

Bases de Datos

Clase 5: Diseño de Bases de Datos

Hasta ahora

Conocemos el modelo relacional y SQL por lo que podemos comenzar a diseñar una base de datos, pero... ¿Sabemos si lo estamos haciendo bien?

Un error en la modelación puede ser muy costoso!

Por ejemplo: olvidar añadir una columna

Diseño de base de datos

Análisis de requisitos

Usuarios



Requisitos



Diseño conceptual de bases de datos

Requisitos



Modelo entidad-relación



Diseño lógico de bases de datos

Modelo entidad-relación



Modelo relacional

Construir una aplicación con un RDBMS

En la práctica es imposible saber de antemano todos los requisitos que debe cumplir un software.

De la mano con eso, el esquema de la base de datos va cambiando a medida que progresa el desarrollo de un proyecto.

Un esquema bien diseñado no solo nos permite consultar con facilidad y guardar los datos de forma óptima, si no que también permite modificarlo y aumentarlo con menos dolores de cabeza.

Los errores en el diseño son muy costosos a la larga!

Diseño conceptual de la BD

Por qué diseñar y diagramar la base de datos:

- Identificar las entidades.
- Entender cómo se asocian esas entidades.
- Visualizar las restricciones del dominio.
- Para lograr un buen diseño!
- Para mantener el esquema bien documentado.

Diagramas E/R

Diagramas E/R

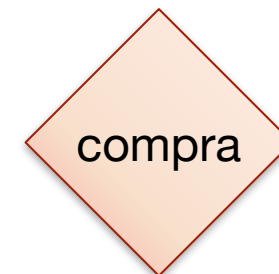
Entidad



Atributo

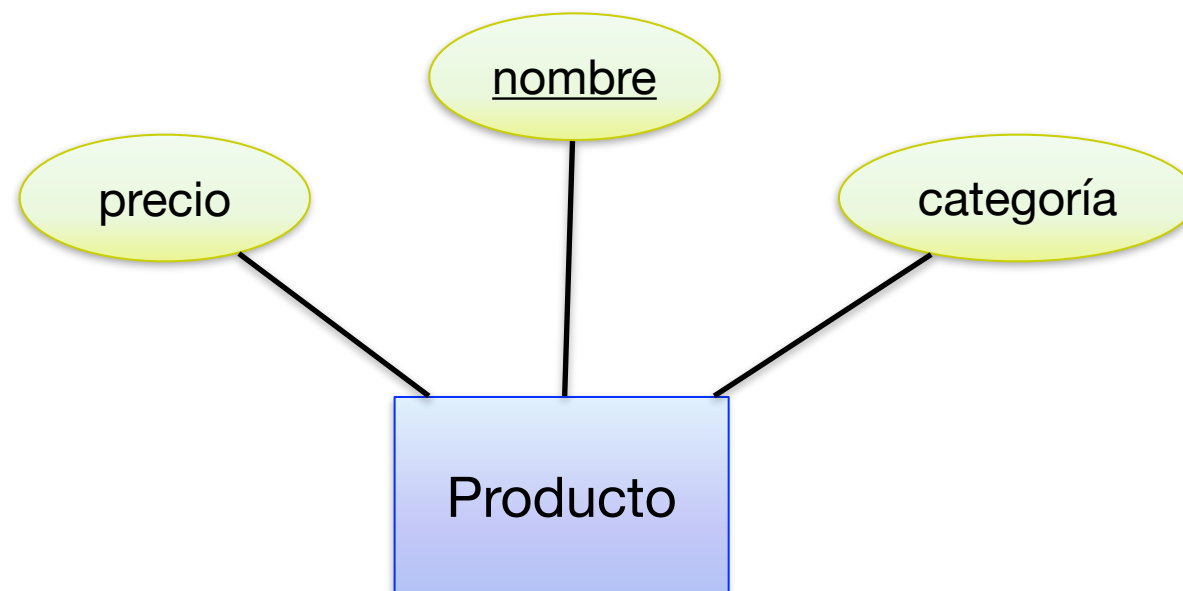


Relación



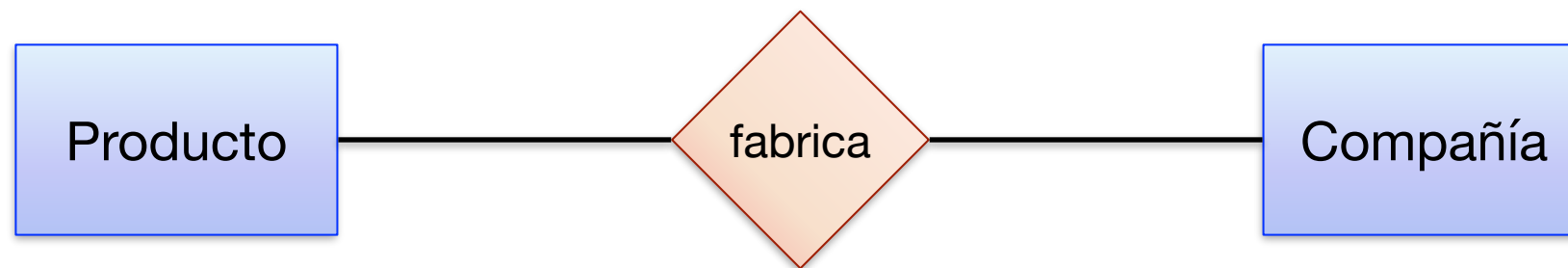
Diagramas E/R

Entidad con sus atributos



Obligatorio: cada entidad debe tener una **llave**,
i.e. un conjunto de **atributos** *mínimo*
cuyos valores identifican de manera unívoca
a cada **entidad** del conjunto

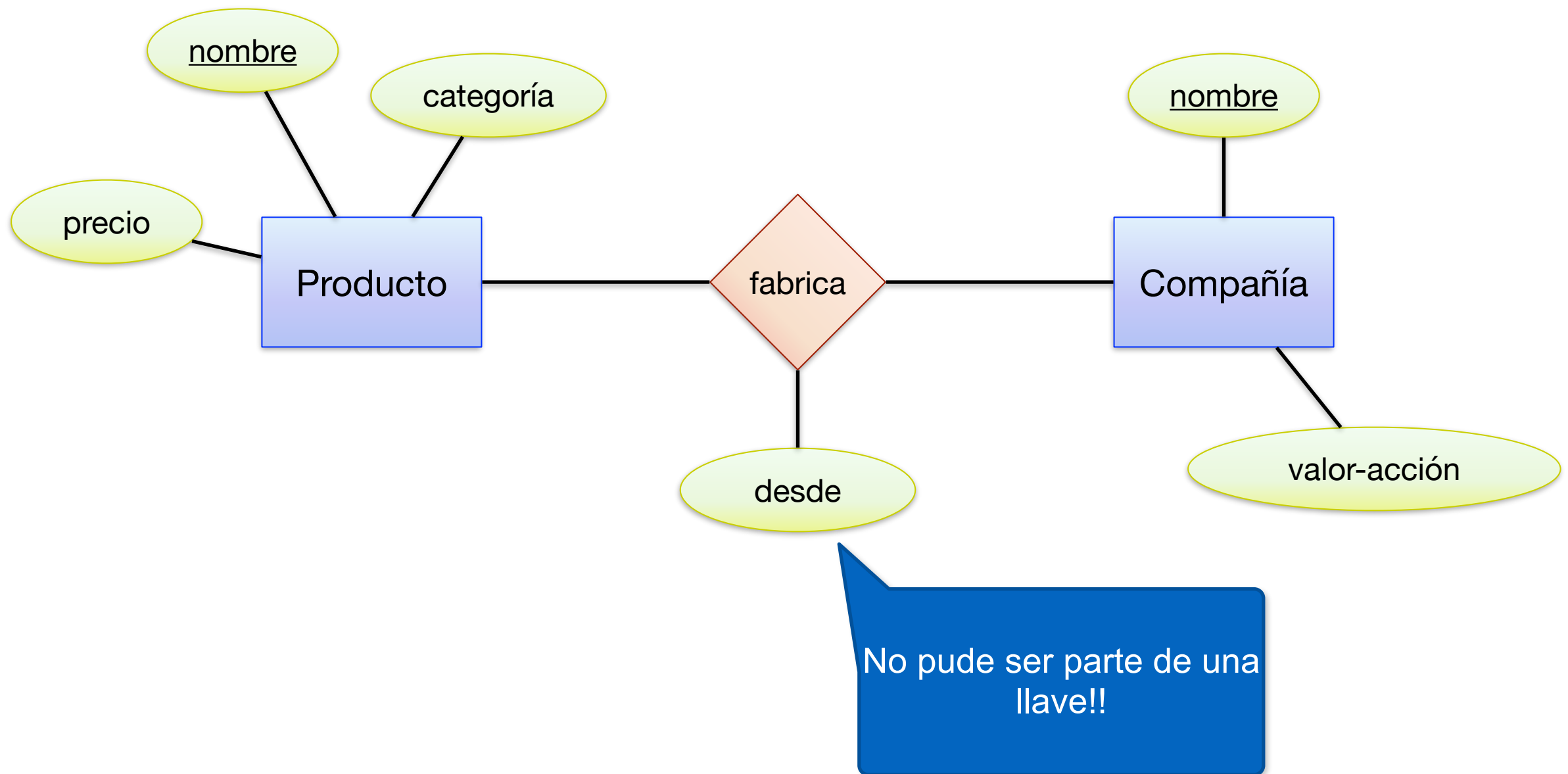
Relaciones Binarias



Dos entidades relacionadas

Relaciones Binarias

Entidad con sus atributos



Relaciones Binarias

Multiplicidades

n a n

0 o más



Una compañía puede fabricar varios productos

n a 0 o 1



Un producto se fabrica por como máximo una compañía

0 o 1 a n



Un producto es fabricado ser fabricado por muchas compañías

Una compañía fabrica como máximo un producto :S

0 o 1 a 0 o 1

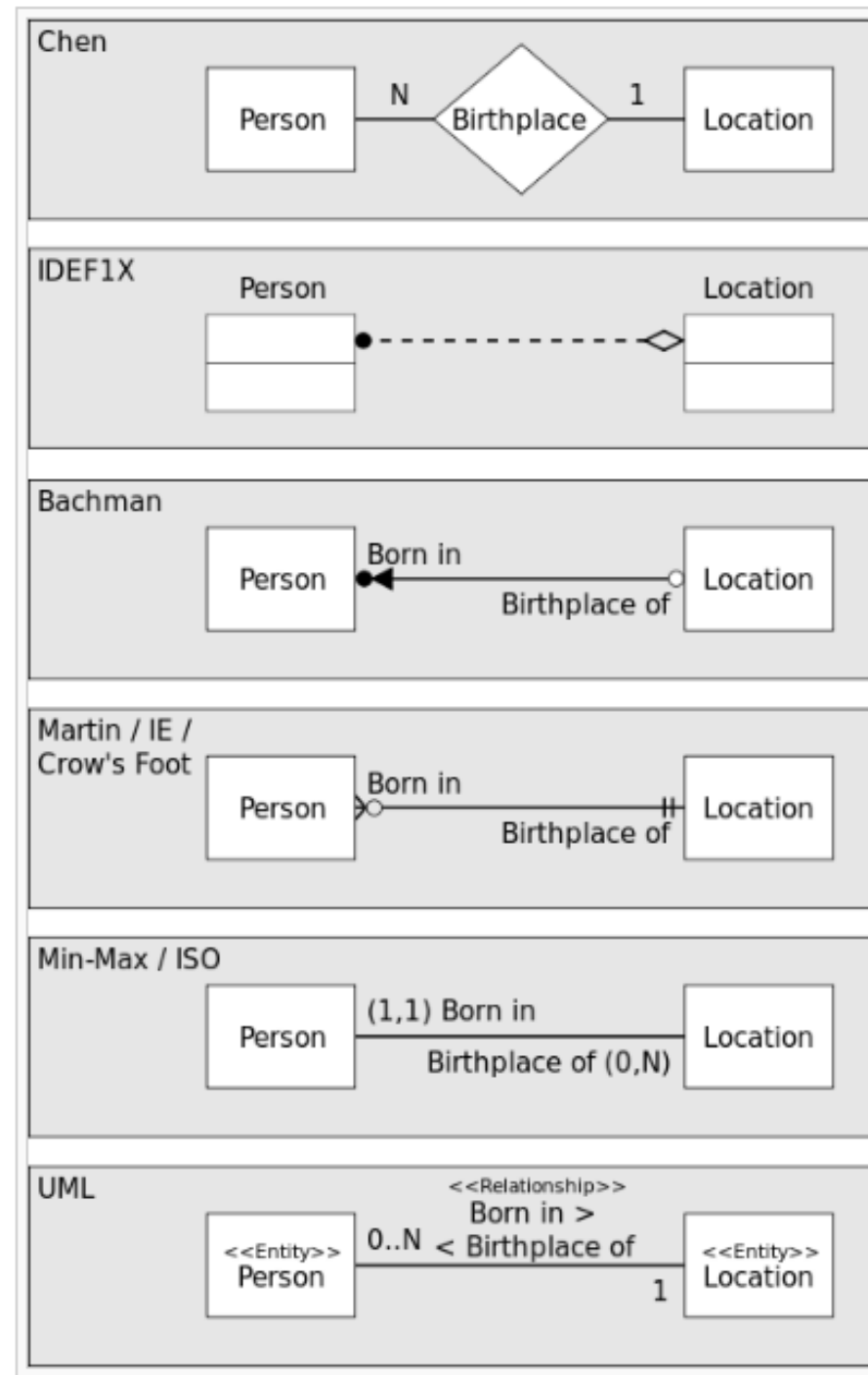


Un producto se fabrica por como máximo una compañía

Una compañía fabrica como máximo un producto :S

Relaciones Binarias

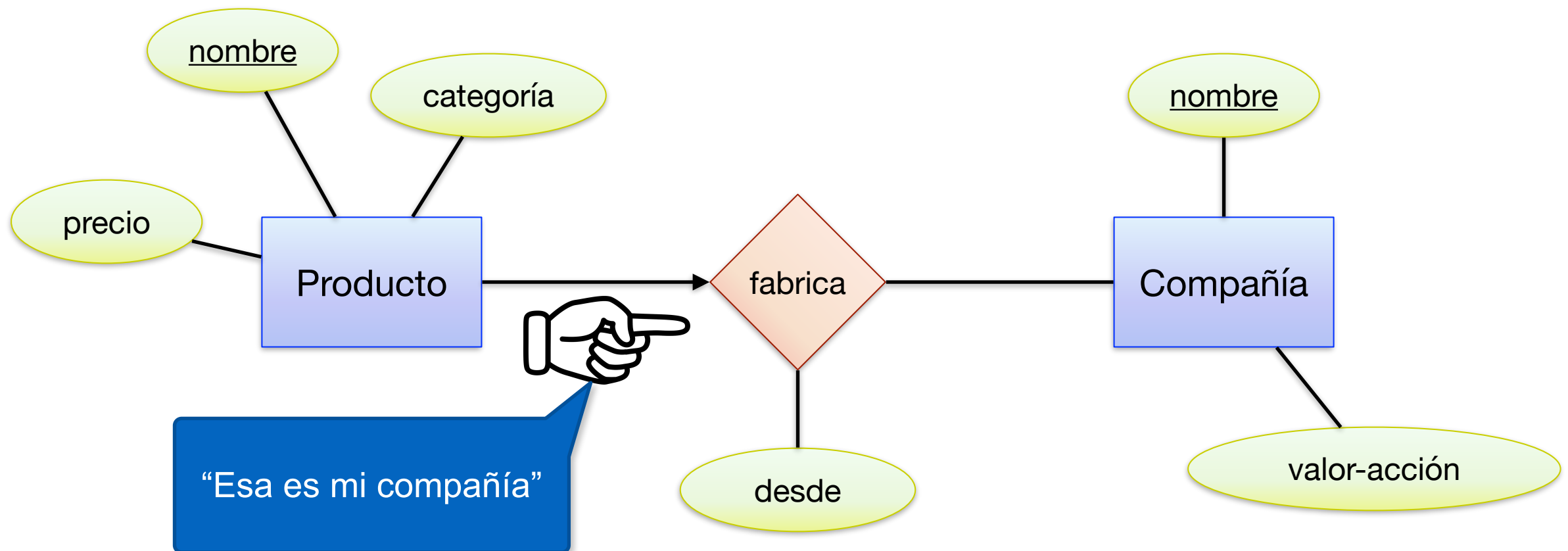
Multiplicidades



Muchas convenciones distintas

Relaciones Binarias

Sólo utilizaremos esta convención:



