

DISEÑO DE BASES DE DATOS

UNIDAD III: MODELO DE DATOS ORIENTADO A OBJETOS



Departamento Ingeniería de Sistemas
Facultad de Informática
Universidad Nacional del Comahue



Indice

- BDs Orientadas a Objeto
- Persistencia
- ODL y OQL
- BDs Objeto Relacionales
- Comparación de los tipos de SGBD

Bases de Datos OO

- La información se representa mediante **objetos** como los presentes en la programación OO
- Se diseñaron para trabajar en conjunción con **lenguajes de programación OO** (como Java) ya que usan el mismo modelo que estos lenguajes
- Son una buena elección para aquellos sistemas que necesitan un *buen rendimiento* en la manipulación de tipos de datos complejos

Bases de Datos OO

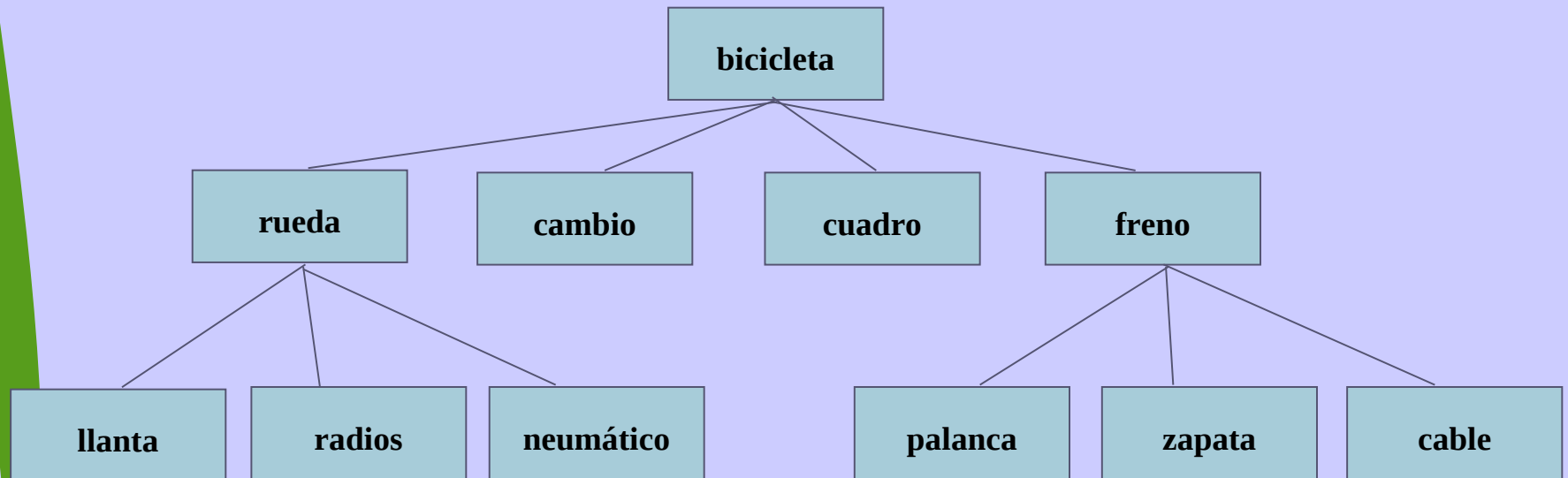
- Proporcionan los costos de desarrollo más bajos
- **Buen rendimiento:**
 - cuando se usan objetos gracias a que almacenan objetos en disco
 - tienen una integración transparente con el programa escrito en un lenguaje de programación OO ya que almacenan exactamente el modelo de objetos usado a nivel de aplicación reduciendo los costos de desarrollo y mantenimiento

Bases de Datos OO

- Los principales conceptos que se utilizan en las Bases de Datos OO son:
 - Encapsulamiento
 - Clases – Objetos...
 - Polimorfismo
 - Herencia
 - **Manejo de objetos complejos**
 - **Identidad de objetos**
 - **Lenguajes de programación OO**



Objetos Complejos



Objetos Complejos

- Objetos que contienen a otros objetos.
- Forman una Jerarquía de continentes.
 - Por ejemplo: Clase *bicicleta*:
 - *Atributos*: marca, color, modelo, tipo_diseño.
 - *Referencias a objetos*: de las clases rueda, freno, cambio y cuadro.
- Un objeto puede estar incluido en varios objetos. Esto se representa mediante un GAD.

