultas viola el encapsulamiento.
 - Son sólo una **reencarnación de los SGBD en red**.
 - No soportan versiones y transacciones de larga duración.
 - Soportan directamente los datos multimedia.
 - **No tienen un fundamento teórico.**
 - **No sirven para aplicaciones importantes.**
 - No son escalables.
 - No se usan en aplicaciones de producción.
 - No se pueden consultar.



BD Orientadas a Objetos Puras

- **Ventajas:**
 - Capacidades de modelado enriquecidas
 - Extensibilidad
 - Eliminación del desajuste de impedancia
 - Lenguaje de consulta más expresivo
 - Soporte para la evolución de esquemas
 - Soporte para transacciones de larga duración
 - Apto para aplicaciones de BD avanzadas
 - Rendimiento mejorado
- **Desventajas:**
 - Falta de un Modelo de Datos universal
 - Falta de experiencia
 - Falta de estándares
 - Competencia
 - La optimización de consultas hace peligrar la encapsulación
 - Bloqueos a nivel de objeto pueden perjudicar el rendimiento
 - Complejidad
 - Sin soporte para vistas
 - Sin soporte para seguridad



BD Orientadas a Objetos Puras: *ventajas*

- **Capacidades de modelado enriquecidas:** el modelo OO permite que el "mundo real" sea modelado mejor. El objeto, que encapsula el estado y el comportamiento, es una manera más natural y real de representar objetos del "mundo" real.
- **Extensibilidad:** permiten que se puedan definir nuevos tipos a partir de los tipos existentes.
- **Eliminación del desajuste de impedancia:** Una sola interfaz entre el LMD y el lenguaje de programación. Esto elimina de las ineficiencias que ocurrían al querer corresponder un lenguaje declarativo como SQL y un lenguaje de programación como C.
- **Lenguaje de consulta más expresivo:** el acceso navegacional es más adecuado para manejar consultas recursivas, etc ... Sin embargo, se argumenta que la mayoría de los SGBDOO están ligados a un lenguaje de prog., que en general no es demasiado fácil de usar para los usuarios finales. Por ello el estándar ODMG especifica un lenguaje declarativo basado en una versión OO de SQL.
- **Soporte para la evolución de esquemas:** El estrecho acoplamiento entre aplicaciones y datos permite que la evolución de esquemas sea más viable. La generalización y la herencia permite que los esquemas estén mejor estructurados, que sean más intuitivos, y que capturen más la semántica de la aplicación.



BD Orientadas a Objetos Puras: *ventajas*

- **Soporte para transacciones de larga duración:** los SGBD actuales obligan la serializabilidad de transacciones concurrentes, para mantener la consistencia. Eso no es útil para transacciones de larga duración, por ello algunos SGBDOO usan otros protocolos para manejar transacciones de larga duración.
- **Apto para aplicaciones de BD avanzadas:** Las capacidades de modelado enriquecida en los SGBDOO, hacen que sean adecuados para aplicaciones como CAD, CASE, multimedia, OIS, etc.
- **Rendimiento mejorado:** existen varios estudios que manifiestan la mejora en rendimiento con respecto a los SGBDR. Aunque algunos argumentan que tienen mejor rendimiento para aplicaciones más adecuadas a la OO, pero que en aplicaciones tradicionales de SGBD (OLTP) son mejores los SGBDR.



BD Orientadas a Objetos Puras: *desventajas*

- **Falta de un Modelo de Datos universal:** no existe un modelo de datos universal y la mayoría carece de fundamentos teóricos. Aunque ODMG ha propuesto un modelo de objetos que se ha convertido en el estándar *de facto* en los SGBDO
- **Falta de experiencia:** en comparación con los SGBDR el uso de los SGBDOO es todavía limitado. Por ello, no se tiene demasiada experiencia en ellos.
- **Falta de estándares:** como no existe un modelo de datos estándar, tampoco existe un lenguaje de consulta estándar. Aunque ODMG ha propuesto un lenguaje de consultas OO, que se ha convertido en *de facto*.
- **Competencia:** los SGBR y los SGBDOR están más difundidos y existen muchas más herramientas desarrolladas para ayudar a los usuarios finales.
- **La optimización de consultas hace peligrar la encapsulación:** la optimización de consultas requiere un entendimiento de la implementación subyacente para acceder a la BD eficientemente, eso puede comprometer el concepto de encapsulación.
- **Bloqueos a nivel de objeto pueden perjudicar el rendimiento:** muchos SGBDOO usan el bloqueo como la base de los protocolos de concurrencia. Sin embargo, si el bloqueo se aplica a nivel de objeto, el bloqueo de una jerarquía de herencia puede ser problemática, como así también su rendimiento.
- **Complejidad:** la gran funcionalidad que proveen los SGBDOO, hace que sean más complejos, y esto lleva a productos más difíciles de usar y más caros.
- **Sin soporte para vistas:** las vistas proveen muchas ventajas, como personalización, seguridad, reducción de la complejidad, etc.
- **Sin soporte para seguridad:** actualmente los SGBDOO no proporcionan mecanismos adecuados de seguridad.