Multiplicidad de atributos es siempre a 1

Diagramas E/R

Relaciones Múltiples

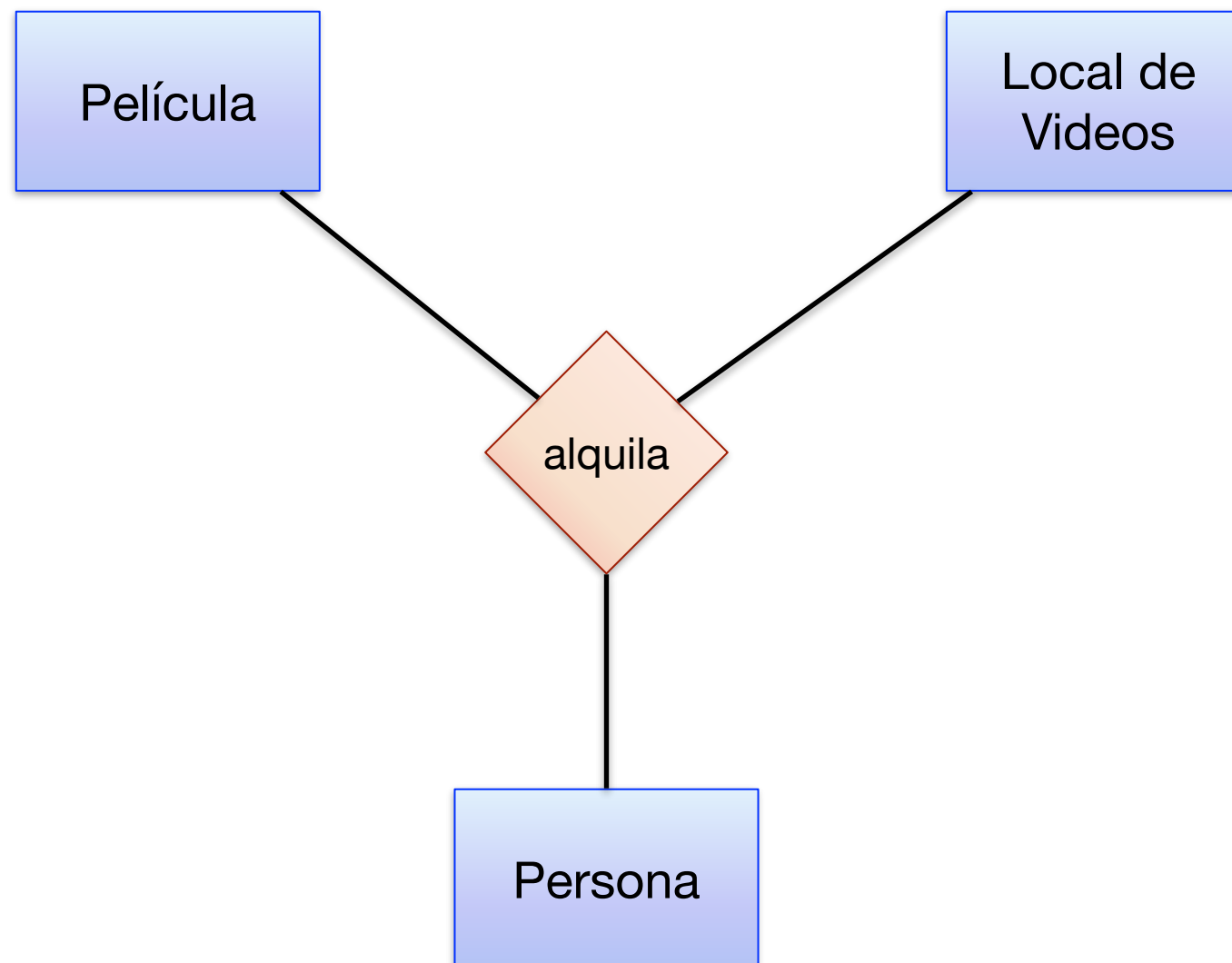
Relaciones Múltiples

¿Cómo se puede modelar un alquiler
que involucra **Personas**, **Películas** y
Locales de Videos?



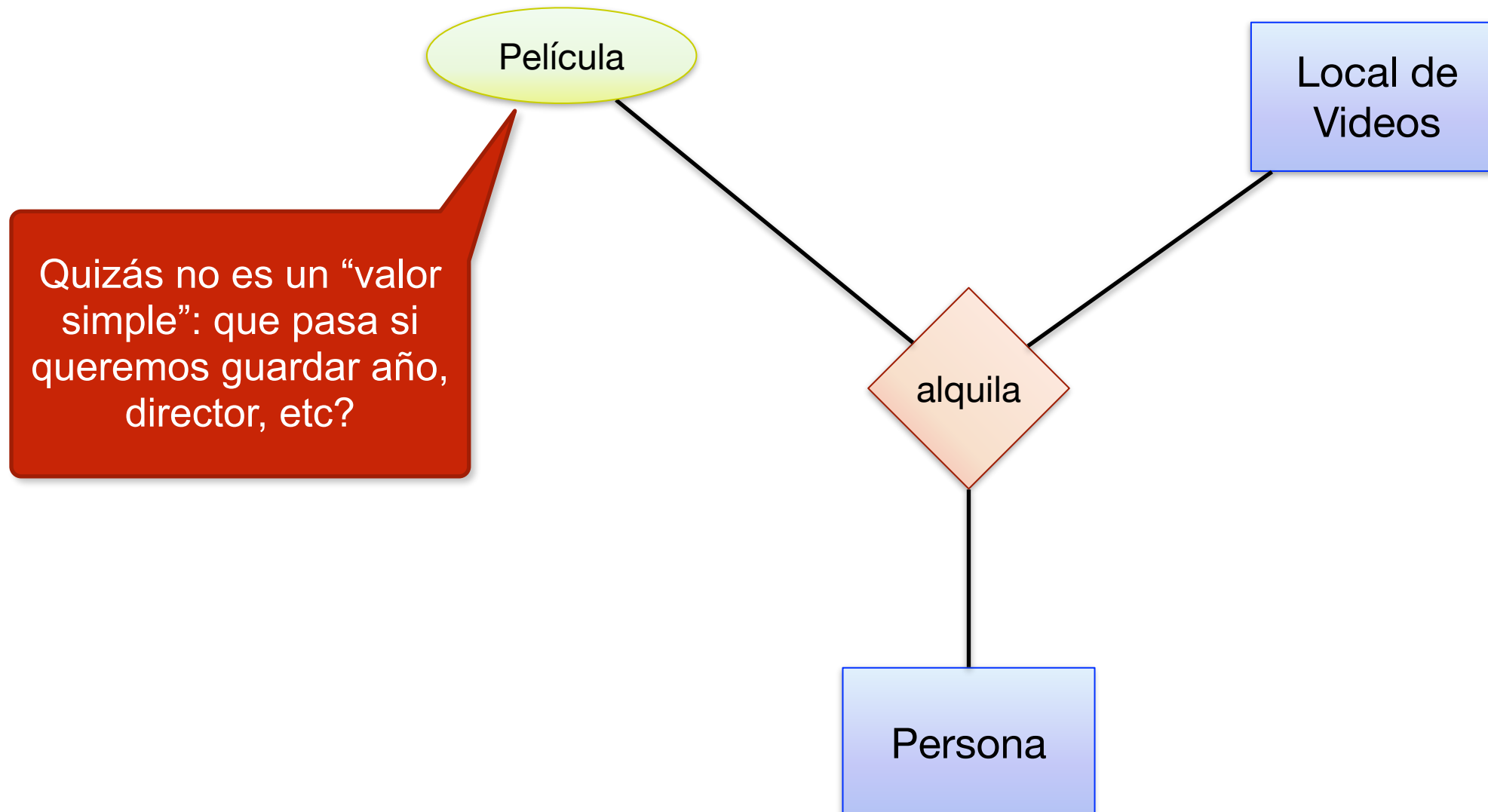
Relaciones Múltiples

¿Cómo se puede modelar un alquiler
que involucra **Personas**, **Películas** y
Locales de Videos?



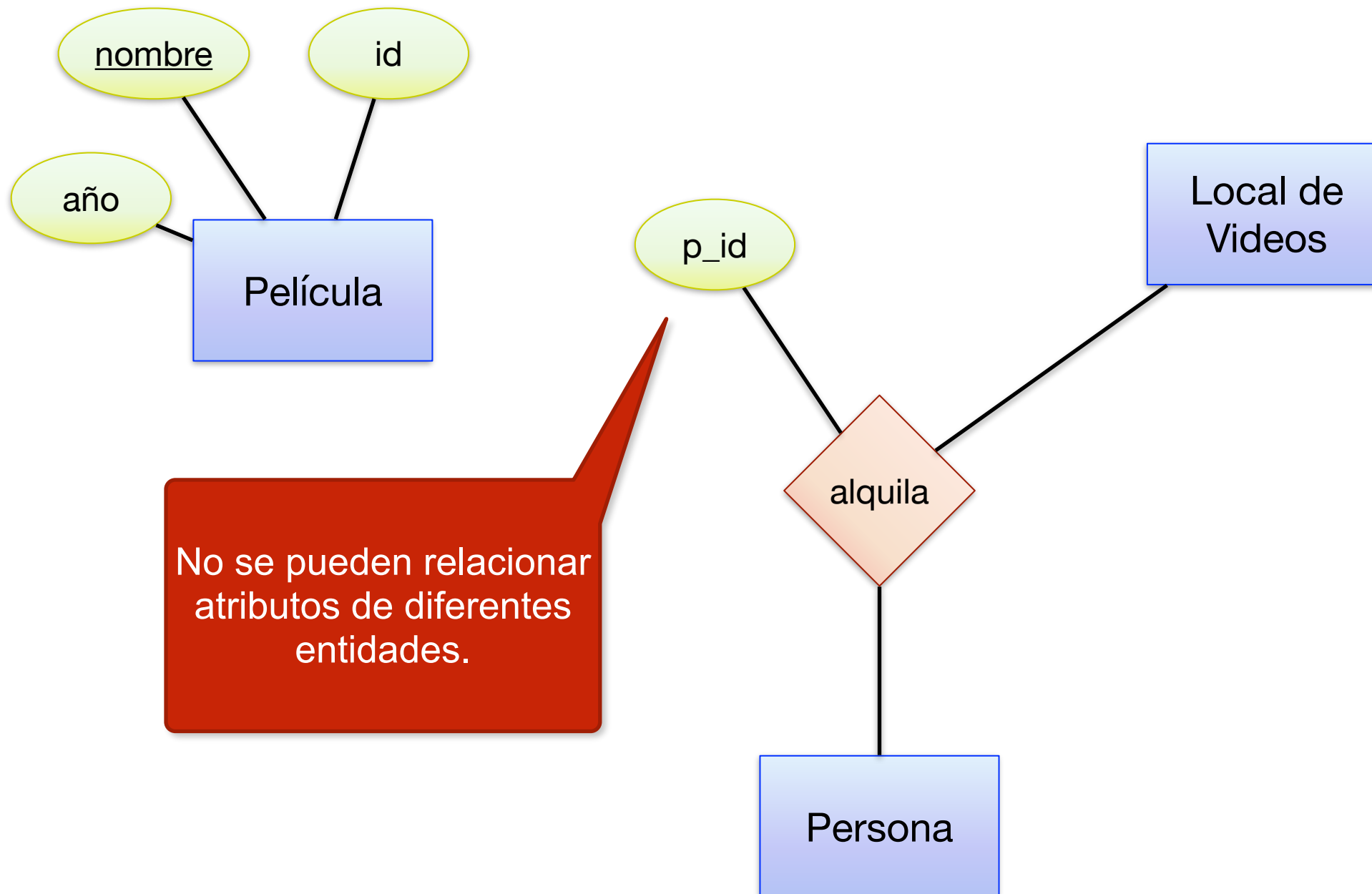
Relaciones Múltiples

¿Por qué no un atributo?



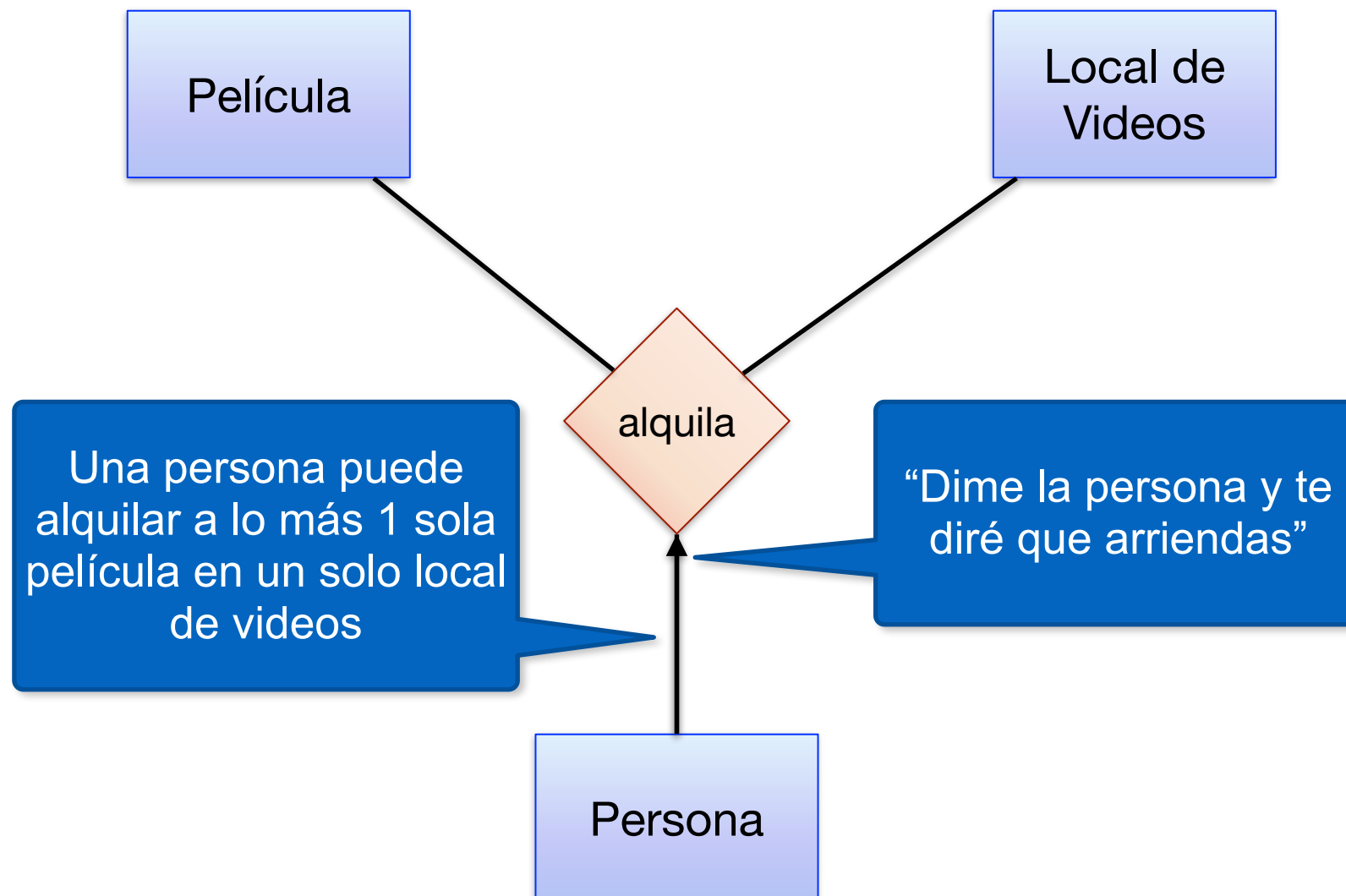
Relaciones Múltiples

¿Y ahora?