Identidad de los Objetos

- En OO no hace falta que el usuario proporcione un identificador. Cada objeto recibe del sistema un identificador en el momento en que se crea llamado **OID (object identifier)**
- Los **identificadores de objeto (OIDs)**:
 - Son únicos e **inmutables** (no cambian-no se reasignan)
 - No son necesariamente fáciles de recordar
 - Hay que tener cuidado cuando se traduce a otro tipo de sistema, por ejemplo relacional. Hay atributos que por si solos ya identifican unívocamente un objeto

Lenguajes OO

■ Lenguajes de Programación Persistentes

- Trabajan con datos persistentes, es decir, los datos siguen existiendo una vez que el programa que los creó ha concluido.
- Son lenguajes de programación extendidos con constructores para el tratamiento de datos persistentes
- El lenguaje de consulta se encuentra totalmente integrado con el lenguaje anfitrión

Lenguajes OO

■ Lenguajes de Programación Persistentes

- Los objetos se pueden crear y guardar en la BD sin ningún cambio
- No se debe escribir código explícito para buscarlos en memoria o volver a guardarlos en el disco

Lenguajes OO

■ Lenguajes de Programación Persistentes

- **Inconvenientes:**

- Son potentes por lo que resulta sencillo cometer errores que dañen la BD
- Son complejos por lo que la optimización automática de alto nivel (por ej. Reducción de E/S a disco) es mas difícil

Persistencia de los Objetos

■ **Persistencia por clases**

- Declaro que una clase es persistente
- Todos los objetos de la clase son persistentes de manera predeterminada

■ **Persistencia por creación**

- Extensión de la sintaxis para la creación de objetos transitorios
- Los objetos de una clase son transitorios o persistentes en función de la manera de crearlos
- Es un enfoque muy usado

Persistencia de los Objetos

■ **Persistencia por marcas**

- Es una variante del caso anterior
- Todos los objetos son transitorios
- Los objetos se marcan como persistentes después de haberlos creado para que existan mas allá de la ejecución del programa
- Originalmente se crean como no persistentes

Persistencia de los Objetos

■ **Persistencia por referencia**

- Uno o varios objetos se declaran persistentes (objeto raíz) de manera explícita
- Un objeto será persistente si (y solo si) se hace referencia a él de manera directa o indirecta desde un objeto persistente (objeto raíz)
- Genera que sean persistentes estructuras de datos completas
- Puede resultar costoso debido a la cantidad de referencias

Acceso a Objetos Persistentes

- **¿Cómo encontrar los objetos de la BD?**

- **Tres enfoques:**

- *Enfoque 1:*

- Dar nombre a los objetos (como se hace con los archivos)
 - Resulta impráctico cuando tenemos un número muy grande de objetos

Acceso a Objetos Persistentes

■ **¿Cómo encontrar los objetos de la BD?**

- *Enfoque 2:*

- Usar los punteros persistentes de los objetos
- Generalmente son difíciles de recordar
- Los genera el sistema

Acceso a Objetos Persistentes

■ ¿Cómo encontrar los objetos de la BD?

- *Enfoque 3:*

- Guardar colecciones de objetos y permitir la iteración sobre ellas (Ej. listas, arreglos, etc.)
- Caso especial: **extensiones de clases**
 - Permiten examinar todos los objetos de una clase. Son como las tuplas de una relación

Extensiones de Clases

- En la realidad, generalmente solo se da nombre a las **extensiones de las clases** y a los de tipo colección
- Las **extensiones** poseen los objetos persistentes de cada clase

Claves y Extensiones

- Una clase con una **extensión** puede tener una o mas **claves**
- Una **clave** posee una o más propiedades (atributos y asociaciones) cuyos valores están restringidos a ser **únicos por cada objeto** en la **extensión**

Comparación de DBO y DBR

■ Relaciones:

- DBO: Por medio de referencias a objetos de otras clases (*referencias OIDs*). Unidireccional o bidireccional (mediante inversas generando integridad referencial). Solo Binarias.
- DBR: Por medio de valores de atributos (*claves foráneas*). N-arias.

■ Claves:

- DBO: No son necesarias. Existen los OIDs
- DBR: Son la base del modelo relacional.

Comparación de DBO y DBR

■ Herencia:

- DBO: Se heredan todas las propiedades (atributos, relaciones y operaciones). No puede ser solapada.
- DBR: Se utiliza la restricción de clave primaria y foránea.

■ Operaciones:

- DBO: Se especifican durante el diseño de la BD.
- DBR: Se especifican en etapas siguientes.