ODMG 3.0

- **Object Database Management Group**
 - Consorcio de fabricantes de SGBD-OO para promover el uso del modelo de objetos puro en el sector de las BD.
 - Sun, POET Software, Computer Associates, ..
 - Los principales componentes de la arquitectura ODMG, versión 3.0 (de 1999) son:
 - **Modelo de Objetos** (OM)
 - **Lenguaje de Definición de Objetos** (ODL)
 - **Lenguaje de Consulta de Objetos** (OQL)
 - Lenguaje para vínculos con C++, Java y Smalltalk

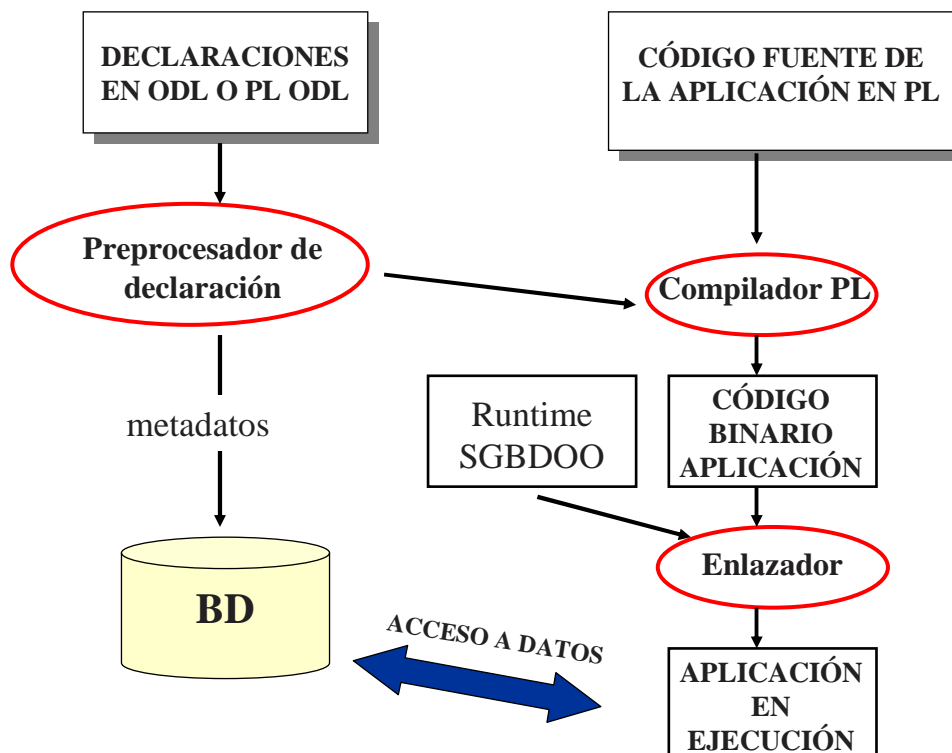


ODMG 3.0 Historia

- 1991** Reunión de vendedores
- 1993** ODMG-93
- 1995** ODMG-93 v 1.2
- 1997** ODMG v 2.0
- 1998** Revisión para JDK 1.2
- 2000** **ODMG v 3.0**



ODMG 3.0 Propuesta de Arquitectura Operativa





• Resumen de Características

- Los primitivas básicas de modelado son los **objetos** y los **literales**. Sólo los objetos tienen **OID**.
- Objetos y literales están organizados en **tipos**.
 - Todos los objetos y literales de un mismo tipo tienen un **comportamiento** y **estado común**.
 - Un tipo también es un objeto en sí mismo.
 - Un objeto es una **instancia** de su tipo.
- El **comportamiento** está definido por un conjunto de **operaciones** que se pueden realizar sobre o por el objeto.
 - Las operaciones pueden tener una lista de **parámetros** de E/S tipados y pueden retornar un resultado tipado.



• Resumen de Características

- El **estado** está definido por los **valores** que el objeto toma para un conjunto de **propiedades**.
 - Una propiedad puede ser:
 - Un **atributo** del objeto.
 - Una **interrelación** entre el objeto y otro u otros objetos.
 - Los valores de las propiedades suelen cambiar con el tiempo.
- Un Sistema de Gestión de Datos Objeto (**SGDO** / ODMS) almacena objetos de forma que pueden compartirse por usuarios y aplicaciones.
 - Un SGDO está basado en un **esquema** que está definido en el **Lenguaje de Definición de Objetos** (ODL).
 - Un SGDO contiene **instancias** de los tipos definidos por su esquema.



Modelo de Objetos – objetos y literales

- La primitiva fundamental es el **objeto**:
 - En ODMG todo son objetos
- Distingue entre **objetos** mutables e inmutables (literales).
- Los **literales** son objetos cuyo valor es constante.
 - Pueden tener estructura compleja.
 - No tienen OID.
 - No pueden ser referenciados de forma individual, sino que siempre se manejan dentro de objetos.
- Un **objeto está descrito por** cuatro características:
 - **Estructura**. Atómicos, Colecciones, Estructurados
 - **Identificador**. Único para la BD.
 - **Nombre**. También permite identificar al objeto en la BD.
 - **Tiempo de vida** (lifetime). Transitorio / Persistente.