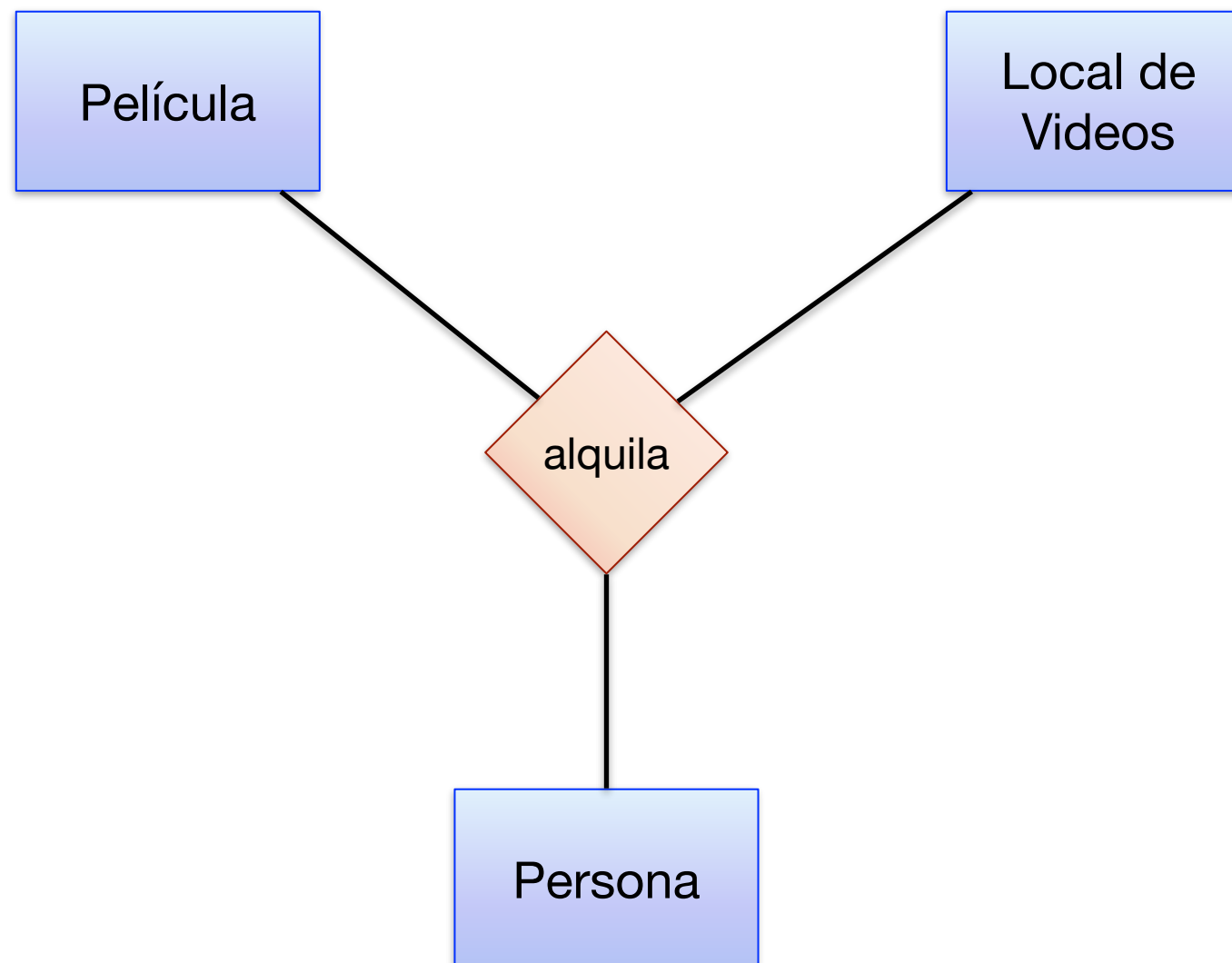
Relaciones Múltiples

¿Qué significa esto?



Relaciones Múltiples

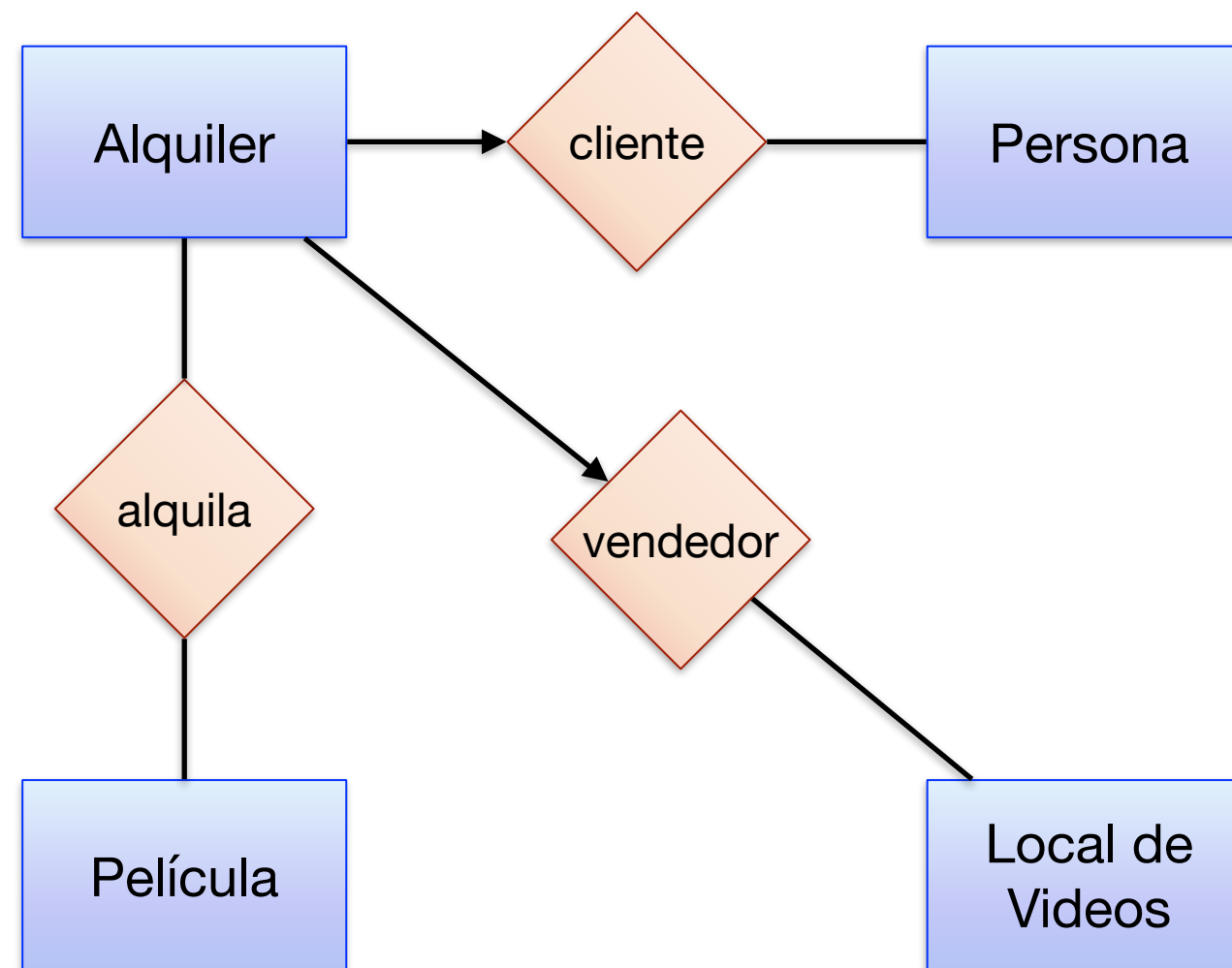
¿Y qué pasa si decimos que una **persona** puede alquilar **varias películas** pero de **un solo local de video**?



Ups...

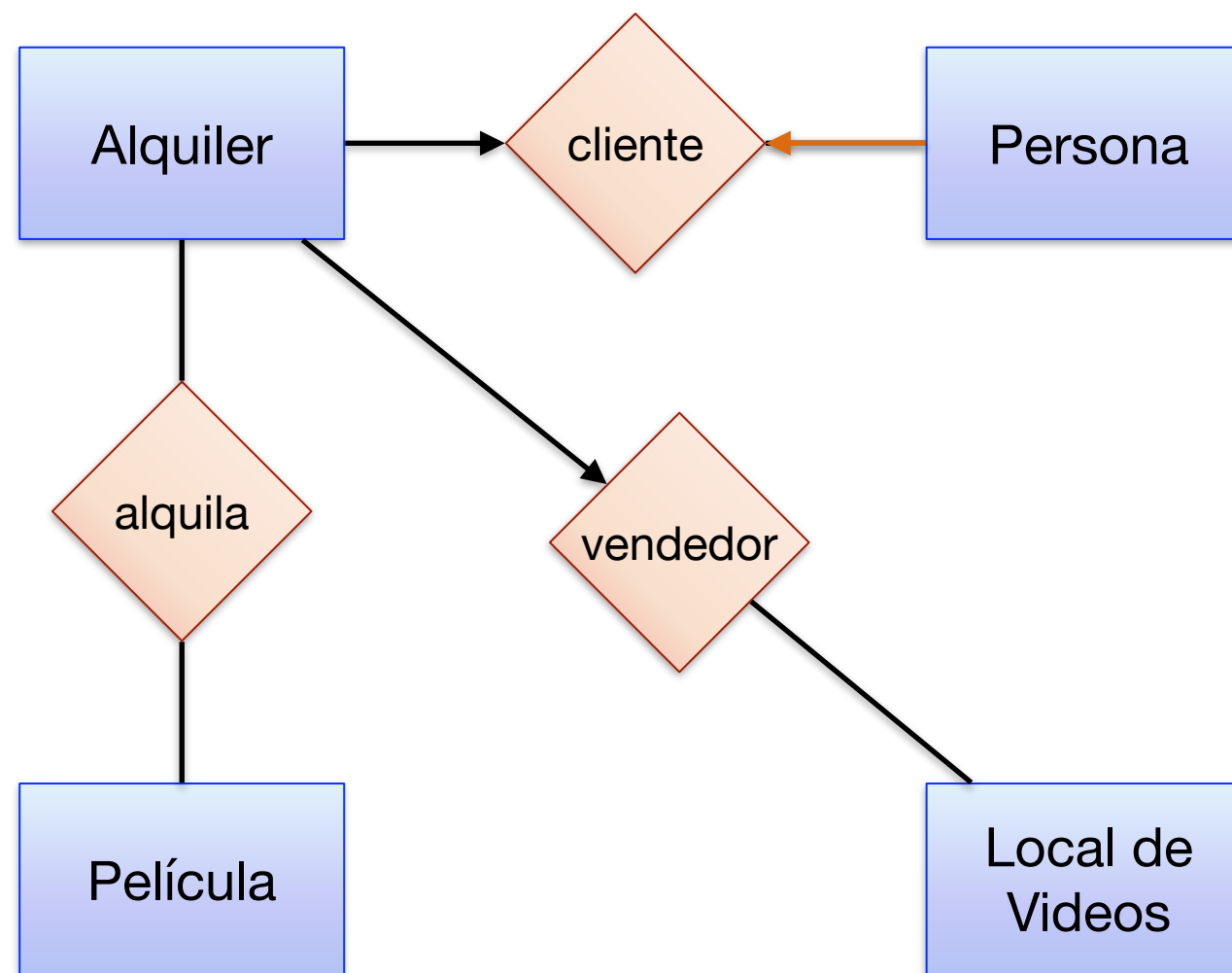
Relaciones Múltiples

¿Qué pasa si usamos solo relaciones binarias?



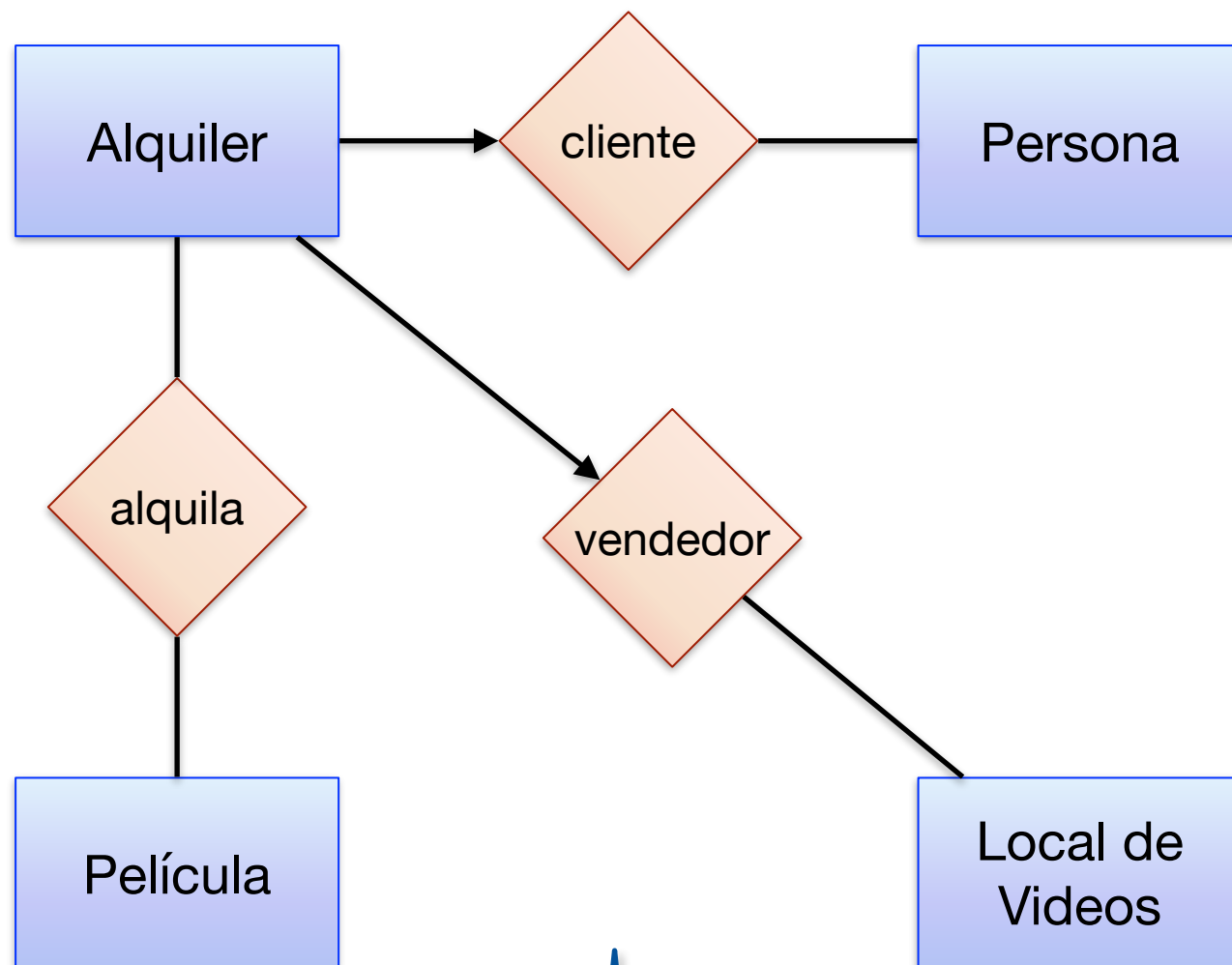
Relaciones Múltiples

¿Y qué pasa si decimos que una **persona** puede alquilar **varias películas** pero de **un solo local de video**?

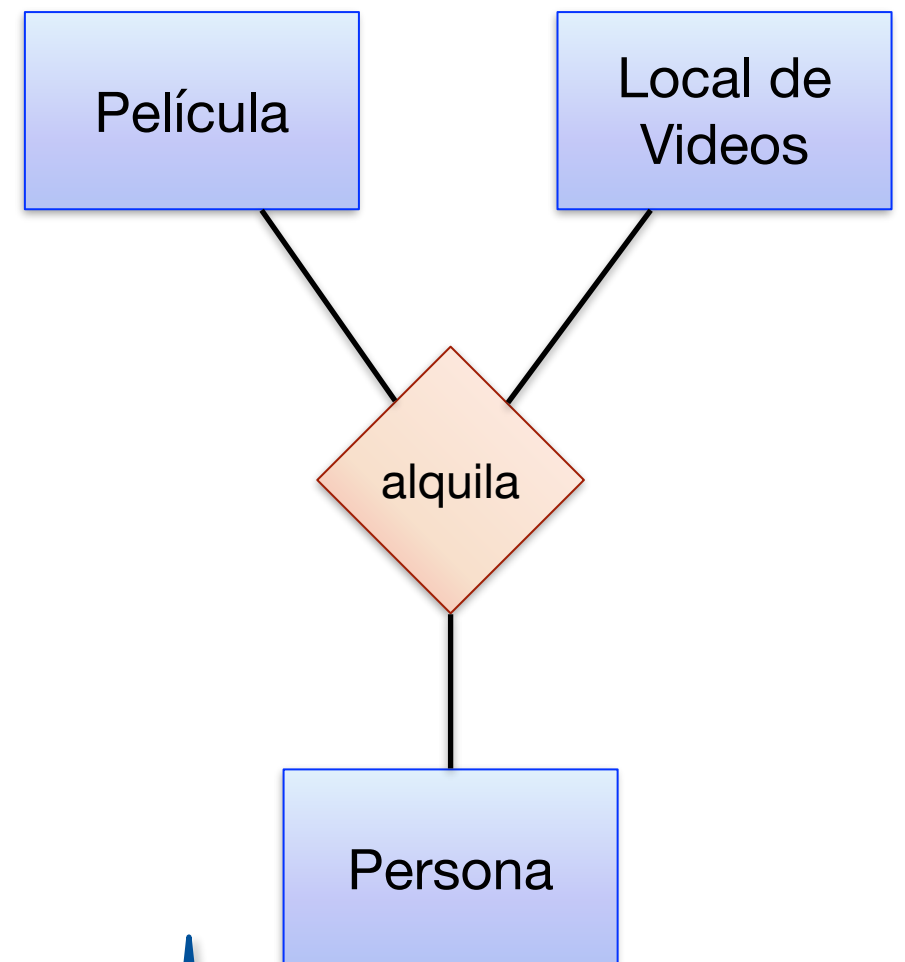


Relaciones Múltiples

¿Cuál es mejor?



