PATRONES DE PERSISTENCIA DE OBJETOS

OBJETIVOS

- Conocer cuáles son los patrones para la persistencia de objetos.
- Seleccionar adecuadamente el patrón de persistencia de objetos de acuerdo con los requerimientos del proyecto.

1

PATRONES DE PERSISTENCIA

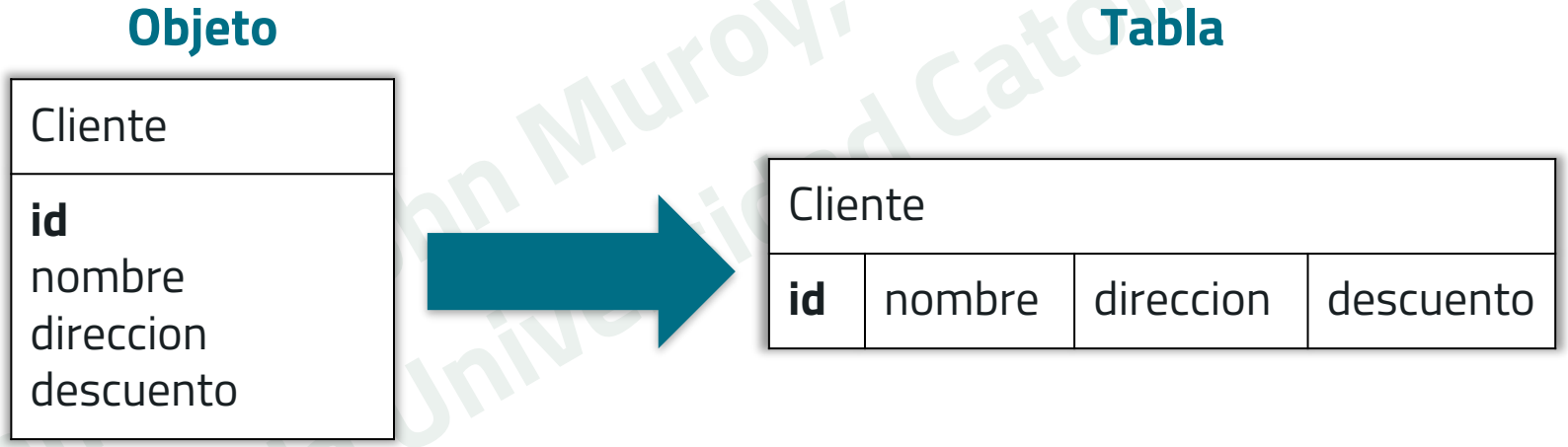
Patrones comúnmente utilizados ...

RELACIÓN ENTRE OBJETOS Y TABLAS

Guían el modelamiento de las tablas en base a las clases de diseño.

PATRÓN: IDENTITY FIELD

Almacena el id del registro de base de datos en el objeto.



Caso de Uso:

Mapear objetos en memoria y registros en base de datos.

¿CÓMO SELECCIONAMOS EL IDENTIFICADOR?

Llave representativa vs. Llave no representativa

(Ejemplo: DNI vs. Valor entero aleatorio o incremental)

- La llave debe ser única e inmutable.
- La llave representativa podría requerir edición (error humano).

Llave simple vs. Llave compuesta

(Un campo vs. Varios campos)