Correspondencia MER con UML

MER	UML
Entidad	Clase
Atributo	Atributo
Relación binaria	Asociación
Relación con atributos	Clase Asociación
Generalización/Especialización	SuperClase/SubClase
Relación n-aria	Clase separada

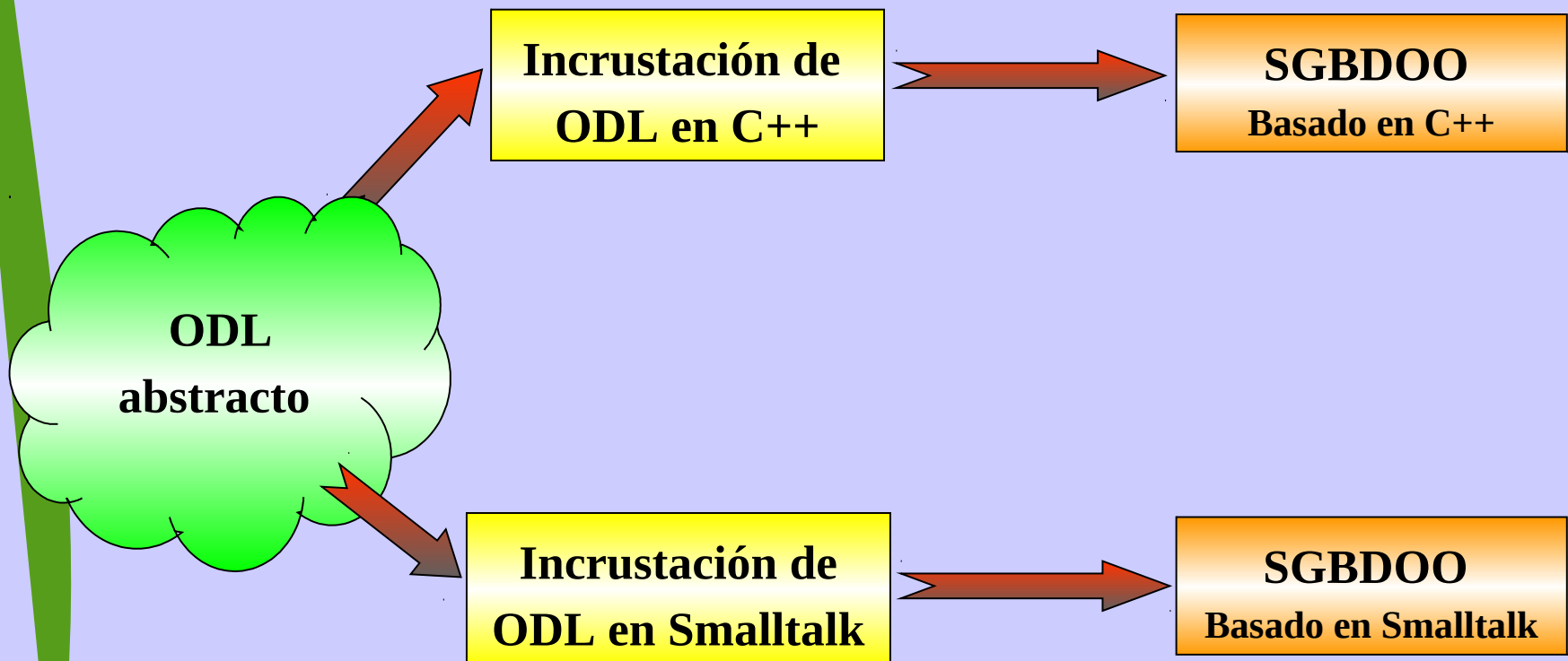
Lenguajes para BDOO

- Se definen dos lenguajes
 - **ODL (Object Definition Language):**
Lenguaje de Definición de Objetos
 - **OQL (Object Query Language):**
Lenguaje de Consulta de Objetos

ODL

- Es un lenguaje estándar para especificar la estructura de las BDOO
- Permite escribir el diseño de una BD y traducirla directamente a declaraciones de un Sistema de Administración de BDOO
- No es un lenguaje de programación. Generalmente se traduce a C++ o Smalltalk
- Es el equivalente al DDL de BDR