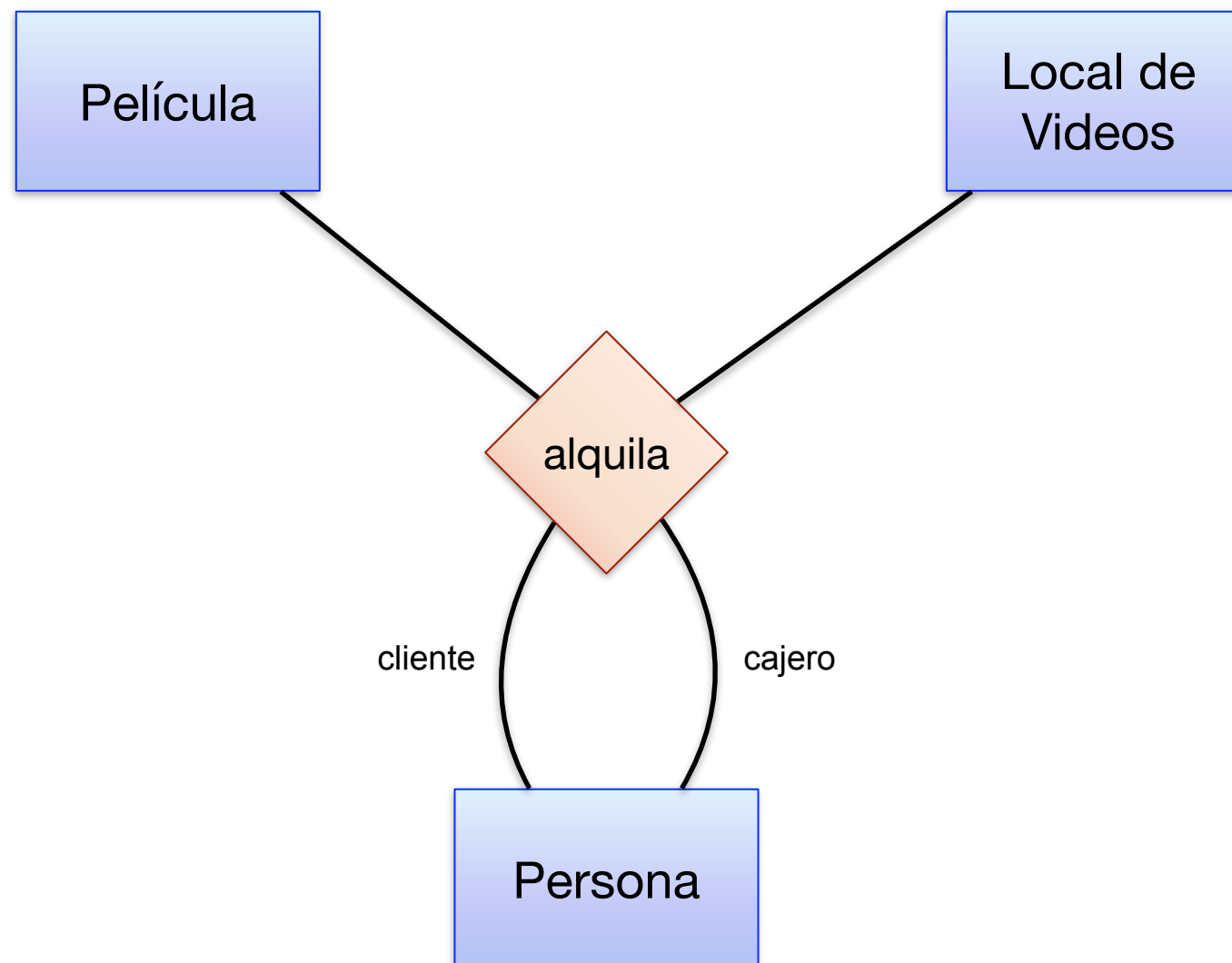
Más flexible



Más conciso

Relaciones Múltiples

Una entidad puede participar más de una vez en una relación

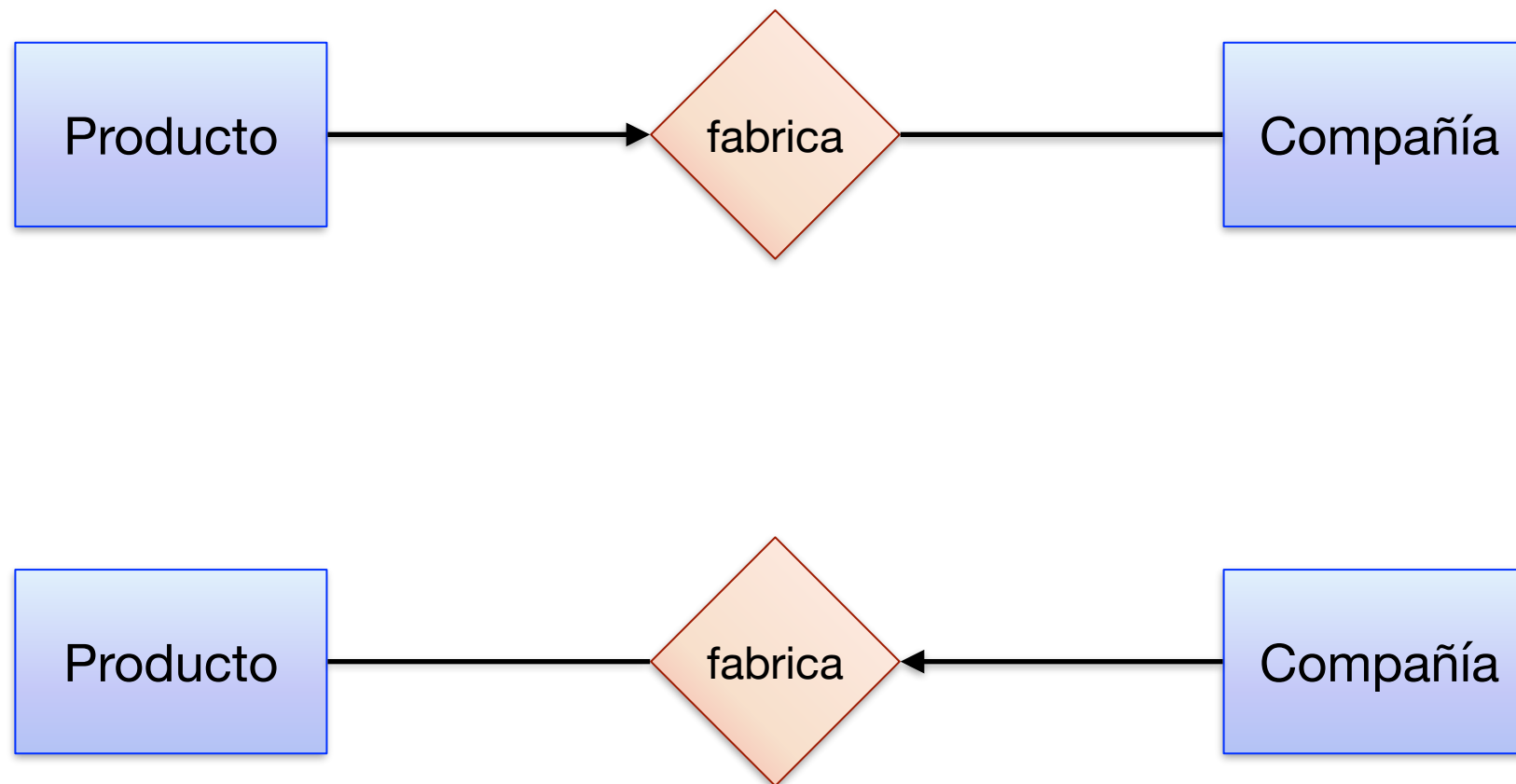


Diagramas E/R

Restricciones Avanzadas

Restricciones Avanzadas

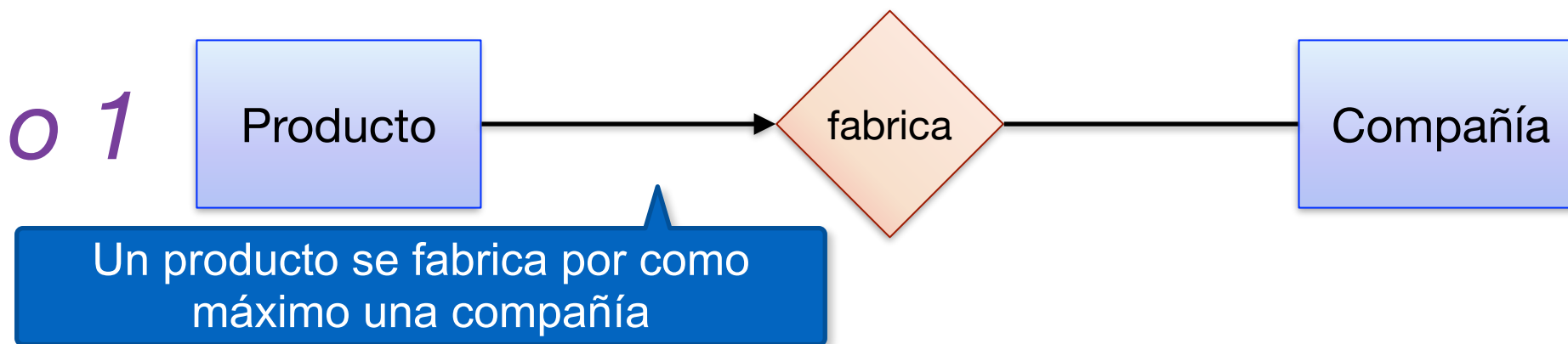
¿Que es más natural?



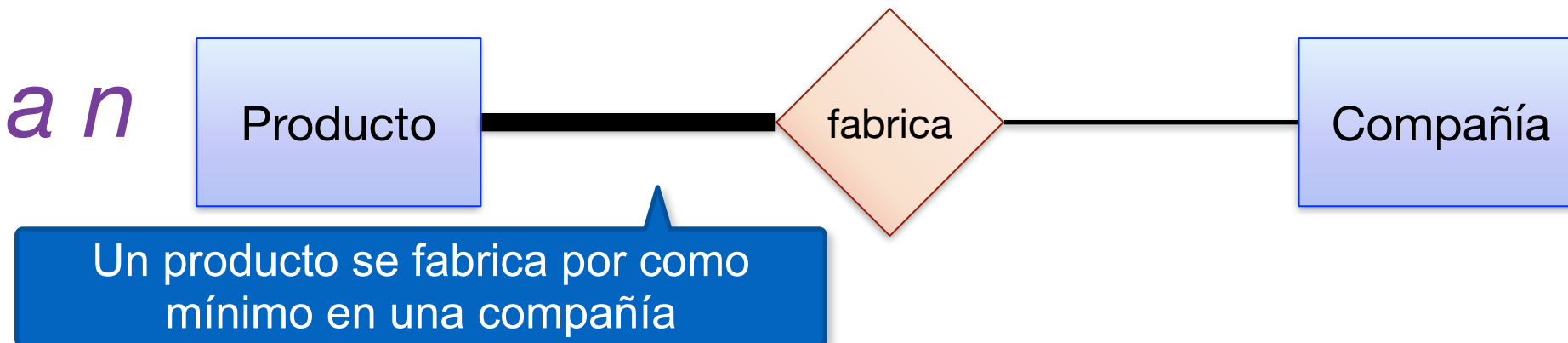
Restricciones Avanzadas

Participación y participación total

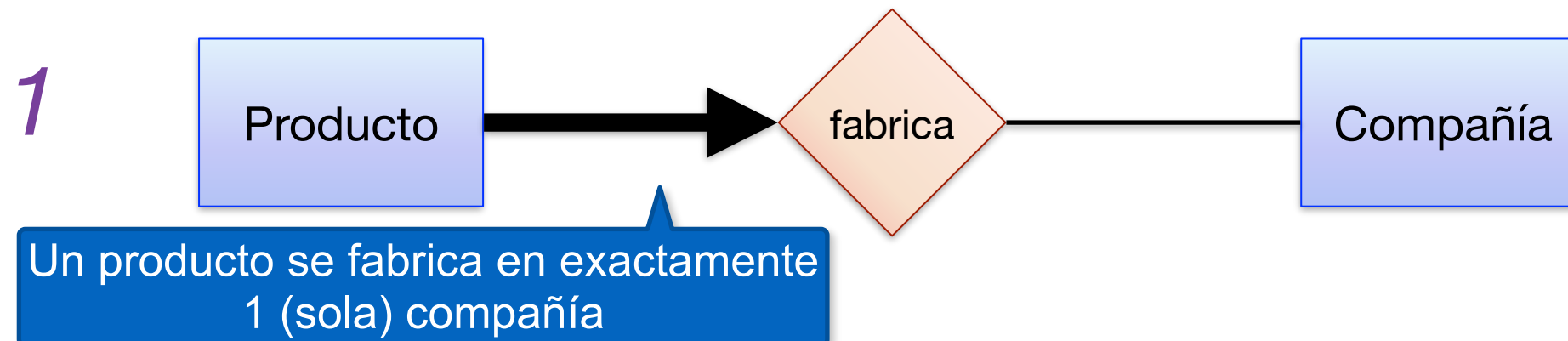
n a 0 o 1



n a 1 a n

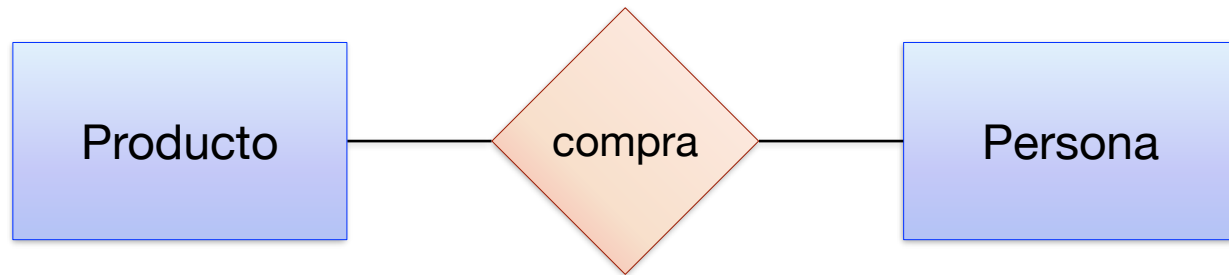


n a 1



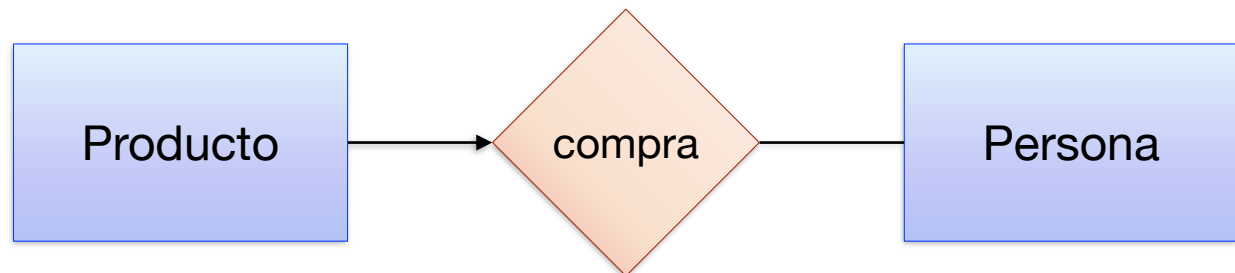
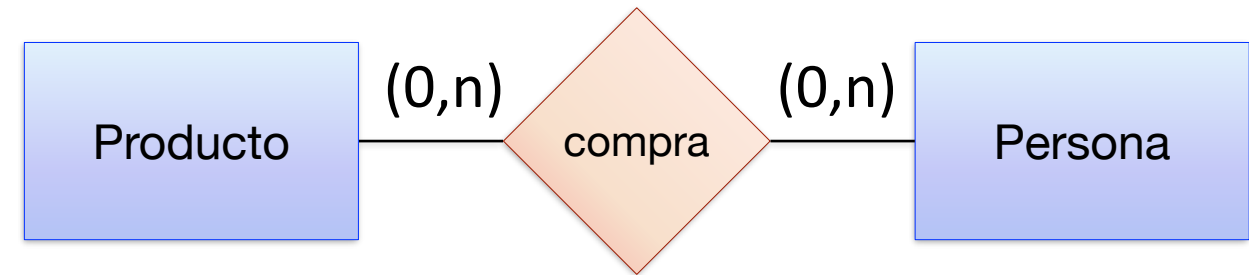
Equivalencia de multiplicidades