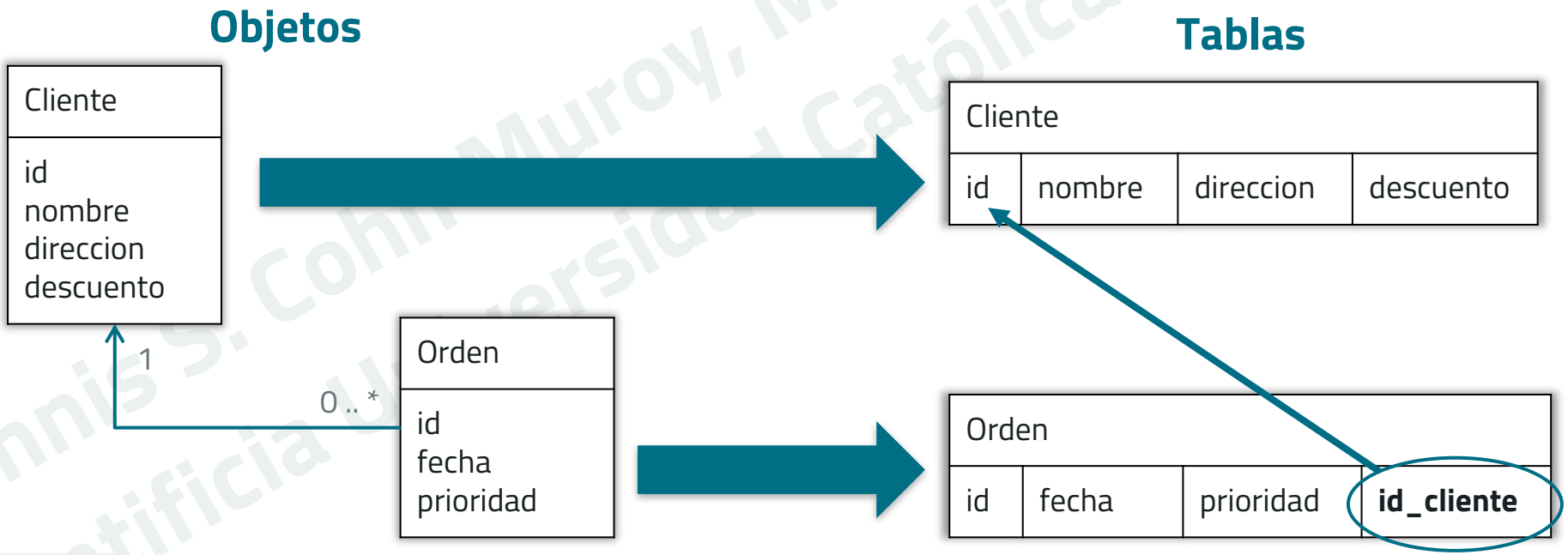
- La llave compuesta requiere ser única a nivel de su combinación.
- Las llaves compuestas necesitan utilizar lógica para su creación.

¿CÓMO CALCULAMOS EL IDENTIFICADOR?

- Estrategias:
  - Que la base de datos la autogenere.
  - GUID (Global Unique IDentifier): Valor único entre todas las bases de datos.
  - Crear tu propio método.

PATRÓN: FOREIGN KEY MAPPING

La relación entre dos clases se representa a través de una llave foránea.

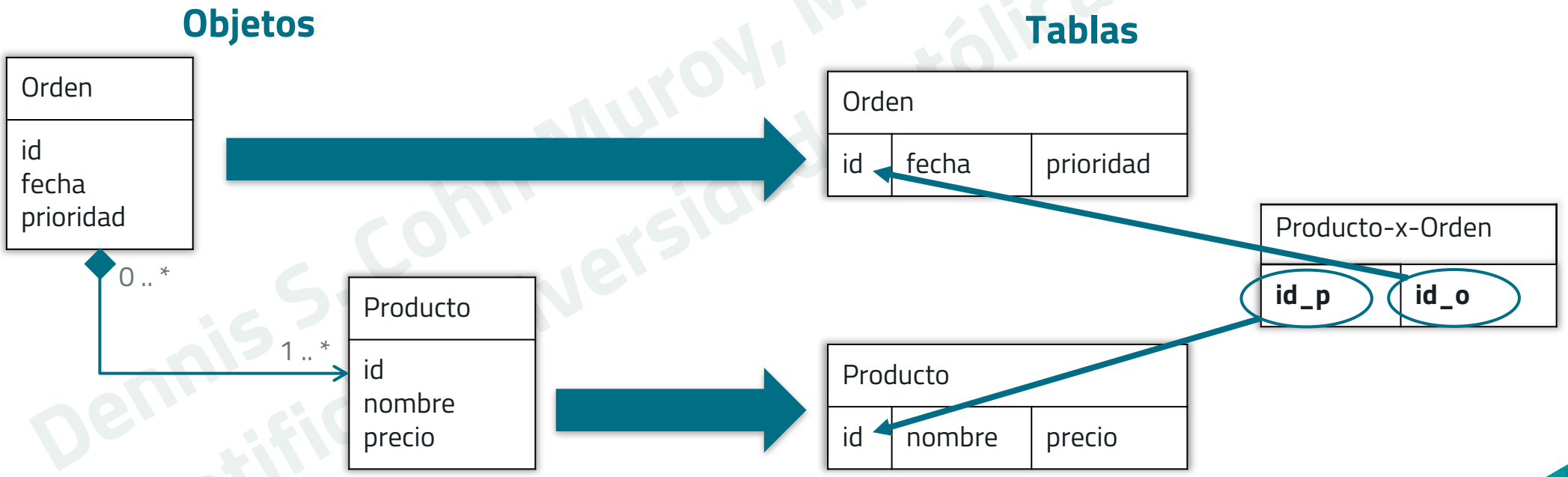


Caso de Uso:

Cuando la multiplicidad entre clases es de “1 a 1”; o de “1 a muchos”.