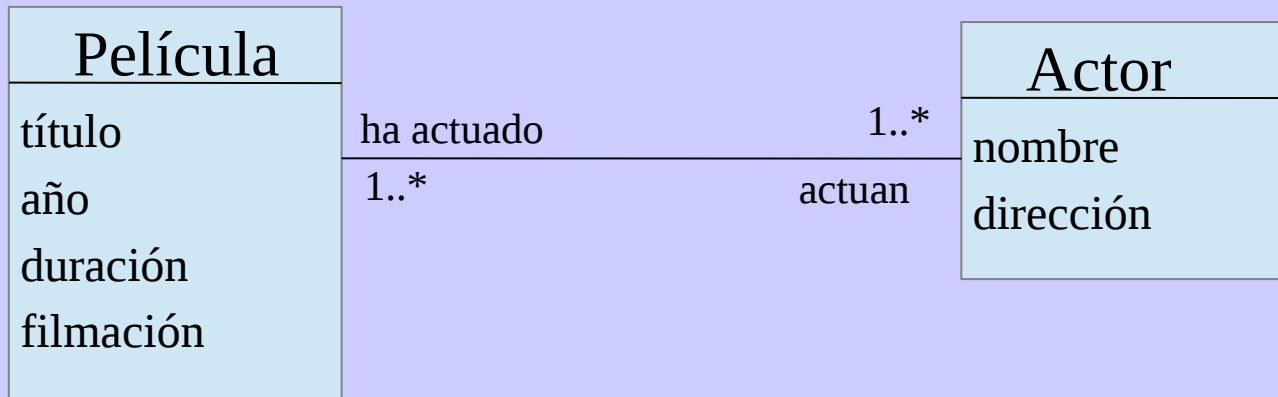
ODL



ODL

- En ODL tenemos tres tipos de propiedades:
 - **Atributos**: son las propiedades cuyos tipos son primitivos
 - **Relaciones**: son propiedades cuyos tipos son referencias a objetos de otras clases
 - **Métodos**: son funciones que pueden aplicarse a los objetos de la clase

ODL: Base de Datos OO

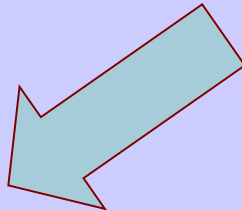


ODL: Atributos

■ Atributos

Atributos de la clase Película

class Pelicula{



**attribute string titulo;
attribute integer año;
attribute integer duración;
attribute enum filmación (color, blancoNegro) tipoDeFilm;**

};

Objeto Pelicula (“El Código Enigma”, 2014, 114, color)

ODL: Relaciones

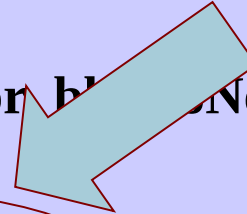
■ Relaciones

relationship Set <Actor> actores;

Ahora tenemos

```
class Pelicula{  
  attribute string titulo;  
  attribute integer año;  
  attribute integer duración;  
  attribute enum filmación (color blanco y Negro) tipoDeFilm;  
  
  relationship Set <Actor> actuan;  
  
};
```

Todos los objetos de la clase Película contienen un conjunto de referencias a Objetos de la clase Actor



ODL: Inversas

■ Relaciones Inversas

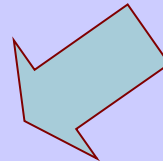
- Para conocer no solo los actores de una película sino también las películas en las cuales ha actuado un actor.

- **relationship Set <Pelicula> haActuado;**

Ahora tenemos

```
class Actor{  
    attribute string nombre;  
    attribute string dirección;
```

La relación haActuado es la inversa a
La relación actuan de la clase Película



```
    relationship Set <Pelicula> haActuado  
        inverse Pelicula::actuan;
```

```
};
```

ODL: Ejemplo

```
class Pelicula{  
  attribute string titulo;  
  attribute integer año;  
  attribute integer duración;  
  attribute enum filmación (color, blancoNegro) tipoDeFilm;  
  relationship Set <Actor> actuan  
    inverse Actor::haActuado;  
};
```

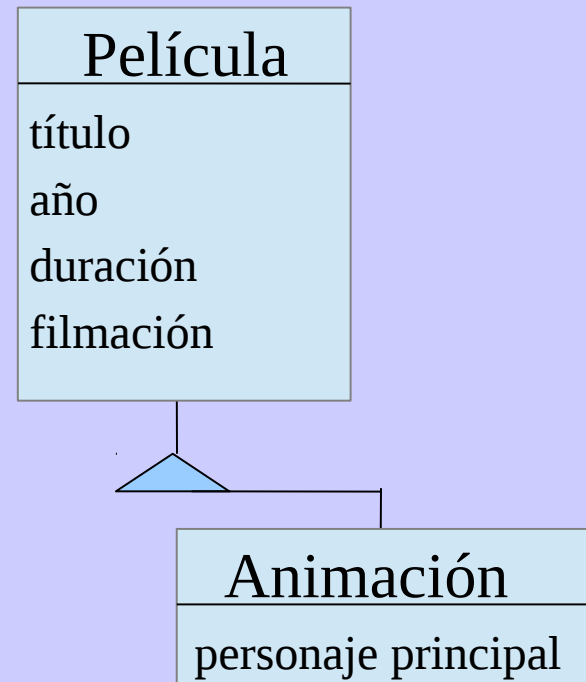
Analizamos
bidireccionalidad?

```
class Actor{  
  attribute string nombre;  
  attribute string dirección;  
  relationship Set <Pelicula> haActuado  
    inverse Pelicula::actuan;  
};
```