Modelo de Objetos - tipos

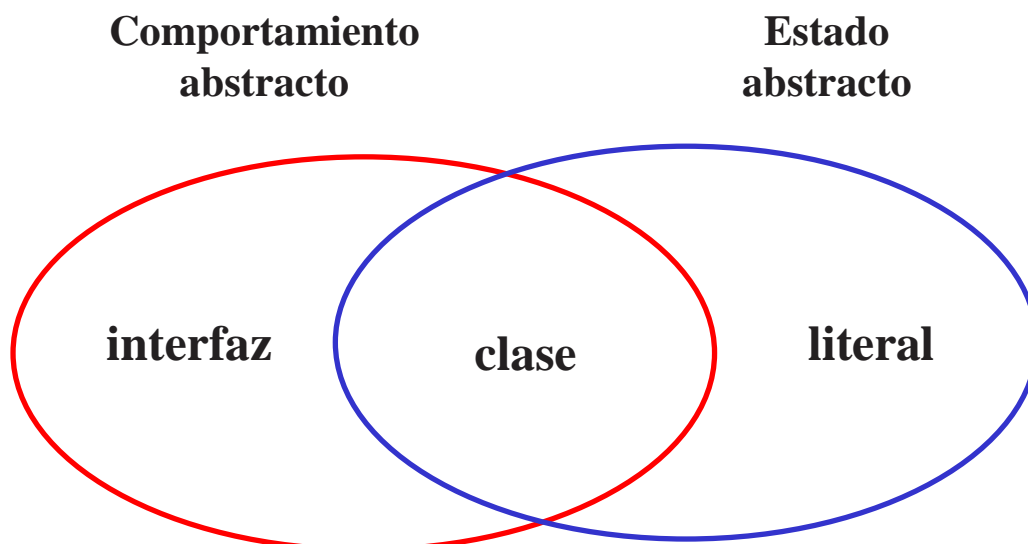
- Los objetos se clasifican en **tipos**:
 - Todos los objetos de un mismo tipo tienen unas propiedades y un comportamiento común:
 - **Comportamiento**: conjunto de **operaciones** que se pueden ejecutar sobre un objeto.
 - **Estado**: **valores** que toman sus propiedades (**atributos y relaciones**).
 - Un tipo tiene una especificación externa y una o más implementaciones.
 - La **especificación externa** es una descripción abstracta, independiente de la implementación, de los aspectos visibles del tipo: propiedades, operaciones y excepciones.
 - La **implementación** define los aspectos internos: la implementación de las operaciones y algunos otros detalles.



- En ODMG hay dos maneras de **especificar tipos de objetos**:
 - Un **interfaz** es una especificación que describe **sólo el comportamiento** abstracto de un tipo de objeto (usando firmas de operaciones).
 - Los interfaces pueden ser heredados por otros interfaces y clases.
 - Un interfaz no es instanciable (como las clases abstractas).
 - Se usan para especificar operaciones abstractas.
 - Una **clase** es una especificación que define **el comportamiento abstracto y el estado** abstracto de un tipo de objeto.
 - Son instanciables.
 - A herencia múltiple se realiza de forma indirecta.
 - **Extent** define la extensión (conjunto de todas las instancias).
 - **Key** permite establecer la clave para identificar las instancias.



Especificación de tipos





- Se puede hacer **herencia** del comportamiento o del estado:
 - Relación **ISA** (:), define la herencia simple o múltiple de **comportamiento** entre tipos de objetos (interfaces o clases).
 - GENERALIZACIÓN
 - Relación **EXTENDS** (**extend**), define la herencia simple de **estado** entre tipos de objetos (clases, no literales).
 - Se hereda tanto el comportamiento como las propiedades.



- En función de lo anterior, los tipos de objetos pueden organizarse formando **jerarquías** de **supertipos y subtipos**, de forma que:
 - Un subtipo hereda propiedades (estado) y operaciones (comportamiento) del supertipo.
 - Un subtipo puede añadir propiedades (estado) y operaciones (comportamiento) propias.
 - Un subtipo puede redefinir propiedades (estado) y operaciones (comportamiento) del supertipo.
 - Sobrecarga.
 - Se soporta herencia múltiple (sólo en comportamiento, sólo en operaciones)



- **Colecciones**

- Número variable de elementos.
- Los elementos pueden ser objetos o literales.
- Todos los elementos son del mismo tipo.

- Tipos de colecciones:

- **LIST<t>**: colección ordenada de elementos de tipo t.
- **SET<t>**: colección desordenada de elementos de tipo t que no admite duplicados.
- **BAG<t>**: colección desordenada de elementos de tipo t que admite duplicados.
- **ARRAY<t>**: vector de una dimensión y de longitud variable de elementos de tipo t.
- **DICTIONARY<t,v>**: secuencia desordenada de pares <clave, valor> sin claves duplicadas.