PATRÓN: ASSOCIATION KEY MAPPING

La relación entre dos clases se representa a través de una tabla intermedia.



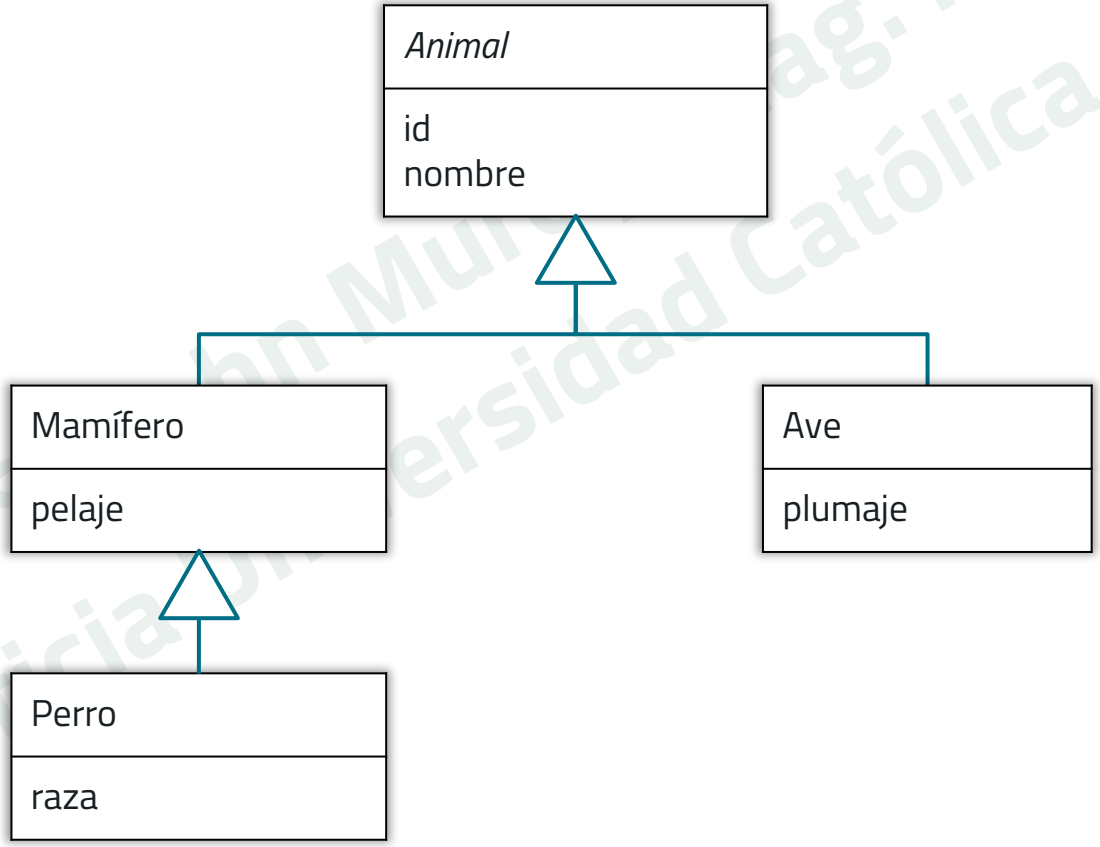
Caso de Uso:

Cuando la multiplicidad entre clases es de “muchos a muchos”.

PATRONES PARA LAS HERENCIAS DE CLASES

Existe más de una solución ...