ODL: Herencia

- Son iguales a las subclases de OO
- Se indican con un “**extends**”

```
class Animacion extends Pelicula  
{  
    .....  
};
```



ODL: Claves

- Se pueden definir claves, aunque es opcional (se pueden usar los OIDs)
- Las claves pueden ser *atributos*, *relaciones* e incluso *métodos*.
- Solo garantizan que para distintos objetos de una clase, la clave retorne valores diferentes.

ODL: Claves

- Después de la declaración de clases colocar:
(key <lista-de-claves>)

```
class Pelicula (key (titulo, año))  
{  
    attribute string titulo;  
    attribute integer año;  
    .....;  
};
```

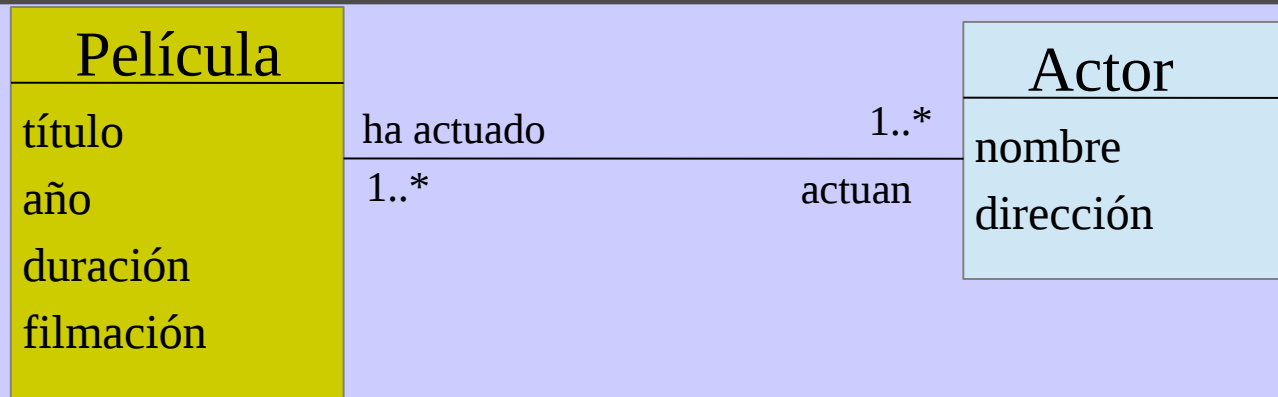
Cuando es compuesta,
colocar paréntesis

ODL: Extensiones de Clases

- Indica el conjunto de objetos de una clase
- Generalmente se utiliza el mismo nombre de la clase en plural
- Después de la declaración de clases colocar:
(extent <nombre>)

```
class Pelicula (extent Peliculas)  
{  
    .....  
};
```

ODL: Usando Todo Junto



```
class Pelicula (extent Peliculas key (titulo, año))  
{  
  attribute string titulo;  
  attribute integer año;  
  attribute integer duración;  
  attribute enum filmación (color, blancoNegro) tipoDeFilm;  
  relationship Set<Actor> actuan inverse Actor::haActuado;  
};
```

OQL

- Es un lenguaje de Consulta similar a **SQL** para OO
- Permite expresar consultas a un nivel mas alto que un lenguaje de programación
 - **SQL-3** aporta al mundo relacional lo mejor de OO
 - **OQL** aporta al mundo OO lo mejor de SQL
- En OO los objetos se administran mediante las consultas OQL y sentencias del lenguaje de programación anfitrión

OQL: Estructura

- Si a es un objeto que pertenece a la clase C y si p es alguna propiedad (un atributo o relación) $a.p$ indica el resultado de aplicar p a a , es decir:
 - Si p es un atributo $a.p$ será su valor en el objeto a
 - Si p es una relación $a.p$ será el objeto o colección de objetos relacionados con a por medio de la relación p
- La sintaxis es muy parecida a SQL

***SELECT** <lista-de-valores>
FROM <colecciones> o <extents>
WHERE <condicion>*

OQL: Ejemplo

- Una consulta en que involucra una sola clase en OQL:

```
SELECT p.año  
FROM p IN Peliculas  
WHERE p.titulo = “El Código Enigma”
```

- Se traduce en:

Por cada **p** en **Peliculas** hacer

si **p.titulo** = “El Código Enigma” entonces
adicionar **p.año** a la salida

Película
título
año
duración
filmación

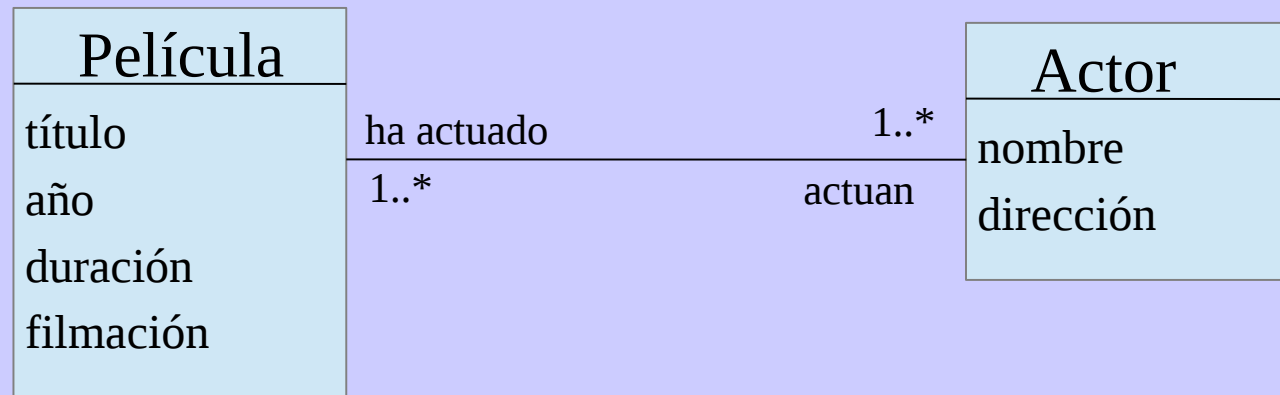
OQL: Ejemplo Join

- Un join se define como:

SELECT *a.nombre*

FROM *p* **IN** *Peliculas*, *a* **IN** *p.actuan*

WHERE *p.titulo* = “El Código Enigma”



OQL: Uso ilegal del “.”

- Hay que tener cuidado al comparar valores ya que podemos generar expresiones ilegales
- No podemos comparar una colección con un único valor
- La siguiente es una *expresión ilegal* para el WHERE ya que *p.actores.nombre* es una colección de objetos, NO un único objeto:

SELECT p.nombre

FROM p IN Peliculas

*WHERE p.titulo=“El Código Enigma” AND
p.actu**in.nombre=‘Allen Leech’*

OQL: Uso legal del “.”

- Esas colecciones deben ser parte del FROM y ser tratadas como los extent

SELECT p.nombre

FROM p IN Peliculas, a IN p.actuan

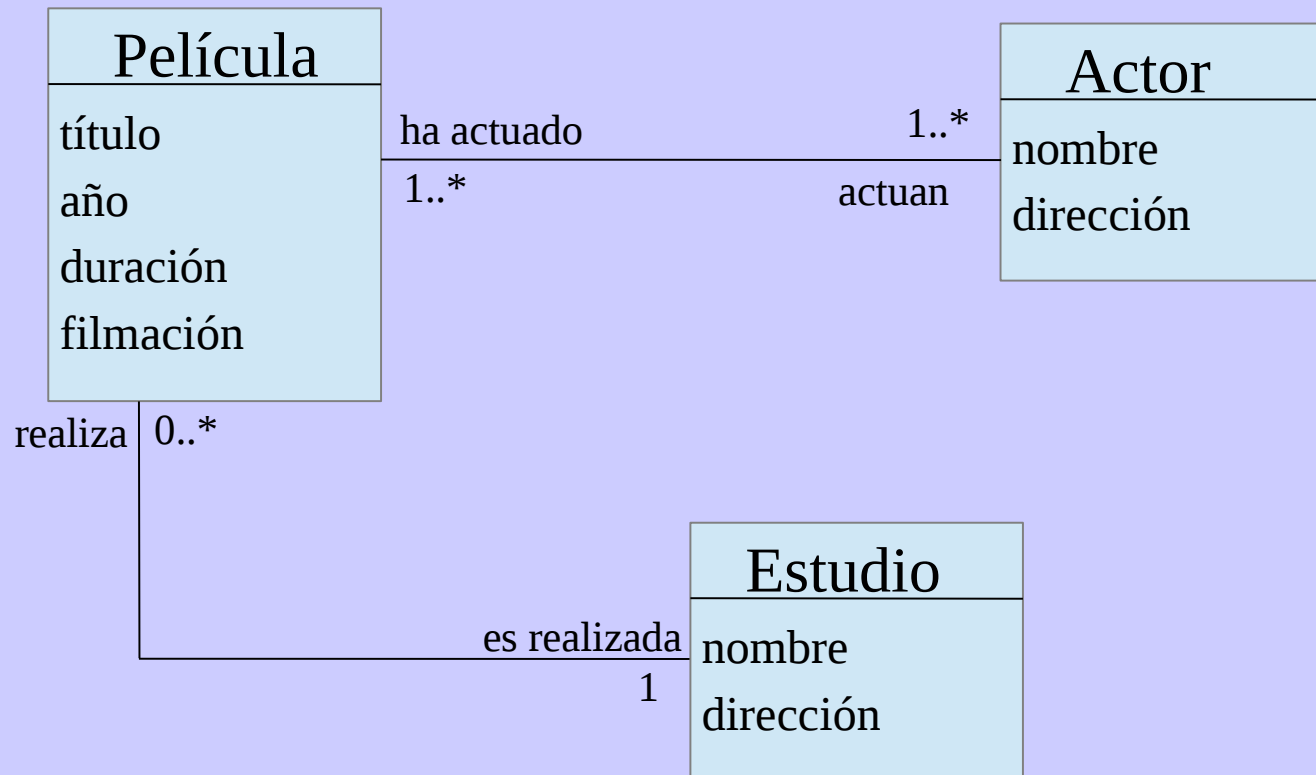
WHERE p.titulo=“El Código Enigma” AND a.nombre=“Allen Leech”