- **Estructuras**

- Número fijo de elementos.
- Pueden ser de distinto tipo.

- Tipos de estructuras:

- **DATE**. Fecha
- **INTERVAL**. Intervalo de tiempo
- **TIME**. Hora
- **TIMESTAMP**. Fecha y hora



- **Tipos de literales:**
 - Literales **atómicos:**
 - LONG, SHORT, UNSIGNED LONG, UNSIGNED SHORT, FLOAT, DOUBLE, BOOLEAN, OCTET, CHAR, STRING, ENUM
 - Literales **colecciones:**
 - SET, BAG, LIST, ARRAY, DICTIONARY
 - Literales **estructuras:**
 - DATE, INTERVAL, TIME, TIMESTAMP
 - Literales **NULL**



- **Object Definition Language**
 - Lenguaje para definir las especificaciones de los tipos de objetos para SGDO.
 - Equivale al DDL en sistemas relacionales.
 - Su principal objetivo es facilitar la portabilidad de esquemas.
 - Define los atributos e interrelaciones de los tipos y especifica la signatura de las operaciones.
 - No es computacionalmente completo: no incluye la implementación de las operaciones, que se deja para lenguajes de programación OO como C++ o Java.
 - La sintaxis está basada en el Idl (Interface Definition Language) de CORBA.
 - Es extensible (nuevas funcionalidades, optimizaciones físicas).



ODMG 3.0 ODL – notación BNF

<definición de tipo>::=

INTERFACE <nombre del tipo>(:<lista de supertipos>)
{ (<lista de propiedades del tipo>)
(<lista de propiedades>)
(<lista de operaciones>) }

<propiedad del tipo>::=

{ **EXTENT** <nombre de la extensión>
| **KEY(S)** <lista de claves> }

<propiedad>::=

{<especificación de atributo>
| <especificación de interrelación>
| <especificación de operación> }

<especificación del atributo>::=

ATTRIBUTE <tipo de dominio> <tamaño> <nombre del atributo>



ODMG 3.0 ODL – notación BNF

<especificación de interrelación>::=

RELATIONSHIP <destino del camino> <nombre del camino>
INVERSE <camino inverso>
[**ORDER BY** <lista de atributos>]

<especificación de operación>::=

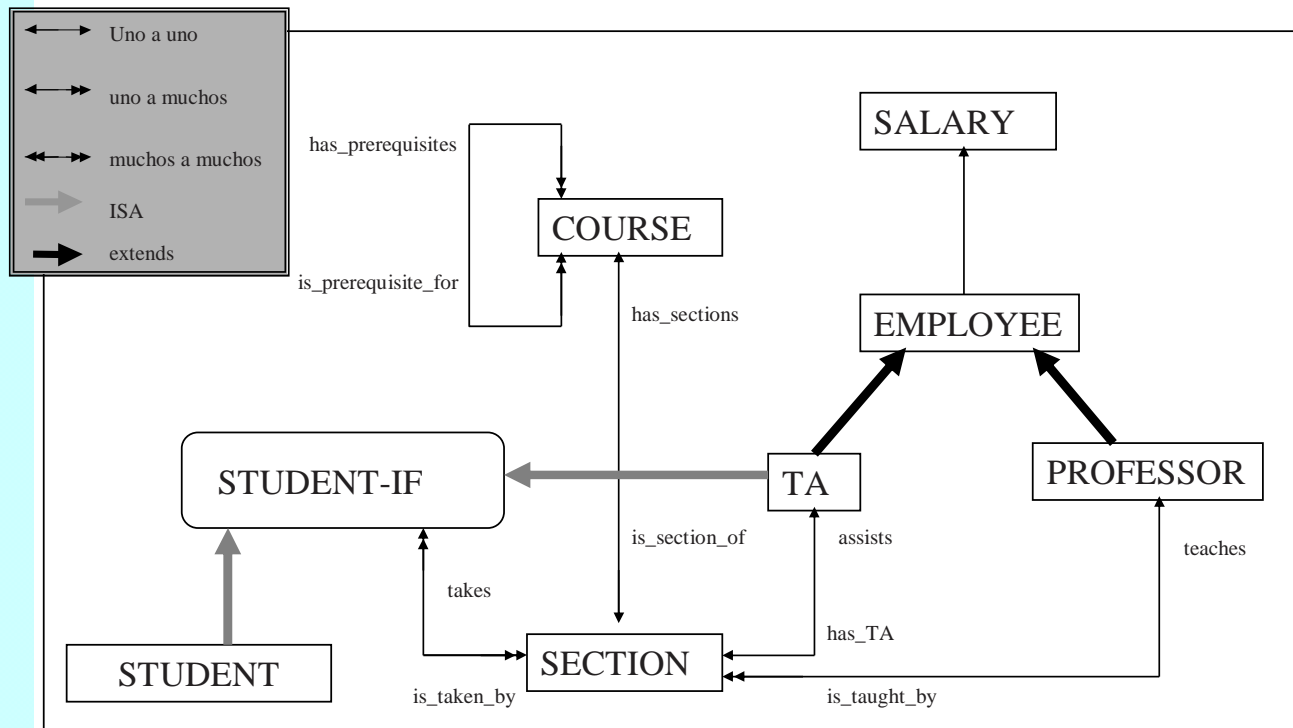
<tipo devuelto> <nombre de la operación>
(<lista de argumentos>)
[**RAISES** (<lista de excepciones>)]