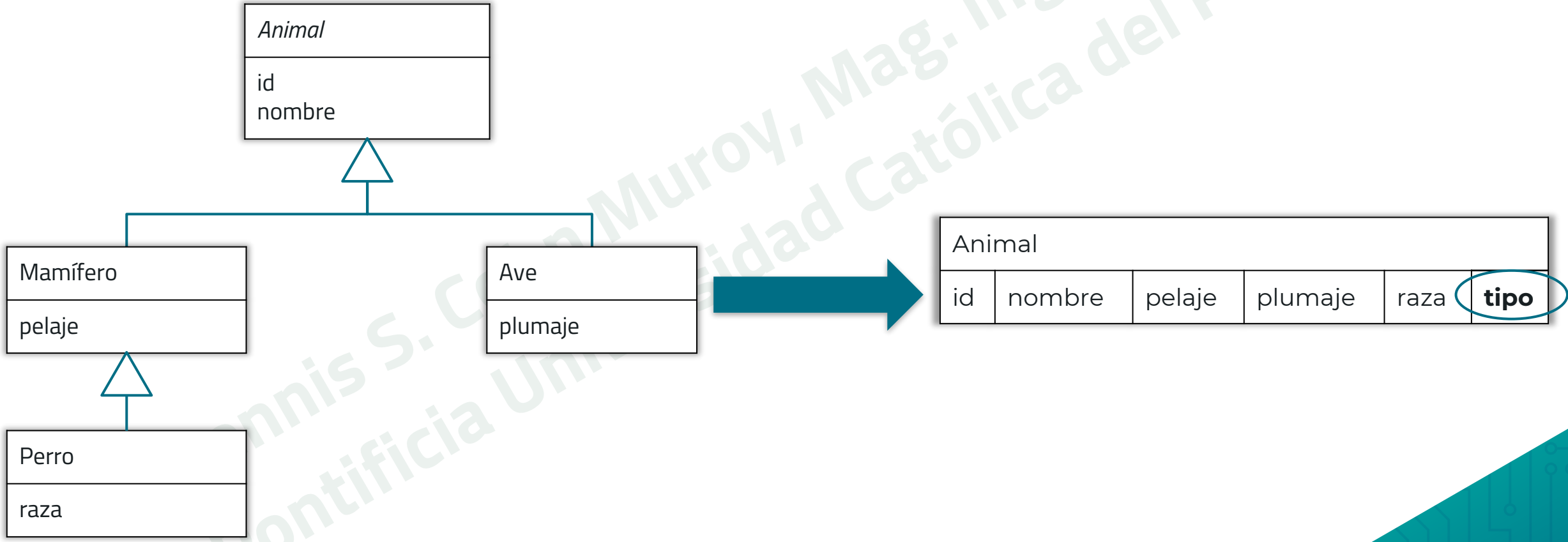
SI TENEMOS LA SIGUIENTE HERENCIA ...



... ¿QUÉ SOLUCIÓN PROPONEN?

PATRÓN: SINGLE TABLE INHERITANCE

La herencia se mapea contra una única tabla que contiene todos los atributos de cada uno de los objetos.



CONSIDERACIONES: SINGLE TABLE INHERITANCE

Ventajas:

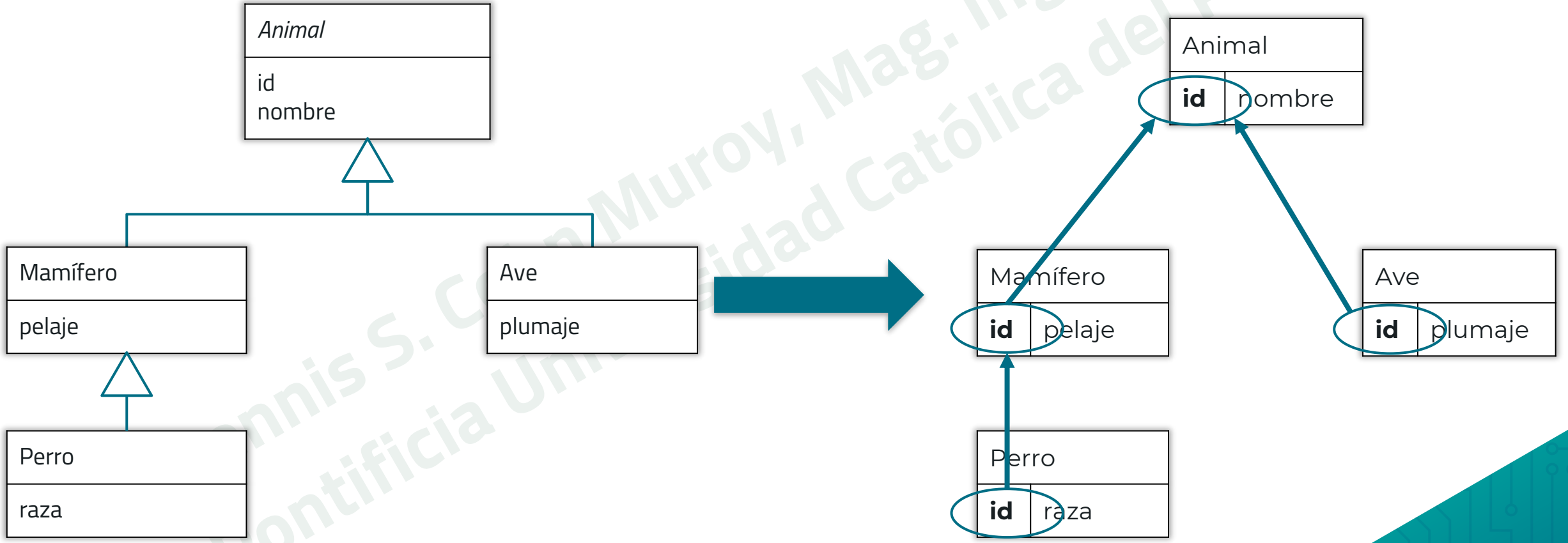
- Una única tabla que mantener.
- No se necesitan "*joins*" para obtener los datos.
- El mover atributos entre clases no implica un cambio en la tabla.