Ramakrishnan

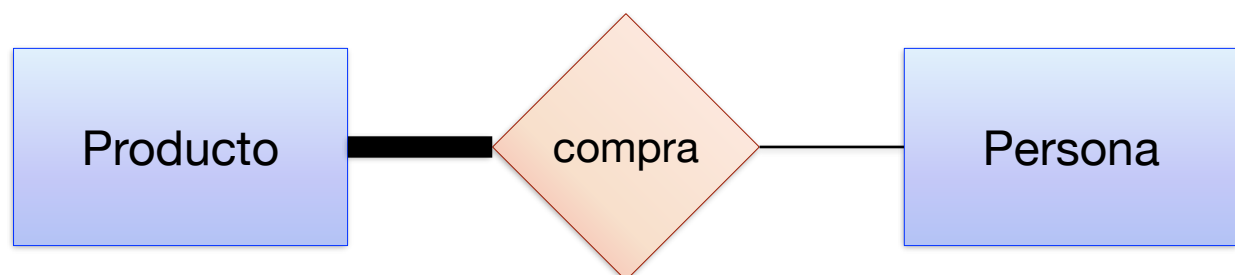
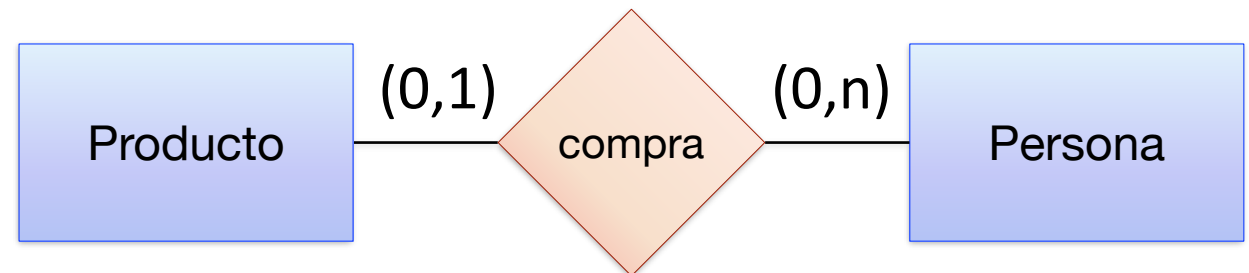


≡

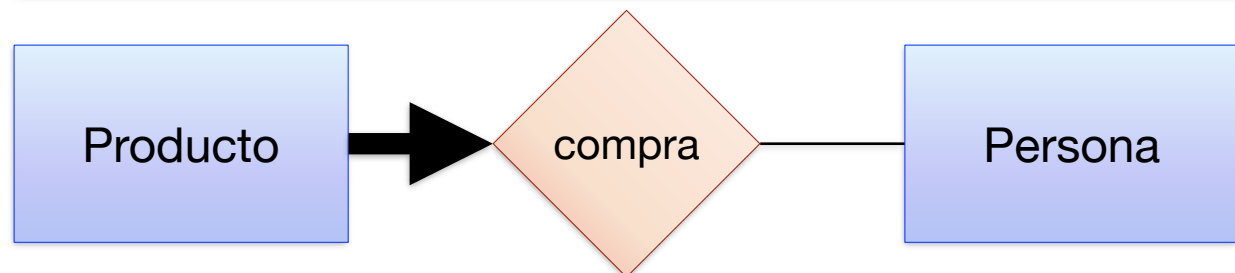
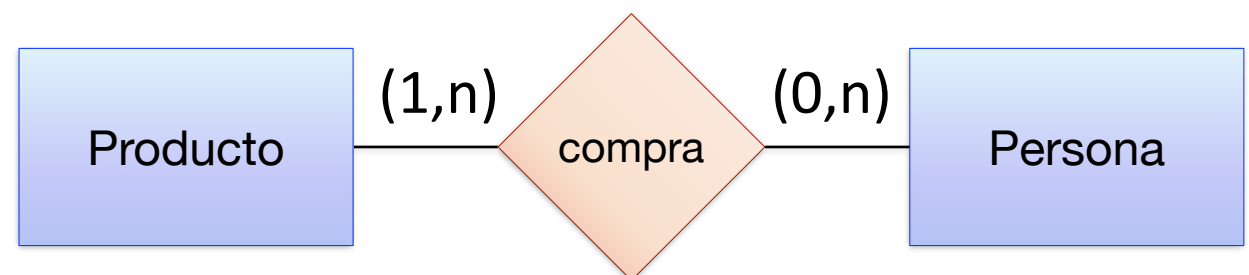
min-max



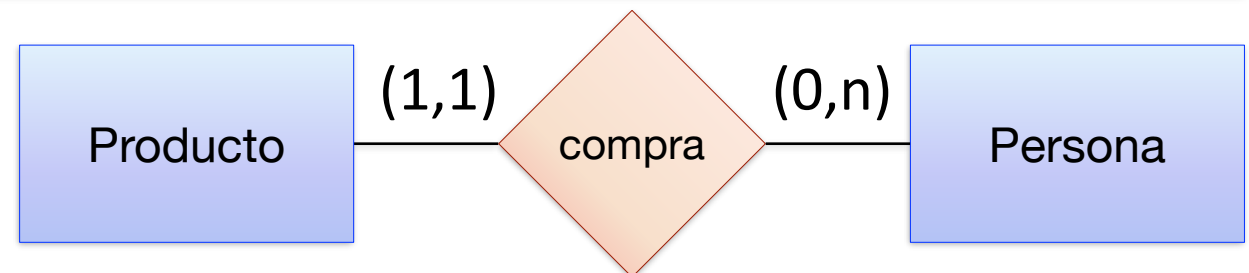
≡



≡



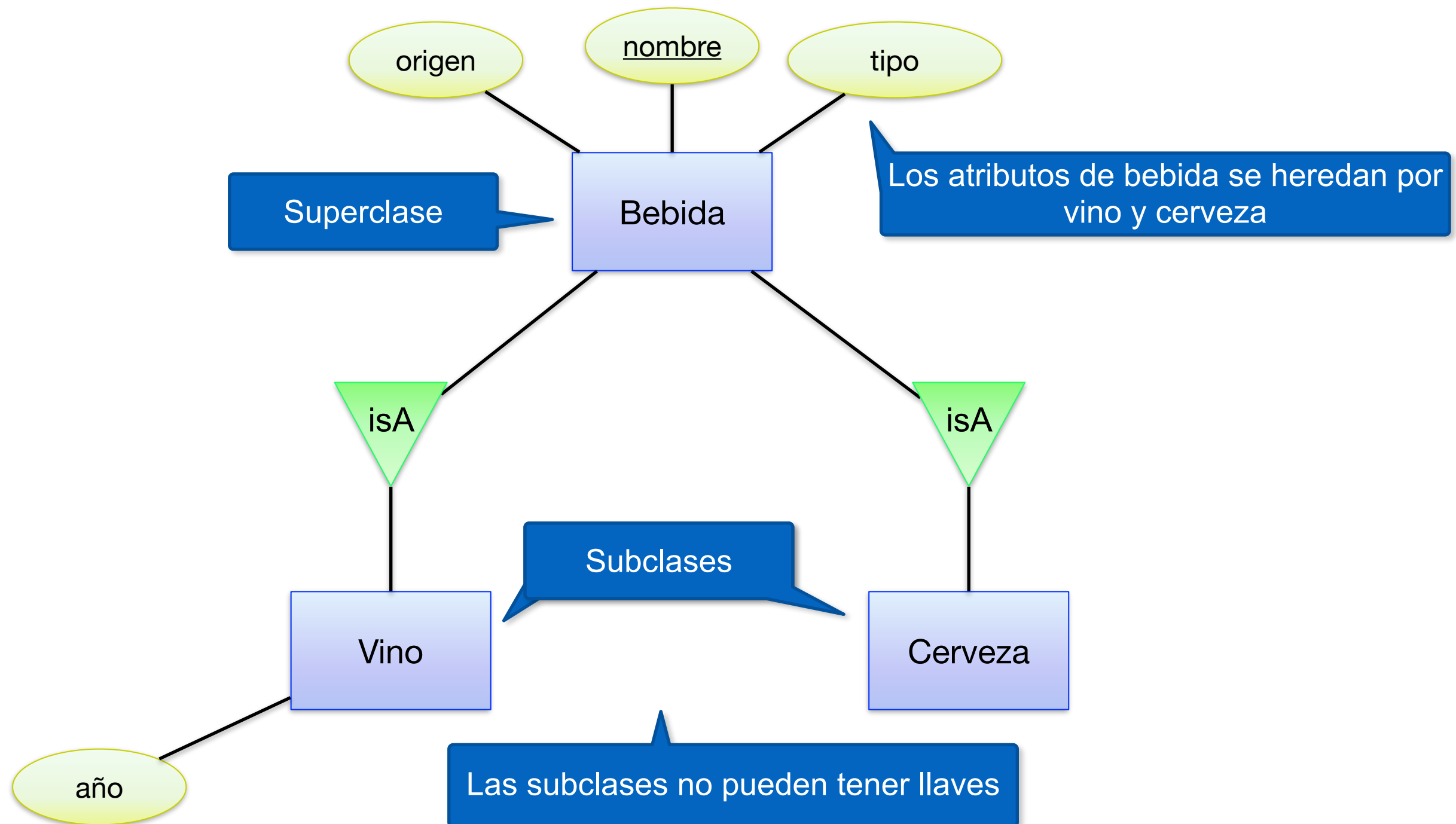
≡



Diagramas E/R

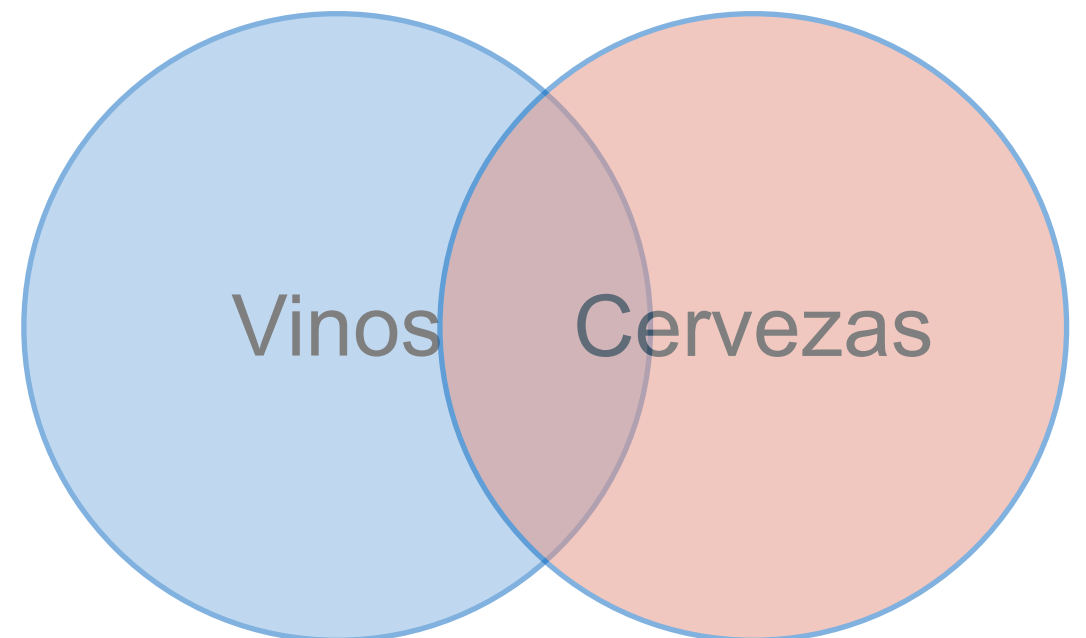
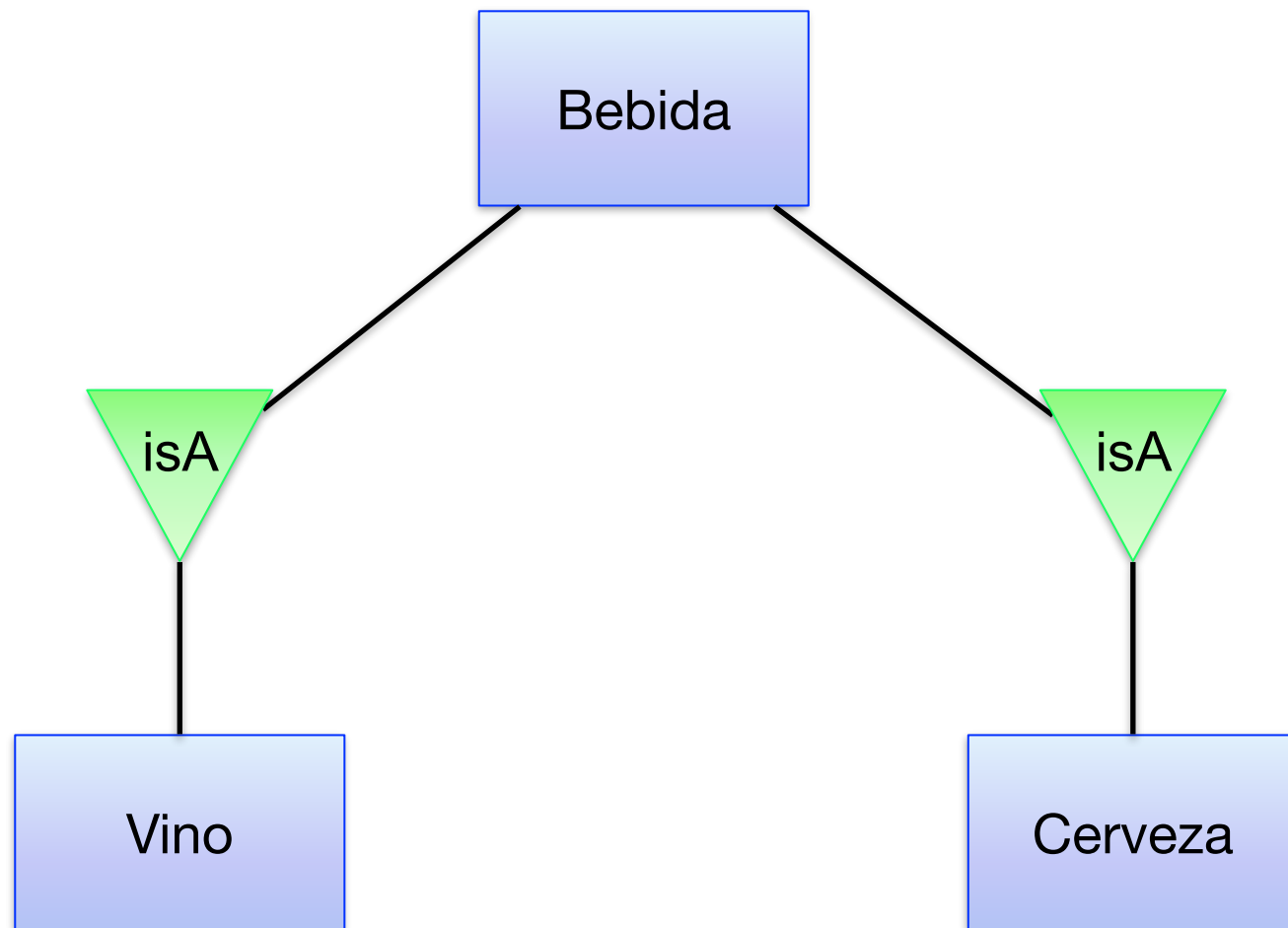
Jerarquía de clases