<argumento>::=

<rol> [<nombre del argumento>:] <tipo del argumento>



ODMG 3.0 ODL – ejemplo



Francisco Ruiz - BDA

3a.31



ODMG 3.0 ODL – ejemplo

```

class Course (extent courses)
{
  attribute string name;
  attribute string number;

  relationship list<Section> has_section
    inverse Section::is_section_of;
  relationship set<Course> has_prerequisites
    inverse Course::is_prerequisite_for;
  relationship set<Course> is_prerequisite_for
    inverse Course::has_prerequisites;

  boolean offer (in unsigned short semester)
    raises (alredy_offered);
  boolean drop (in unsigned short semester)
    raises (not_offered)};
  
```

Francisco Ruiz - BDA

3a.32



ODMG 3.0 ODL – ejemplo

```
class Section (extent sections)
{
attribute string number;

relationship Professor is_taught_by
    inverse Professor::teaches;
relationship TA has_TA
    inverse TA::assists;
relationship Course is_section_of
    inverse
    Course::has_sections;
relationship set<Student>
    is_taken_by
    inverse Student::takes;
};
```

```
class Salary
{
attribute float base;
attribute float overtime;
attribute float bonus;
};
```

```
class Employee (extent employees)
{
attribute string name;
attribute short id;
attribute Salary annual_salary;

void hire ();
void fire()
    raises (no_such_employee)
};
```



ODMG 3.0 ODL – ejemplo

```
class Professor extends Employee (extent professors)
{
attribute enum Rank {full, associate, assistant} rank;

relationship set<Section> teaches
    inverse Section::is_taught_by;

short grant_tenure()
    raises (ineligible_for_tenure));
```



```
interface Student-IF
{struct Address {string college, string room_number}
attribute string name;
attribute string student_id;
attribute Address dorm_address;

relationship set<Section> takes
    inverse Section::is_taken_by;

boolean register_for_course (in unsigned short course,
    in unsigned short Section)
    raises (unsatisfied_prerequisites, section_full, course_full);
boolean drop_course (in unsigned short course)
    raises (not_registered_for_that_course)
void assign_major (in unsigned short Department);
short transfer (in unsigned short old_section,
    in unsigned short new_section)
    raises (section_full, not_registered_in_section)};
```



```
class TA extends Employee:Student-IF
    (extent Employee)
{
relationship Section assists
    inverse Section::has_TA;

attribute string name;
attribute string student_id;
attribute Address dorm_address;

relationship set<Section> takes
    inverse Section::is_taken_by;
};
```

```
class Student:Student-IF
    (extent Students)
{
attribute string name;
attribute string student_id;
attribute Address dorm_address;

relationship set<Section> takes
    inverse Section::is_taken_by;
};
```



- **Object Query Language**

- Provee **acceso declarativo** a una BD de objetos mediante una **sintaxis similar a SQL**.
- **No provee operadores de modificación** explícitos porque esta funcionalidad se deja para las operaciones definidas sobre los objetos.
- Puede ser usado como un **lenguaje autónomo o embebido** dentro de otros lenguajes (C++, Smalltalk, Java).
- Desde OQL se pueden invocar operaciones escritas en estos lenguajes.
- Permite **acceso** tanto **asociativo** como **navegacional**:
 - Una consulta **asociativa** devuelve una colección de objetos.
 - Una consulta **navegacional** accede a objetos individuales y las interrelaciones entre objetos sirven para navegar entre objetos.



- El formato del **SELECT** es similar al de SQL:

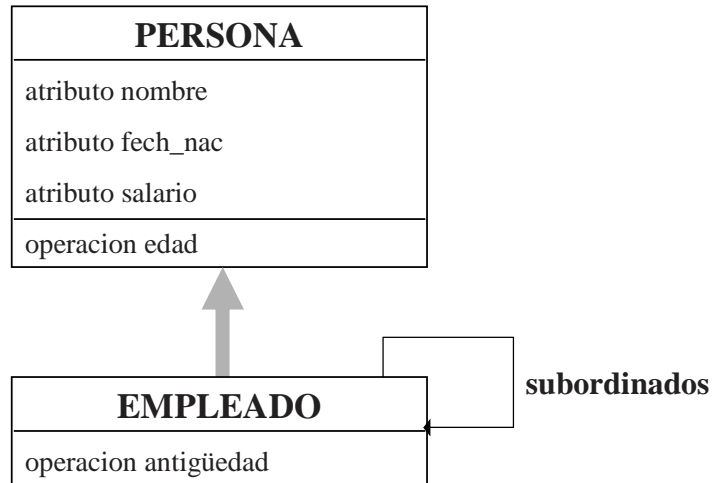
```
SELECT [DISTINCT] <expresión>  
FROM <lista from>  
[WHERE <expresión> ]  
[GROUP BY <atributo1:expresión1, .....>]  
[HAVING <predicado>]  
[ORDER BY <expresión>]
```