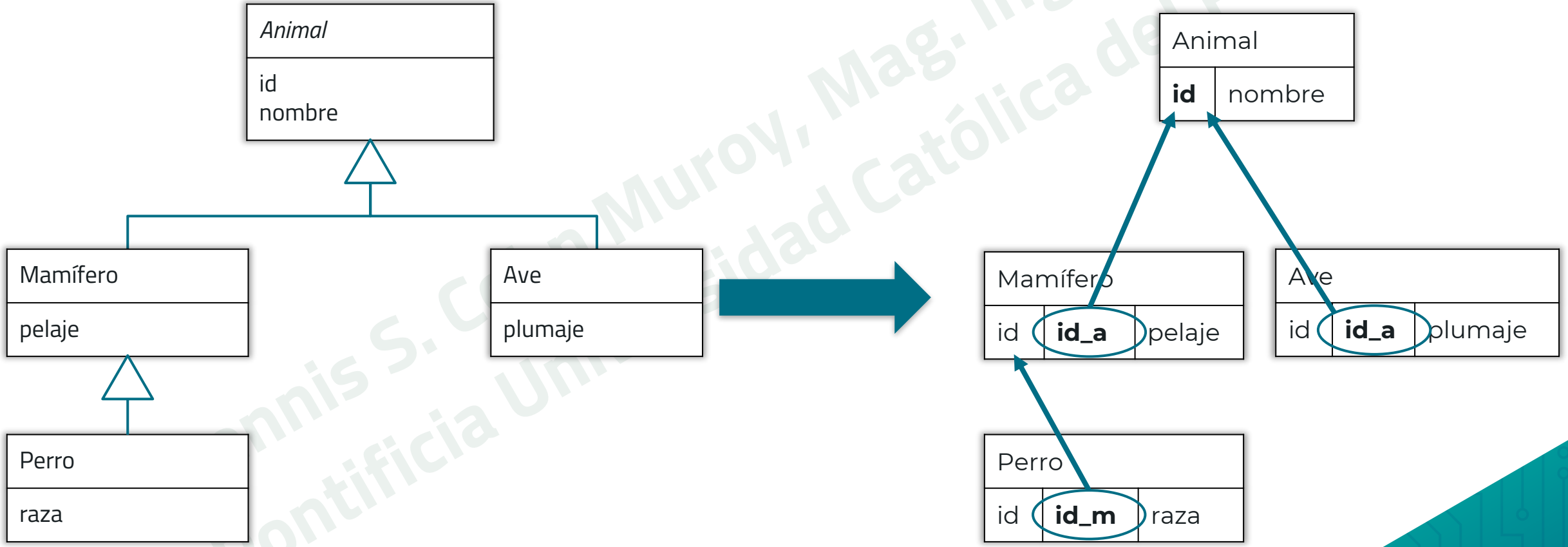
Desventajas:

- Complejidad para determinar la importancia de las columnas.
- Desperdicio de espacio por valores nulos.
- El alto número de registros puede impactar en el desempeño.
- Complejidad al agregar nuevas clases hijas.

PATRÓN: CLASS TABLE INHERITANCE

Cada clase que forma parte de la jerarquía de la herencia es representada por su propia tabla.