Jerarquía de clases



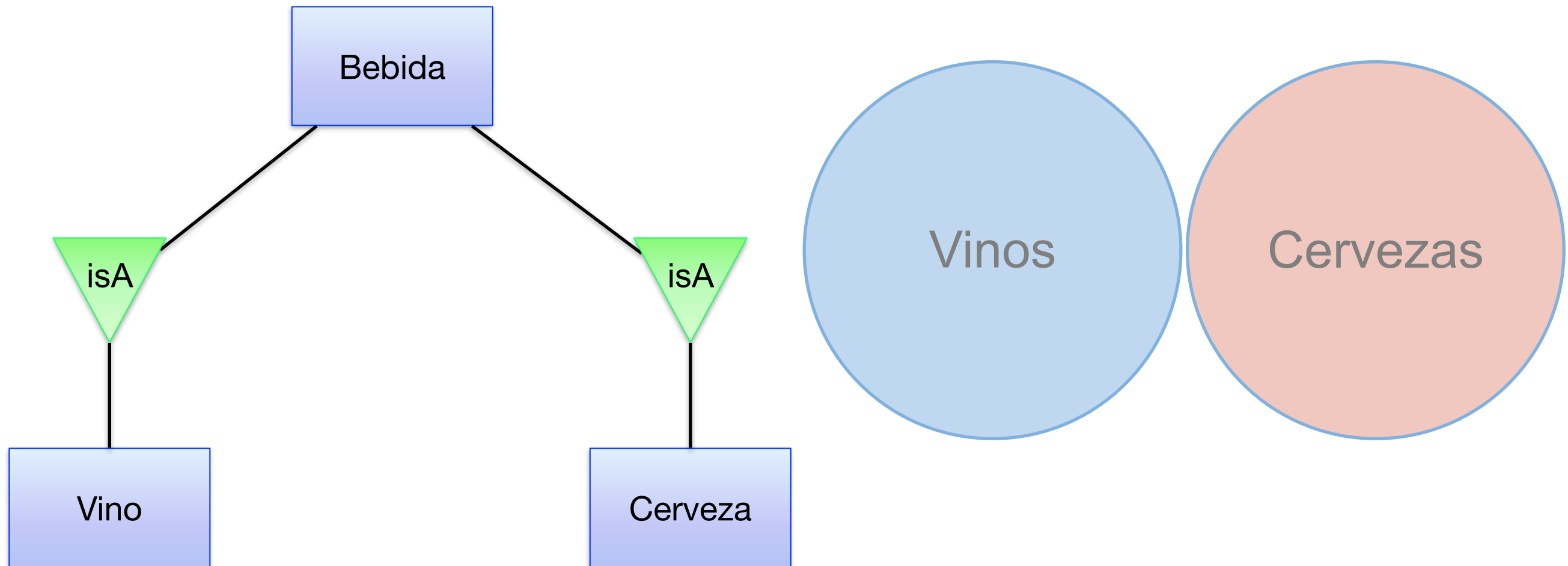
Jerarquía de clases

Solapamiento



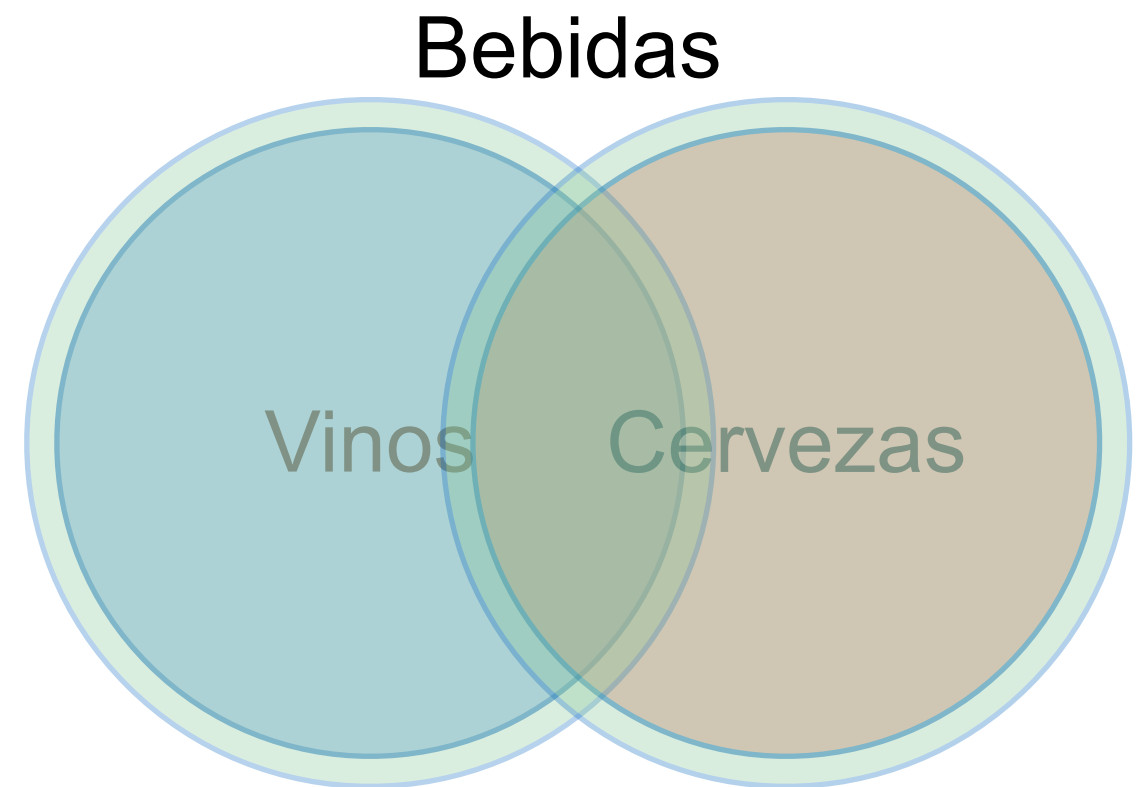
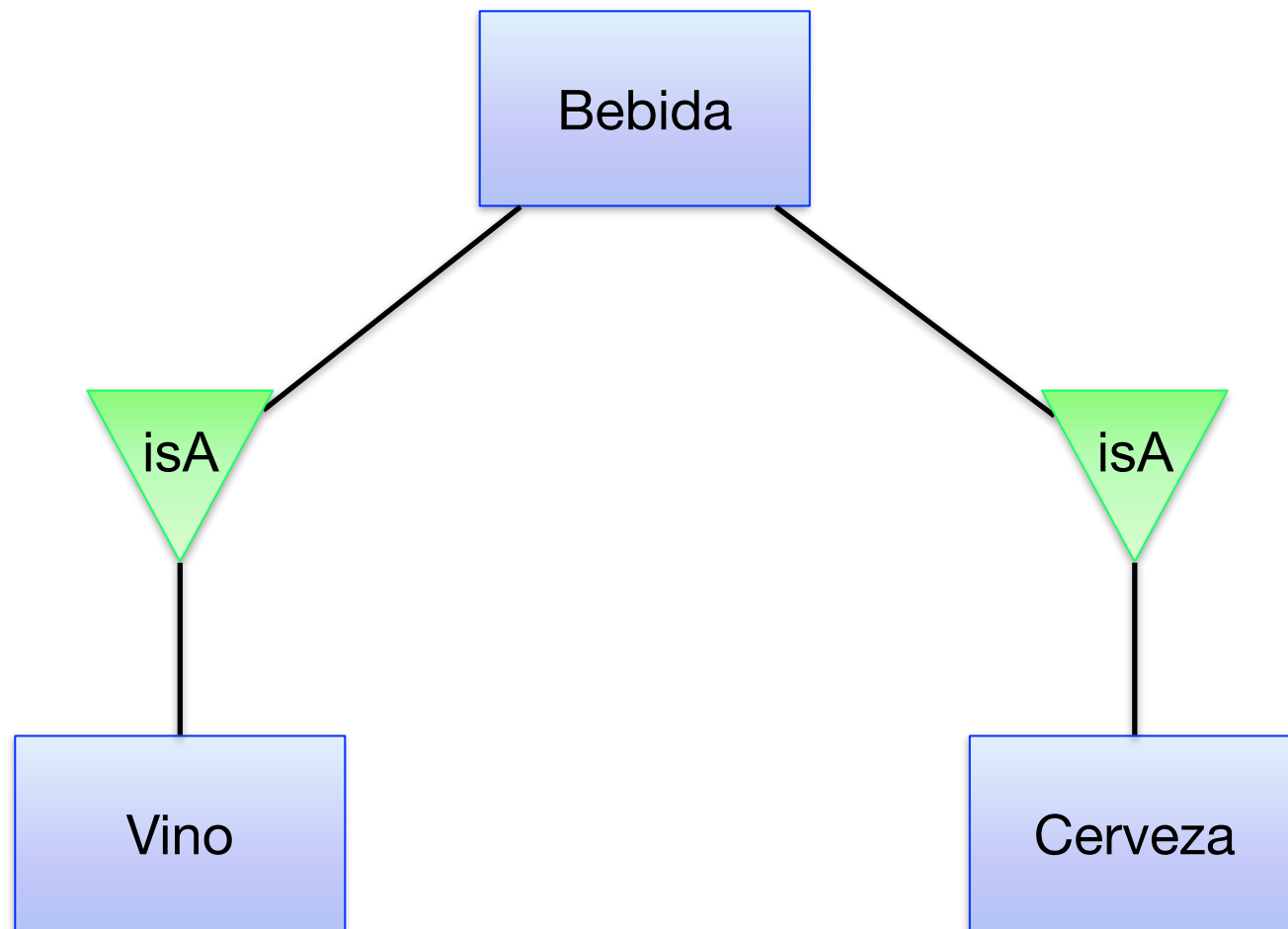
Jerarquía de clases

No Solapamiento



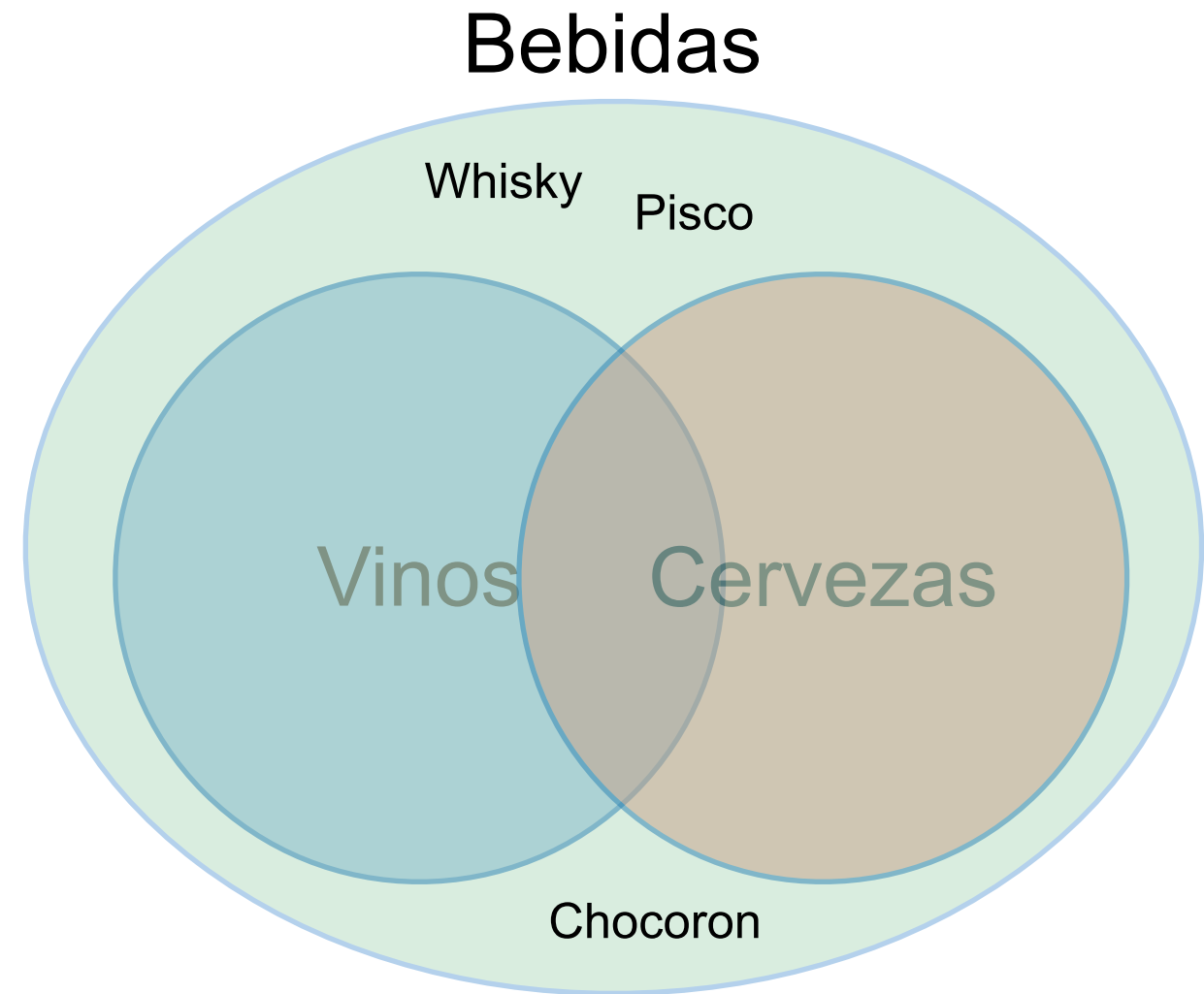
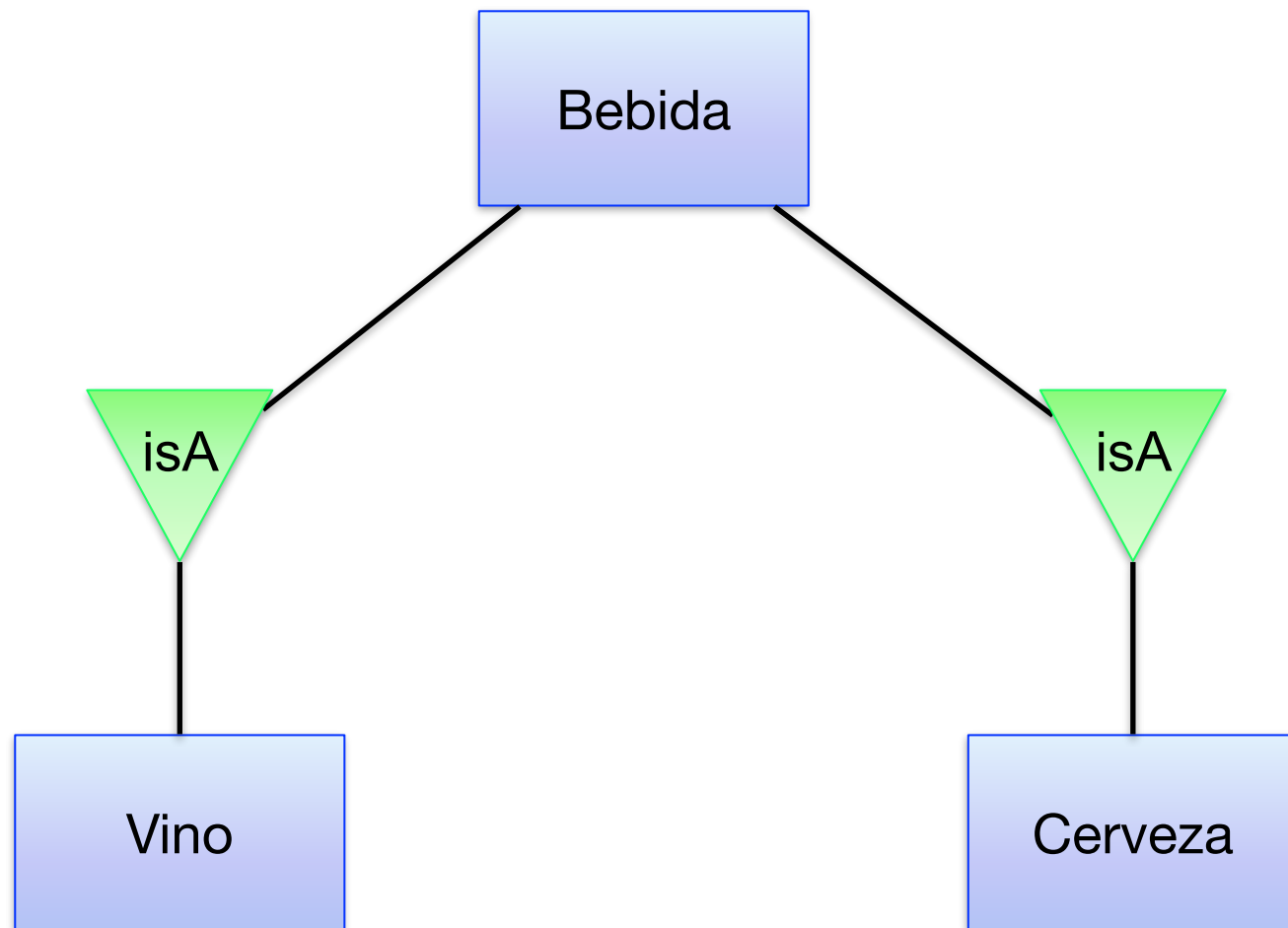
Jerarquía de clases