Si esto no devuelve una colección
puede usarse en el WHERE

OQL: Cuantificadores

- Con DISTINCT elimino duplicados en el resultado de la consulta, igual que SQL.
- Expresiones con Cuantificadores
 - **Cuantificador Universal:**
 - FOR ALL x IN S : $C(x)$ es verdadero si toda x de S satisface $C(x)$ y falso en caso contrario
 - **Cuantificador Existencial:**
 - EXISTS x IN S : $C(x)$ es verdadero si hay por lo menos una x en S tal que $C(x)$ sea verdadero sino es falso

ODL-OQL: Ejemplo



ODL: Ejemplo

```
class Pelicula (extent Peliculas){  
  attribute string titulo;  
  attribute integer año;  
  attribute integer duración;  
  attribute enum filmación (color, blancoNegro) tipoDeFilm;  
  relationship Set ⟨Actor⟩ actuan  
    inverse Actor::haActuado;  
  relationship Estudio esRealizada  
    inverse Estudio::realiza;  
};
```

```
class Actor (extent Actores){  
  attribute string nombre;  
  attribute string dirección;  
  relationship Set ⟨Pelicula⟩ haActuado  
    inverse Pelicula::actuan;  
};
```

```
class Estudio (extent Estudios){  
  attribute string nombre;  
  attribute string dirección;  
  relationship Set ⟨Pelicula⟩ realiza  
    inverse Pelicula::esRealizada;  
};
```

OQL: Ejemplos

• **Cuantificador Existencial**

- Encontrar todos los actores de las películas del estudio Disney

Devuelve solo un objeto

SELECT a

FROM a IN Actores

WHERE EXISTS p IN a.haActuado :

p.esRealizada.nombre="Disney"

OQL: Ejemplos

• **Cuantificador Universal**

- Encontrar los actores que han aparecido solamente en las películas del estudio Disney

SELECT a

FROM a IN Actores

WHERE FOR ALL p IN a.haActuado :

p.esRealizada.nombre="Disney"

OQL – Asignación de Variables

• **Asignación de variables**

- Las consultas OQL producen objetos como valores resultado
- En C++:

```
SELECT DISTINCT p  
FROM p IN Peliculas  
WHERE p.año < 1920
```

- Genera el conjunto de las películas realizadas antes de 1920

OQL – Asignación de Variables

- Definimos una variable del tipo conjunto de películas (Set⟨Pelicula⟩) para obtener ese resultado

```
peliculasViejas = SELECT DISTINCT p  
FROM p IN Peliculas  
WHERE p.año < 1920
```

BD Objeto-Relacionales

- Son BDR que adicionan características de las bases de datos OO
- Esto adiciona al SQL92 varias primitivas de OO quedando en el SQL3.
- **Algunos aspectos extendidos son:**
 - *Constructores de tipos*: para especificar objetos complejos. A traves de tipos de datos definidos (UDTs).
 - *Identidad de los objetos*: a través del uso de referencias de tipos.
 - *Herencia*: a través de la palabra UNDER.
 - *Relaciones Anidadas*: los dominios de los atributos pueden ser atómicos o de relación