Donde

```
<lista from> ::=  
{ <nombre de variable> IN <expresión> |  
<nombre de variable> IN <expresión> , <lista from> |  
<expresión> AS <nombre de variable> |  
<expresión> AS <nombre de variable> , <lista from> }
```



ODMG 3.0 OQL - ejemplo



ODMG 3.0 OQL - ejemplo

```
select distinct x.edad
from x in personas
where x.nombre="Ana"
```

← ¿Edades de todas las personas llamadas "Ana"?

Devuelve un literal del tipo set <integer>

¿Edad y salario de todas las personas llamadas "Ana"?



```
select distinct struct (a:x.edad, s:x.salario)
from x in personas
where x.nombre="Ana"
```

Devuelve un literal del tipo set <struct (a:integer, s:integer)>



ODMG 3.0
OQL - ejemplo

¿Nombre de cada empleado y sus subordinados que tienen un salario mayor de 3000 €?



```
select distinct struct (a:x.nombre, smp:
    (select y
    from y in x.subordinados
    where y.salario>300000)
from x in empleados
```

Devuelve un literal del tipo set <struct (nombre:string, smp:bag<empleado>>>



ODMG 3.0
OQL - ejemplo

Utilización del select dentro de la cláusula from:

```
select struct (a:x.edad, s:x.salario)
from x in
    (select y
    from y in empleados
    where y.antigüedad=10)
```



¿Edad y salario de todos los empleados con justo 10 años de antigüedad?

Devuelve un literal del tipo bag <struct (a:integer, s:integer)>

Efecto del uso de DISTINCT:

SELECT -----> **bag** (con repetición)

SELECT DISTINCT -----> **set** (sin repetición)



- Creación de Objetos:
 - **Objetos mutables:**
 - Para crear un objeto con identificador se debe utilizar un tipo constructor.
`Persona (nombre: "María", fech_nac:"11/2/69", salario:1000)`
 - Si no se inicializa alguna de sus propiedades, se les asignará automáticamente un valor por defecto.
 - **Literales (objetos inmutables):**
 - Para construir un objeto sin identificador se utiliza el constructor **struct**.
`STRUCT (a:10, b:"María")`



- También es posible **crear objetos** mutables (no literales) formados por el **resultado de una consulta**.

```
type bolsaint: bag<integer>;
type estado attributes
  a: integer
  b: salario
end_type;
type estados: bag<estado>;
```

```
bolsaint (select distinct x.edad
from x in personas
where nombre="Pedro")
```

Devuelve un objeto mutable de tipo "bolsaint"

```
estados (select estado (a:x.edad, b:x.salario)
from x in personas
where nombre="Pedro")
```

Devuelve un objeto mutable de tipo "estados"



- Es posible **dar nombre al resultado de una consulta** y utilizarlo en otras consultas posteriores.

define juanes as

```
select distinct x
  from x in persona
  where x.nombre="Juan";
```



¿Conjunto de personas llamadas "Juan"?

```
select distinct x.salario
  from x in juanes;
```



¿Salarios de las personas llamadas "Juan"?



OPERADORES DE ACCESO:

“.” / “->” (aplicados a un atributo, una operación o una relación)

FIRST / LAST (primero / último elemento de una lista o un vector)

Ejemplo:

personas.nombre
personas->nombre

nombre de todas las personas

juan.subordinado.nombre
juan->subordinado->nombre

nombre de los subordinados de Juan

juan.edad
juan->edad

edad de juan



- **EXPRESIONES CON COLECCIONES:**

COUNT , SUM , MIN , MAX , AVG (funciones de agrupamiento)

GROUP...IN...BY...WHERE

SORT...IN...BY

FOR...ALL...IN

EXIST...IN

IN

SELECT...FROM...WHERE

- **Ejemplos:**

COUNT(personas)

FOR ALL e IN empleados:e.salario>3000

- **EXPRESIONES ARITMÉTICAS:**

+ , - , * , / , - (unario) , MOD , ABS

- **Ejemplo:**

COUNT(personas) – COUNT(empleados)



- **OPERADORES DE COMPARACIÓN:**

= , != , < , <= , > , >=

- **OPERADORES LÓGICOS:**

NOT , AND , OR

- **Ejemplo:**

NOT(true)

- **EXPRESIONES SOBRE COLECCIONES:**

INTERSEC , UNION , EXCEPT

- **Ejemplo:**

bag(2,2,3,3,3) EXCEPT bag(2,3,3,3)

el resultado es bag(2)



- Se pueden hacer **combinaciones (joins)** de forma parecida a SQL:

```
Select p
  from Persons p, Flowers f
 where p.nombre = f.nombre;
```

← ¿Conjunto de personas que tienen nombre de flor?