Cobertura



Jerarquía de clases

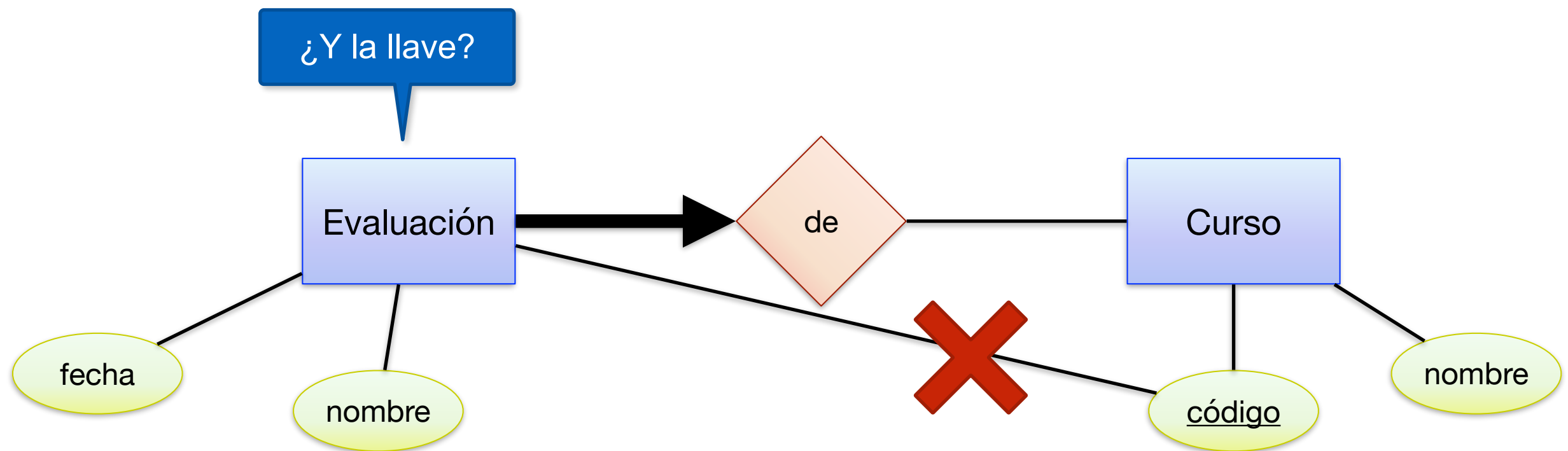
Sin Cobertura



Diagramas E/R

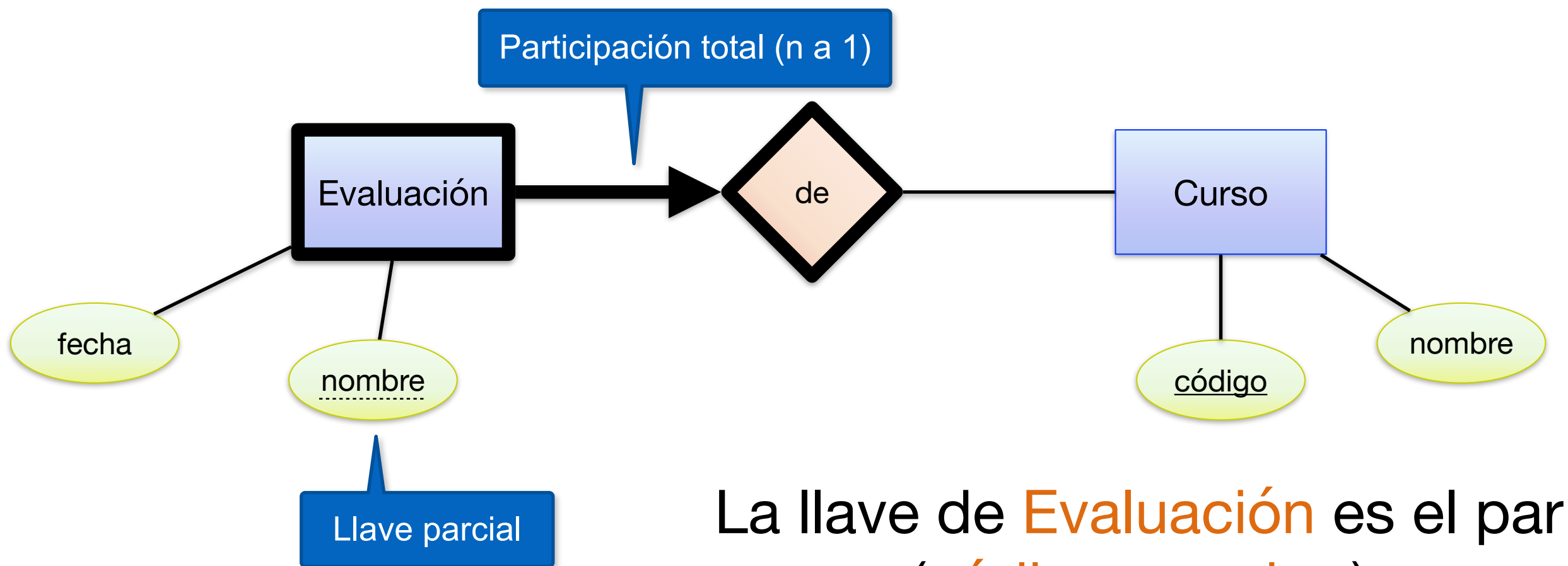
Entidades Débiles

Entidades Débiles



Entidades Débiles

...entidades cuya llave dependa de la llave de otra entidad

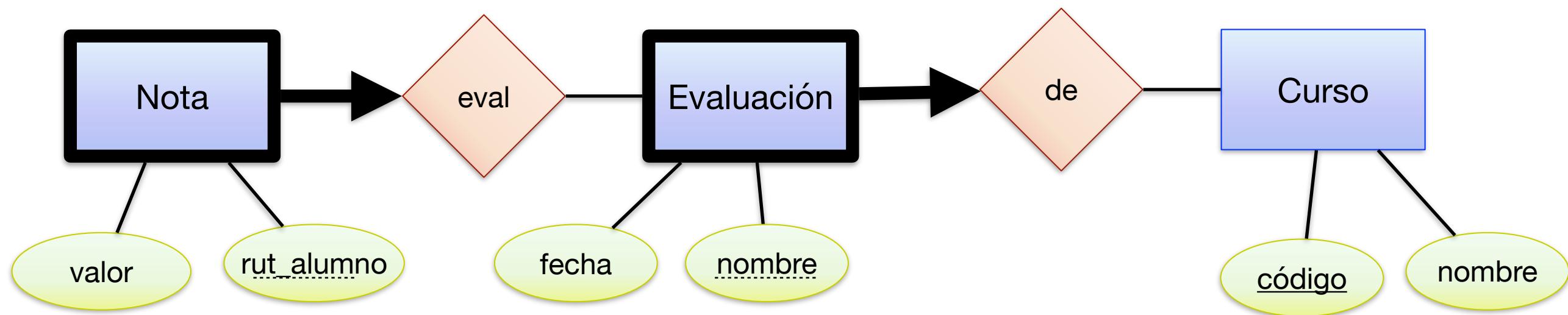


La llave de **Evaluación** es el par
(**código**, **nombre**)

¿Y las notas?...

Entidades Débiles

Podemos encadenar entidades débiles



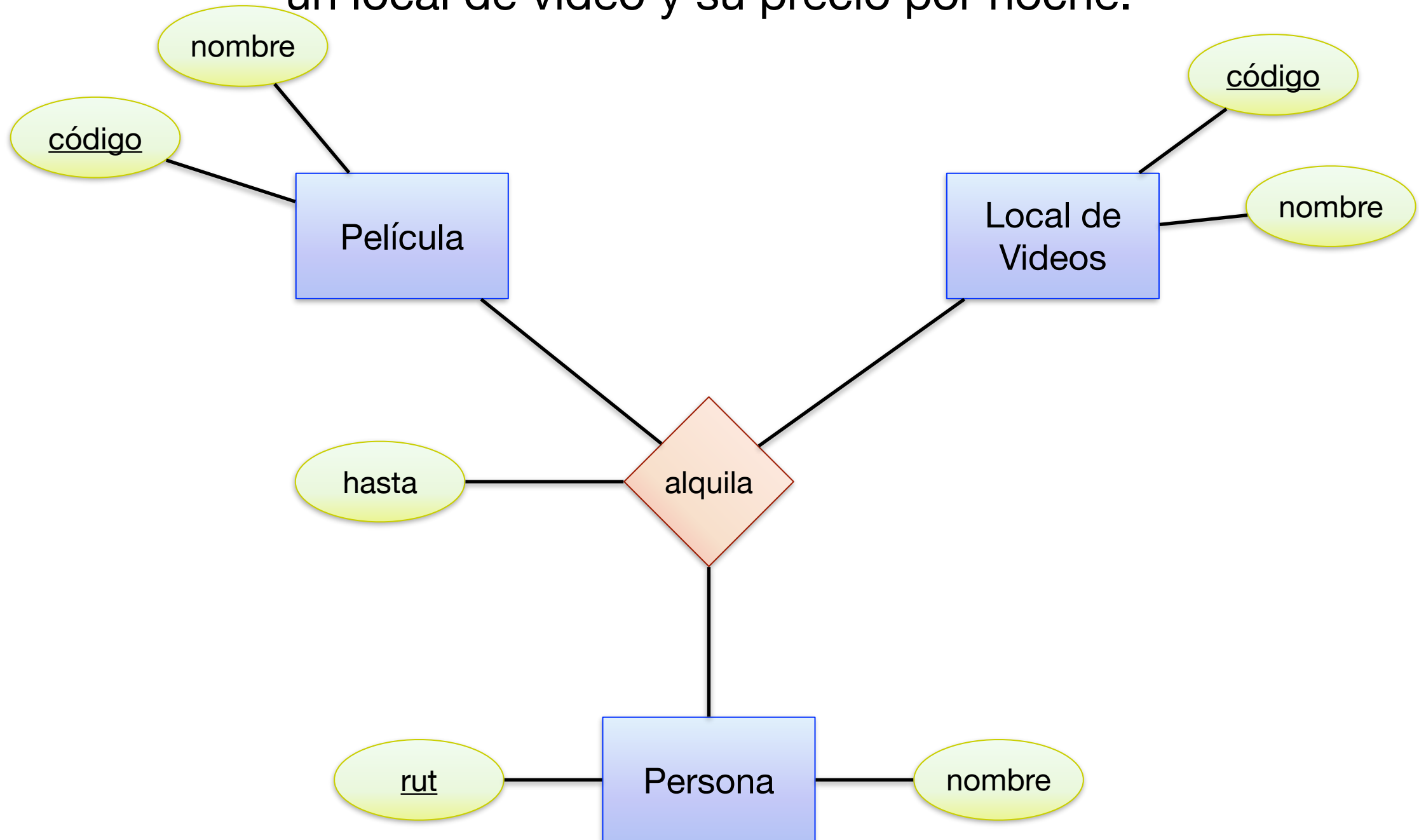
La llave de **Nota** es la tupla
(código, nombre, rut_alumno)

Diagramas E/R

Agregación

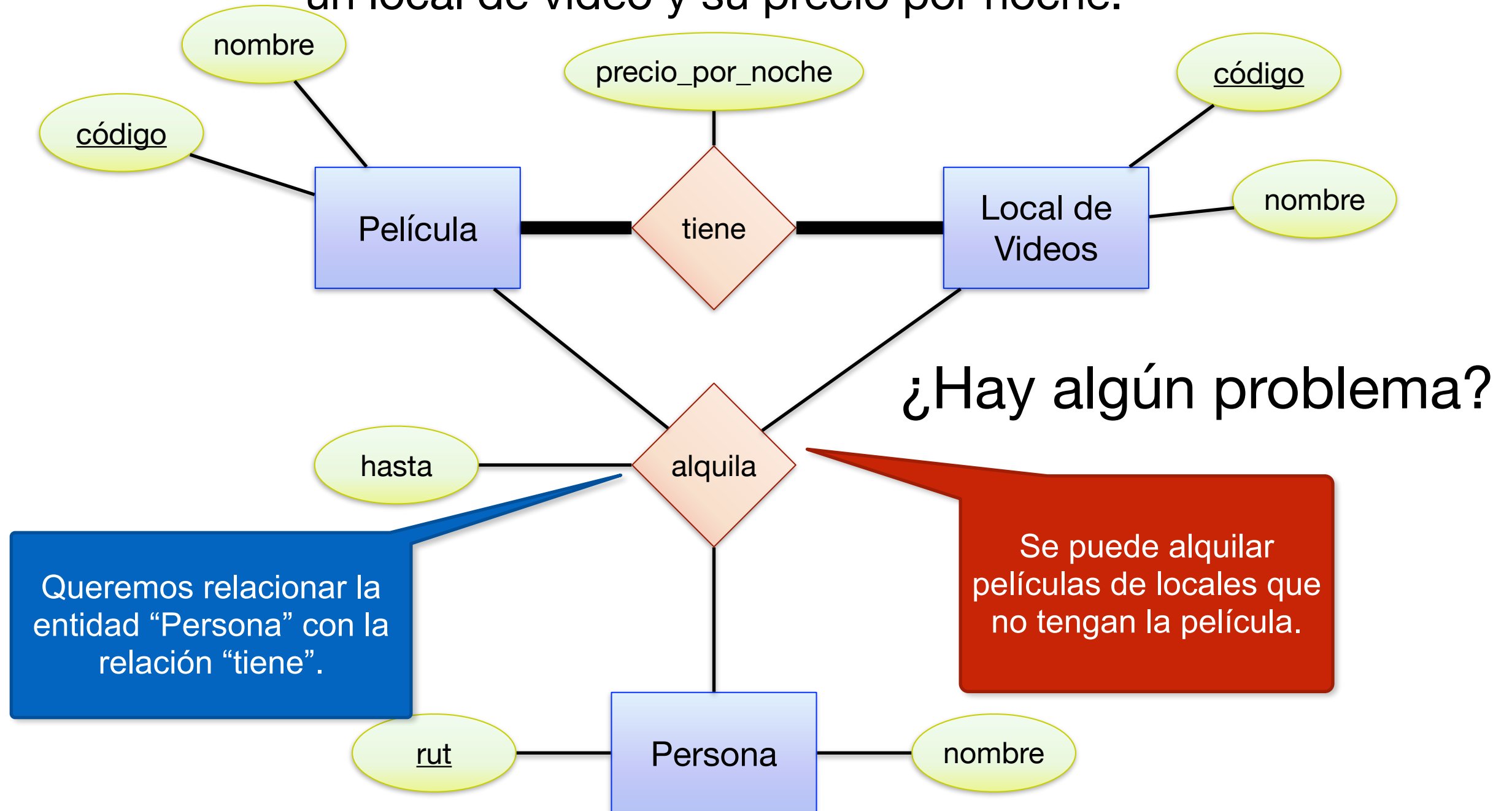
Agregación

Queremos registrar las películas que posee un local de video y su precio por noche.



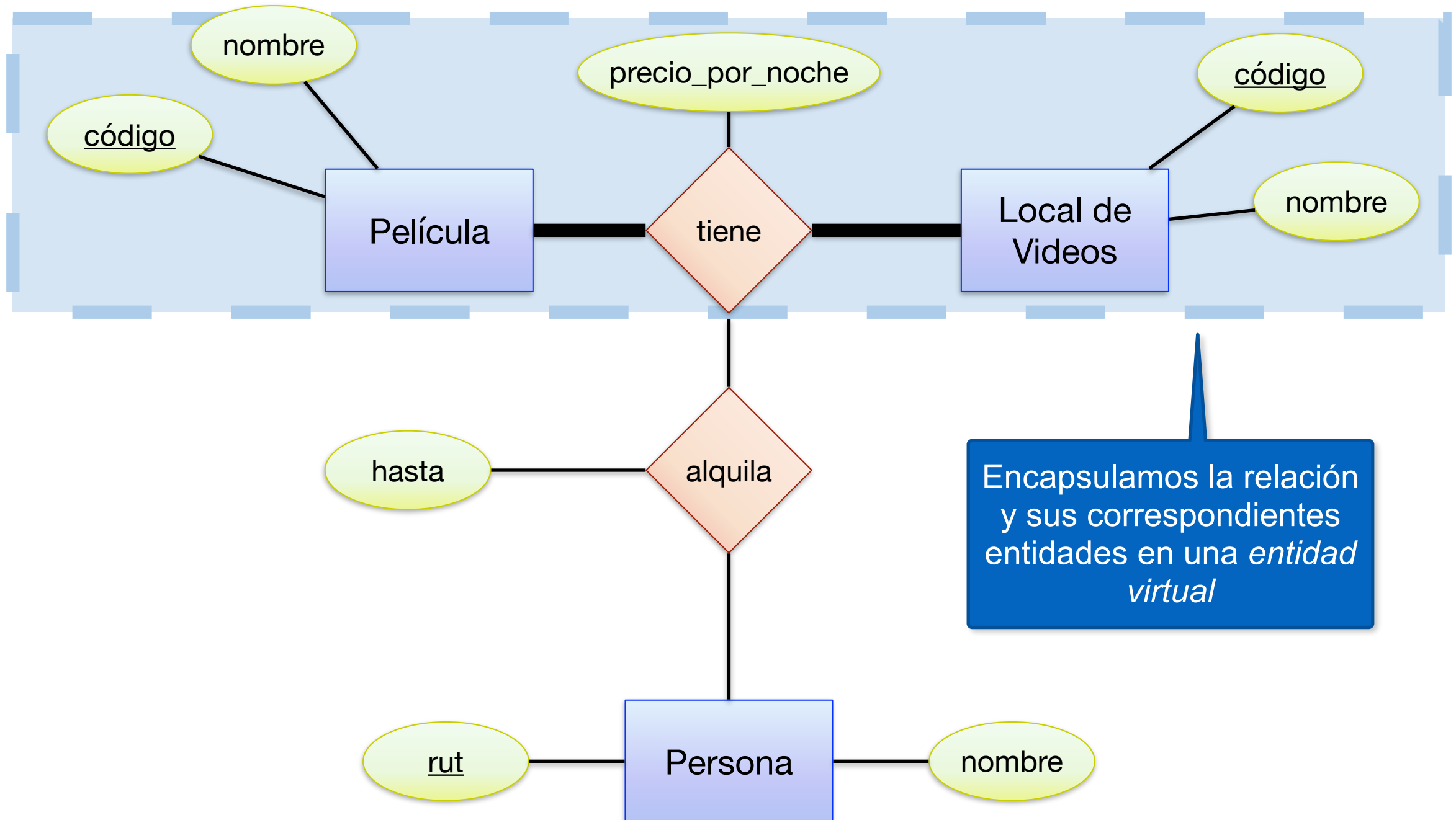
Agregación

Queremos registrar las películas que posee un local de video y su precio por noche.