BDOR: Tipos UDTs

- Se denominan **user-defined types (UDTs)**
- Se definen para:
 - permitir la creación de estructuras de objetos complejas
 - separar la declaración de tipos de la creación de una tabla

```
CREATE TYPE Nombre_Tipo AS  
(<delaracion de componentes>)
```

BDOR: Tipos UDTs (struct)

- Pueden usarse para:
 - definir el tipo de un atributo
 - definir el tipo de una tabla

Luego estos tipos se usan como tipos de datos en tablas de la BD

```
CREATE TYPE tipo_direccion AS (  
    numero INTEGER,  
    nombre_calle VARCHAR(20));
```

```
CREATE TYPE tipo_direccion_completa AS (  
    direccion TIPO_DIRECCION,  
    ciudad VARCHAR(25),  
    cp VARCHAR(10));
```

BDOR: Tipos UDTs (array)

- Puede usarse array, multiset, list y set

Se diferencian en permitir repetidos, ordenamiento y cantidades fijas de elementos

```
CREATE TYPE tipo_telefonos AS (  
    notipo VARCHAR(5),  
    codigo_area INTEGER,  
    numero VARCHAR(10));
```

```
CREATE TYPE tipo_persona AS (  
    dni VARCHAR(20),  
    nombre VARCHAR(20),  
    apellido VARCHAR(20),  
    fecha_nacimiento DATE,  
    telefonos tipo_telefonos ARRAY[4]);
```

telefonos[1] se refiere al primer telefono del arreglo

BDOR: Identidad de Objetos

- Los identificadores de objetos pueden ser creados por medio de tipos de referencia
- Se usan para tipos o tablas luego de la creación de tipos o tablas.

REF IS <atributo OID> <Metodo_de_Generación>

```
CREATE TABLE Persona OF tipo_persona  
REF IS id_persona SYSTEM GENERATED
```

```
CREATE TABLE Persona OF tipo_persona(  
REF IS id_persona DERIVED;  
PRIMARY KEY (dni));
```

BDOR: Herencia

- Una subrelación (especialización) hereda todos los atributos de su superrelación (generalización).
- Cada tupla de una subrelación corresponde a una única tupla en la superrelación.
- Cada tupla de una superrelación corresponde a lo sumo, en una subrelación, a una de sus tuplas.
- Puede haber tuplas de una superrelación que no se correspondan con ninguna tupla de una subrelación.

BDOR: Herencia

- Las operaciones de inserción, modificación o borrado de tuplas se propagan adecuadamente entre subrelaciones y superrelaciones.
- Puede haber herencia de tablas y de tipos.

```
CREATE TABLE Persona OF tipo_persona (  
    REF IS id_persona DERIVED;  
    PRIMARY KEY (dni));
```

```
CREATE TABLE Estudiante(  
    curso VARCHAR(20),  
    dpto VARCHAR(20))  
UNDER persona;
```

```
CREATE TABLE Profesor(  
    sueldo integer)  
UNDER persona;
```


BDOR: Herencia

- Las subrelaciones *estudiante* y *profesor* heredan **TODOS** los atributos de la tabla persona.
- Cada tupla de la superrelacion *persona* puede corresponderse como máximo con una tupla de las relaciones *estudiante* y *profesor*.
- Cada tupla de *estudiante* y *profesor* debe tener una tupla correspondiente en persona

BDOR: Relaciones Anidadas

- El valor de las tuplas de los atributos puede ser una relación y las relaciones pueden guardarse en otras relaciones.

- *lista-autores SETOF(REF(Persona))*

título	lista_autores
Fundamentos de Bases de Datos	{Silberchatz, Korth, Sudarshan}
El Lenguaje de Programación C	{Kernghan, Ritchie}

Comparación entre las BDs

BD Relacionales	BDOR	BDOO
<ul style="list-style-type: none">Trabajan con tipos de datos sencillos (respetan 1FN)Buena protección de los datos (usa SQL)Las mismas desventajas que las BDROO	<ul style="list-style-type: none">Buena protección de los datos (respecto a errores de programación ya que SQL posee una potencia limitada)Optimizaciones sencillasFácil manipular datos complejosRealizan gran cantidad de accesos a la DBNo trabajan en memoria principalNo están integrados con los lenguajes de programación y generalmente hay que hacer traducciones	<ul style="list-style-type: none">Poseen un rendimiento elevado, ya que trabajan en memoria principalNo debemos hacer traducción de los datosTrabajan con datos complejosEstán integrados con los lenguajes de programaciónSon muy potentes y pueden causar deterioro en los datos almacenados físicamenteNo suelen disponer de grandes disponibilidades de consulta