- **Conversores de Tipos:**

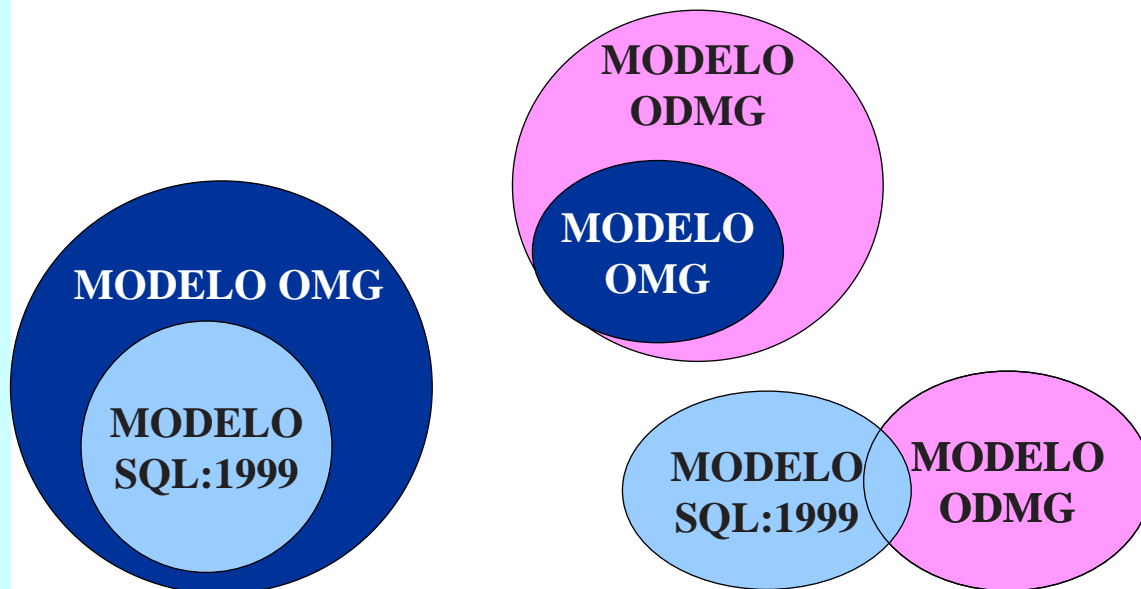
- LISTTOSET , ELEMENT , FLATTEN

- Ejemplo:

```
define juan as element(
  select distinct x
  from x in empleado
  where x.nombre="Juan");
```



Comparativa de los Modelos del ODMG y el SQL

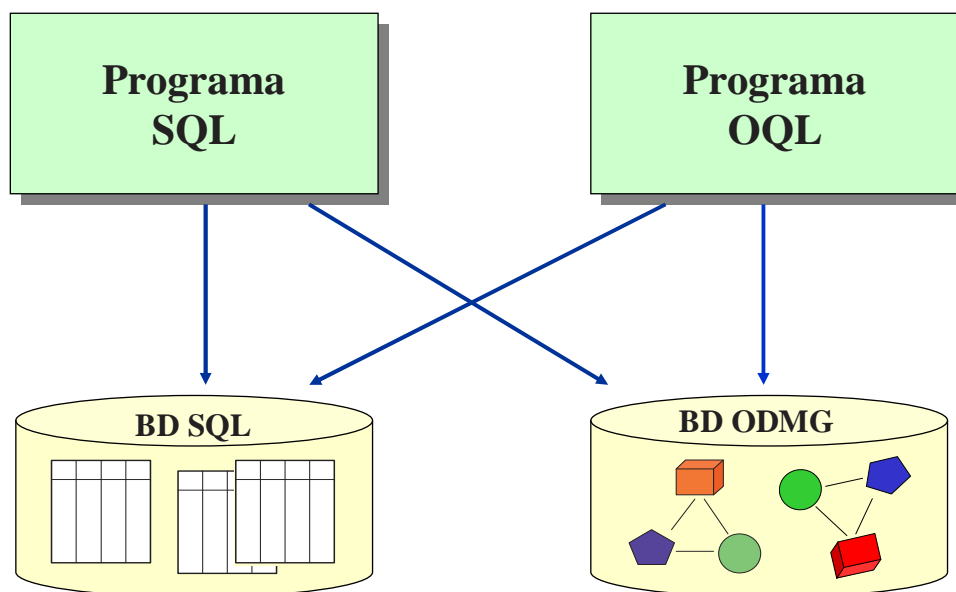




Principal Inconveniente Práctico del ODMG

*“LOS PRINCIPALES PROBLEMAS Y DEFICIENCIAS EN EL LENGUAJE DE BASES DE DATOS DEL ODMG SE DEBEN AL HECHO DE QUE **NO INCLUYE LAS FACILIDADES DEL SQL** (A PESAR DEL HECHO DE QUE EL MODELO DE DATOS DEL ODMG ESTÁ BASADO EN EL MODELO BÁSICO DEL OMG, QUE SÍ INCLUYE TOTALMENTE EL MODELO RELACIONAL)”*