CONSIDERACIONES: CLASS TABLE INHERITANCE

Ventajas:

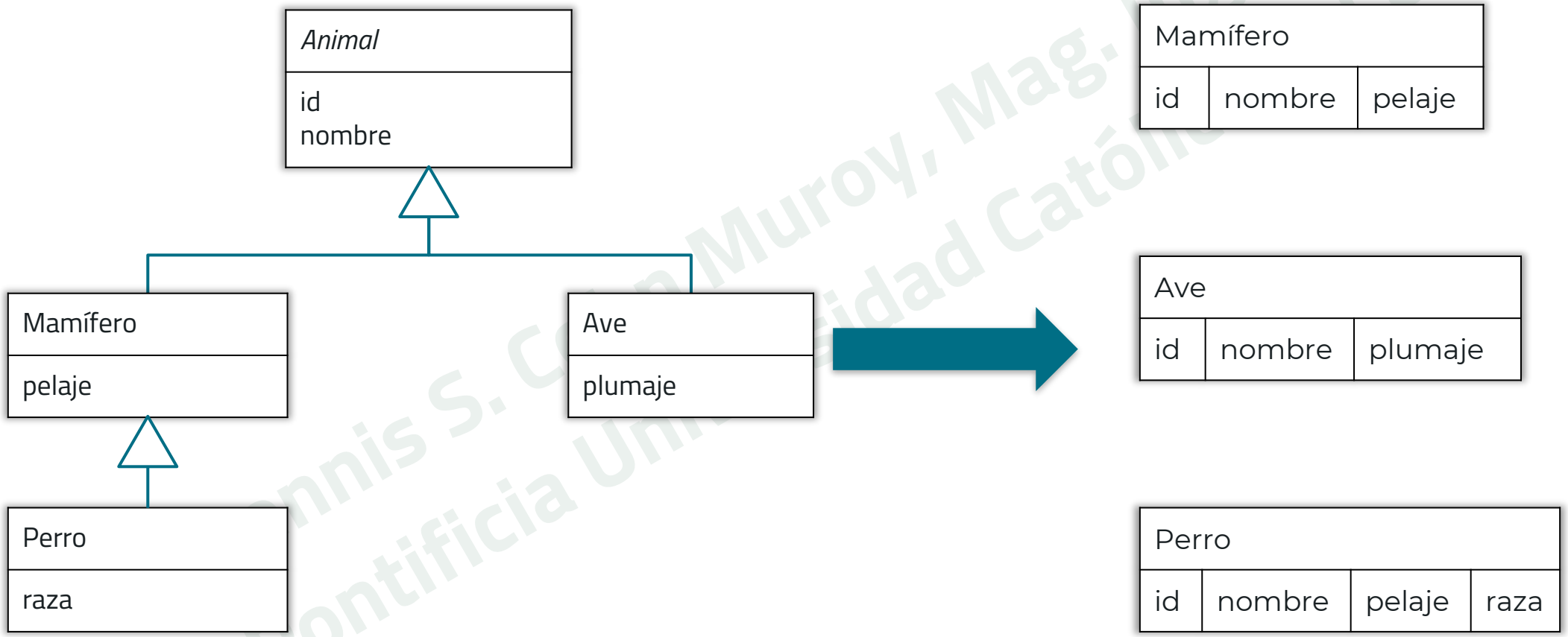
- No hay desperdicio de espacio.
- Modelo de datos entendible.
- Relación directa entre las clases y las tablas.
- Menor impacto al agregar nuevas clases hijas.

Desventajas:

- Consultas complejas con múltiples "*joins*".
- El mover atributos entre clases implica un cambio en la tabla.
- La clase "padre" puede ser un cuello de botella.

PATRÓN: CONCRETE TABLE INHERITANCE

Cada clase concreta que forma parte de la jerarquía de la herencia es representada por su propia tabla.



CONSIDERACIONES: CONCRETE TABLE INHERITANCE

Ventajas:

- No hay desperdicio de espacio.
- Si se requiere leer datos de una clase concreta, no se requiere utilizar "*joins*".
- Una tabla es consultada únicamente cuando se utiliza la clase correspondiente.
- Menor impacto al agregar nuevas clases hijas.

Desventajas:

- El mover atributos entre clases implica un cambio en la tabla.
- El modificar los atributos de la clase "padre" puede impactar en varias tablas.
- Consultas complejas si se requieren los registros sobre la clase "padre".

ES POSIBLE COMBINAR LOS 3 PATRONES

Dentro del diseño para un sistema.

2

REFERENCIAS

BIBLIOGRAFÍA

- Fowler, M. (2012). Patterns of Enterprise Application Architecture. Addison-Wesley.

Créditos:

- Plantilla de la presentación por SlidesCarnival
- Fotografías por Unsplash
- Diseño del fondo Hero Patterns