Kim (1994)





SGBD Objeto-Relacionales vs OO Comparativa – modelado de datos

Característica	Objeto-Relacional (SQL)	Orientado a Objetos Puro (ODMG)
Identidad de objetos (OID)	Tipo REF	SÍ
Encapsulación	a través de UDTs	SÍ pero rota por consultas
Herencia	SÍ (jerarquías separadas de UDTs y tablas)	SÍ
Polimorfismo	SÍ (funciones genéricas)	SÍ (como en un lenguaje de programación OO)
Objetos complejos	SÍ (UDTs)	SÍ
Interrelaciones	SÍ (soporte fuerte con restricciones de integridad referencial)	SÍ (por ejemplo, usando bibliotecas de clases)



SGBD Objeto-Relacionales vs OO Comparativa – acceso a los datos

Característica	Objeto-Relacional (SQL)	Orientado a Objetos Puro (ODMG)
Creación y acceso a datos persistentes	SÍ pero no transparente	SÍ, pero el grado de transparencia depende del producto concreto
Facilidad de consulta "ad hoc"	SÍ, soporte fuerte	a través de ODMG
Navegación	SÍ (tipo REF)	SÍ, soporte fuerte
Restricciones de integridad	SÍ, soporte fuerte	NO
Servidor de Objetos/ Servidor de Páginas	Servidor de Objetos	Ninguno
Evolución de Esquemas	SÍ, pero con limitaciones	SÍ, pero el grado de soporte depende del producto concreto



Comparativa – compartición de datos

Característica	Objeto-Relacional (SQL)	Orientado a Objetos Puro (ODMG)
Transacciones ACID	SÍ, soporte fuerte	SÍ
Recuperación	SÍ, soporte fuerte	SÍ, pero el grado de soporte depende del producto concreto
Modelos de transacciones avanzados	NO	SÍ, pero el grado de soporte depende del producto concreto
Seguridad, integridad y vistas	SÍ, soporte fuerte	SÍ, pero con limitaciones



Manejo de Objetos Identidad

- Una parte clave de la definición de un objeto es su **identidad única**. En un SGBD-OO a cada objeto se le asigna un **Identificador de Objeto (OID)** cuando es creado.
- El **OID** es
 - **Generado** por el sistema.
 - **Único** para cada objeto y único en el sistema.
 - Mecanismo equivalente a la integridad de entidades.
 - **Invariante**, es decir, no cambia a lo largo de la vida del objeto.
 - No será utilizado por ningún otro objeto, incluso después de que el objeto sea eliminado.
 - **Independiente** del estado (valores de los atributos).
 - Dos objetos pueden tener el mismo estado pero no el mismo OID.
 - **Invisible** para el usuario (idealmente, no se suele cumplir).



Ventajas de usar OIDs como mecanismo de identidad de objetos:

- **Eficiencia:** requieren mínimo almacenamiento
- **Rapidez:** los OIDs apuntan a una dirección real o a una ubicación dentro de una tabla que da la dirección del objeto referenciado. Esto permite que los objetos sean localizados rápidamente.
- **No pueden ser modificados por el usuario:** los OIDs son generados por el usuario y se mantienen invisibles, o al menos de solo lectura. El sistema puede asegurar la integridad referencial más fácilmente. Esto evita que el usuario mantenga la integridad.
- **Son independientes del contenido:** Si los valores de los atributos cambian para un objeto, el objeto sigue siendo el mismo, mantiene su OID.



- Mecanismos para **implementar la persistencia** en SGBDOO:
 - **Puntos de comprobación** (checkpointing).
 - El sistema copia todo o parte del espacio de direcciones de trabajo a disco.
 - **Serialización.**
 - Copiar a disco el cierre de una estructura de datos.
 - **Paginación explícita.**
 - El programador indica de forma explícita los objetos que quiere que se graben en disco de forma obligatoria e inmediata por el sistema.
- **Persistencia Ortogonal**, basada en tres principios
 - **Independencia:** la persistencia de un objeto es independiente de cómo el programa lo manipula y al contrario.
 - **Ortogonalidad** respecto a los tipos: Todos los objetos deben poder ser persistentes sin depender de su tipo.
 - **Transitividad:** La manera de identificar y proveer persistencia no depende de los tipos de datos del lenguaje.



- Para hacer **búsquedas eficientes** en sistemas de objetos es necesario utilizar técnicas de **indexación**.
 - Los aspectos a considerar son:
 - La definición de las clases y las colecciones.
 - Las características del lenguaje de consulta.
 - Predicados anidados.
 - Herencia.
 - Servicios.
 - Las estrategias de recorrido en las clases.
 - Las técnicas de recuperación de objetos.
 - Para facilitar la comparación, recuperación y actualización se han propuesto varias **estructuras de índices**:
 - **Anidados** (NESTED-INDEX)
 - **Caminos** (PATH-INDEX)
 - **Multi-índices** (MULTI-INDEX)