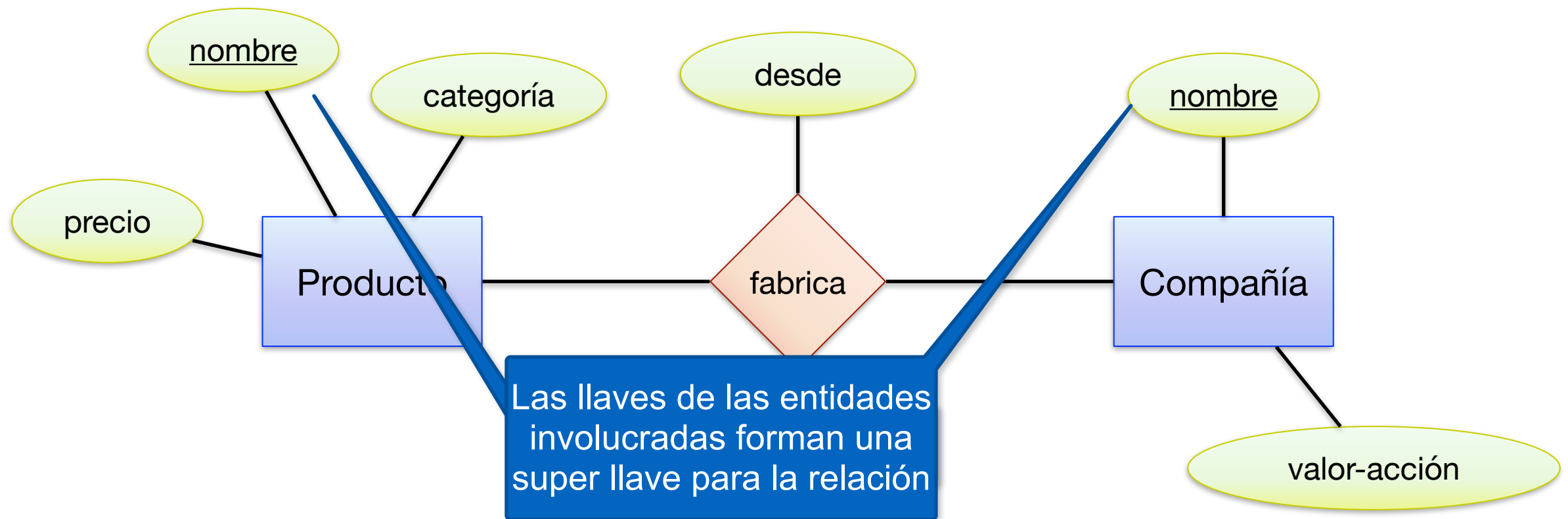
Agregación

Encapsulando relaciones



Del Diagrama E/R al Modelo Relacional

M. E/R → M. Relacional

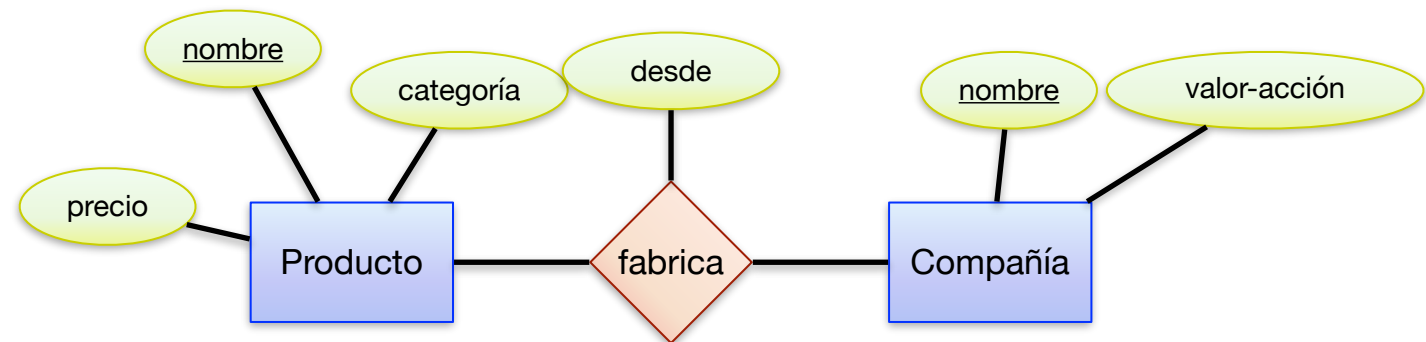


Producto(nombre: string, precio: int, categoría: string)

Compañía(nombre: string, valor-acción: int)

Fabrica(Producto.nombre: string, Compañía.nombre, desde: date)

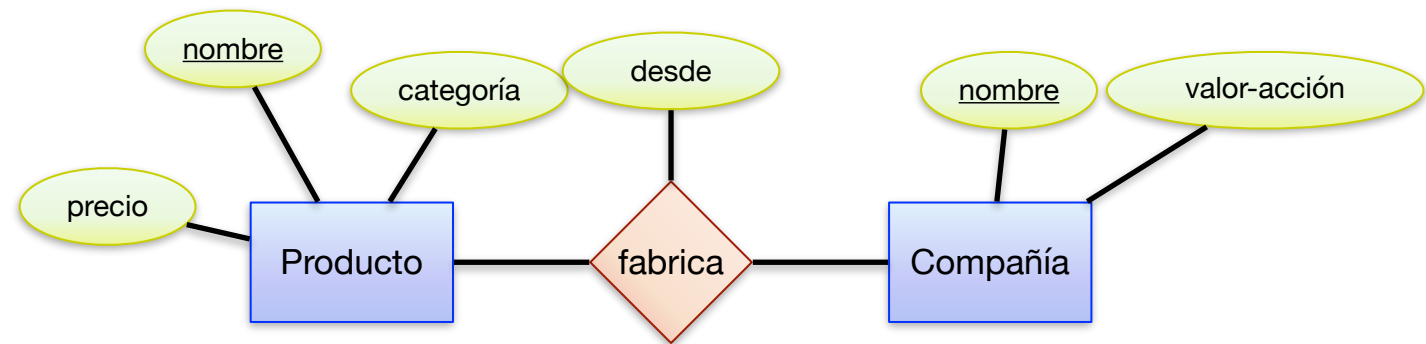
M. E/R → M. Relacional



Producto(nombre: string, precio: int, categoría: string)

```
CREATE TABLE producto(  
    nombre varchar(30),  
    precio int,  
    categoria varchar(30),  
    PRIMARY KEY (nombre)  
)
```

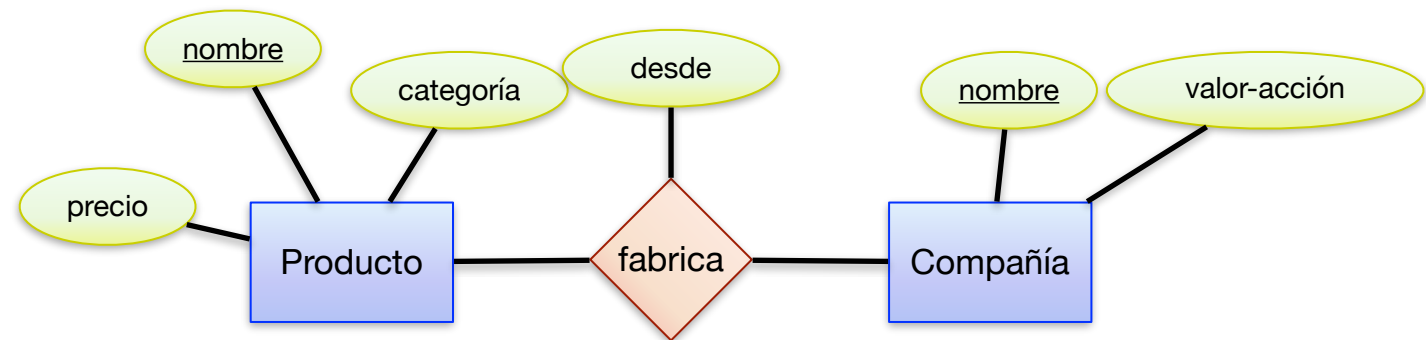
M. E/R → M. Relacional



Compañía(nombre: string, valor-acción: int)

```
CREATE TABLE compania(  
    nombre varchar(30),  
    valor_accion int,  
    PRIMARY KEY (nombre)  
)
```

M. E/R → M. Relacional



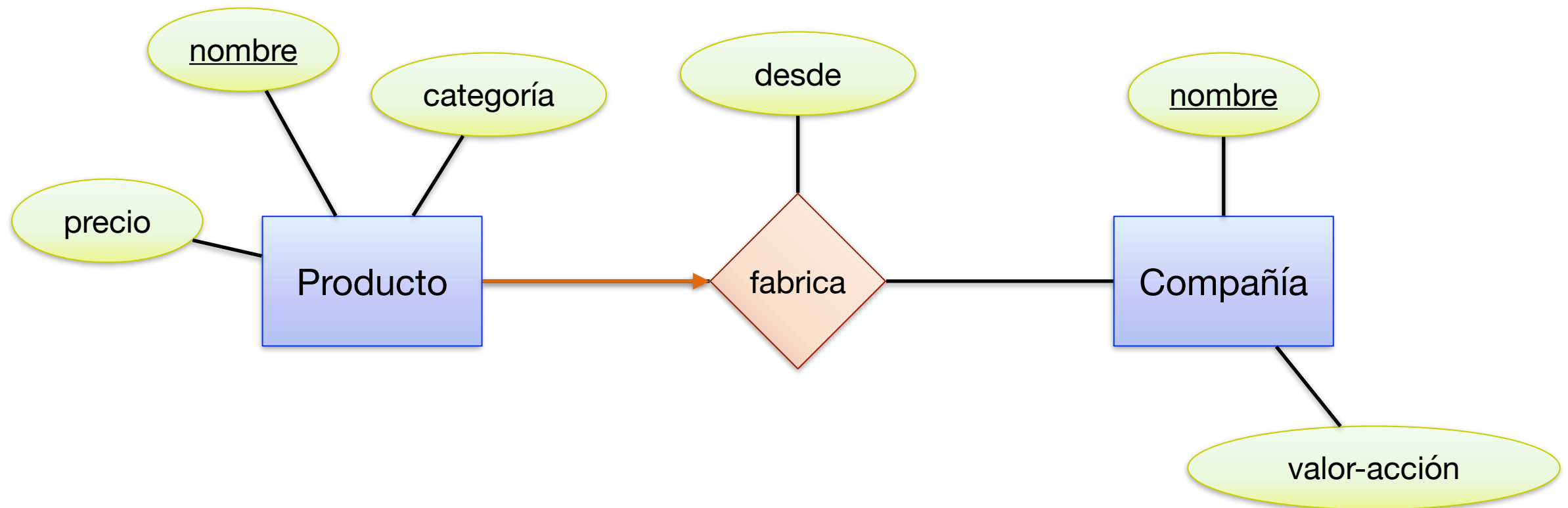
Fabrica(Producto.nombre: string, Compañía.nombre, desde: date)

```
CREATE TABLE fabrica(  
    p_nombre varchar(30),  
    c_nombre varchar(30),  
    desde date,  
    PRIMARY KEY (p_nombre, c_nombre),  
    FOREIGN KEY(p_nombre) REFERENCES producto(nombre),  
    FOREIGN KEY(c_nombre) REFERENCES compania(nombre)  
)
```

Llaves foráneas

M. E/R → M. Relacional

¿Qué pasa aquí?



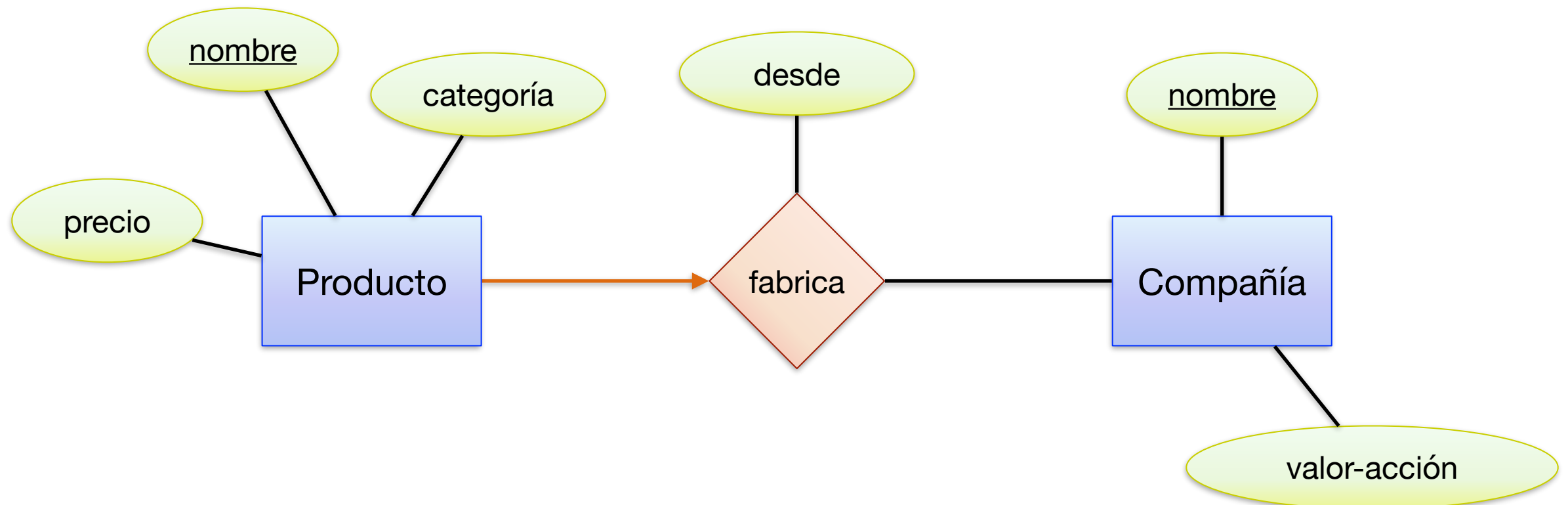
Producto(nombre: string, precio: int, categoría: string)

Compañía(nombre: string, valor-acción: int)

Fabrica(Producto.nombre: string, Compañía.nombre, desde: date)

M. E/R → M. Relacional

¿Qué pasa aquí?



Producto(nombre: string, precio: int, categoría: string)

Compañía(nombre: string, valor-acción: int)

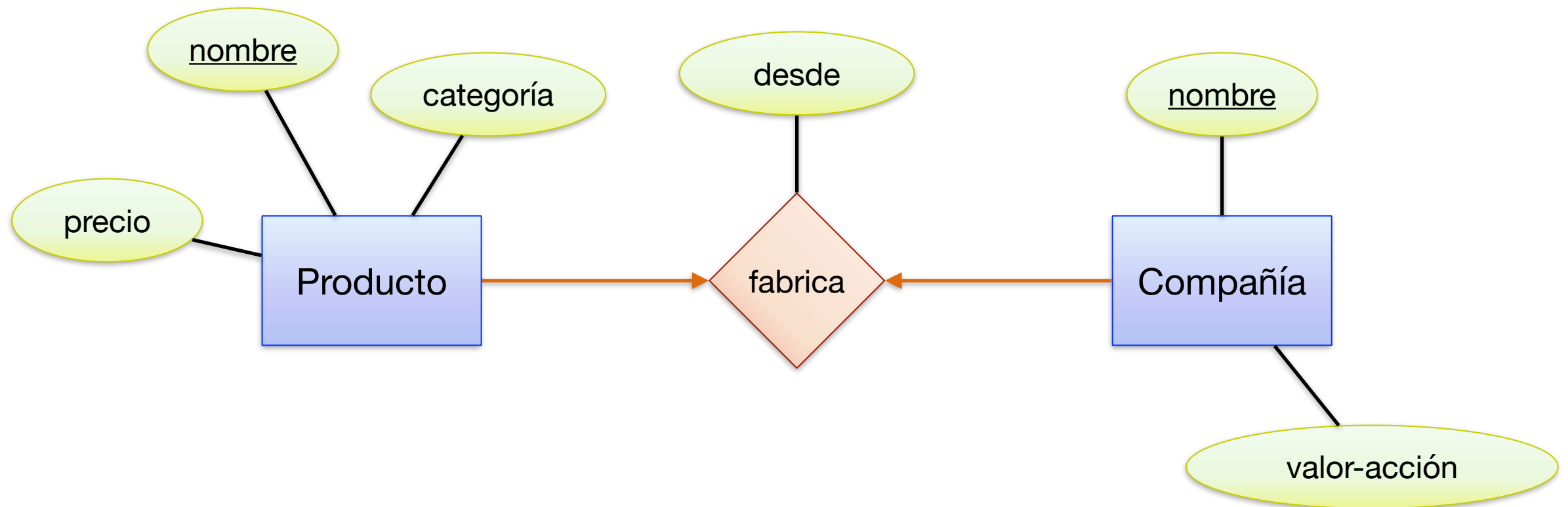
Fabrica(Producto.nombre: string, Compañía.nombre, desde: date)

Producto.nombre forma una llave
candidata

No se necesita que
Compañía.nombre sea llave

M. E/R → M. Relacional

¿Y ahora?



Producto(nombre: string, precio: int, categoría: string)

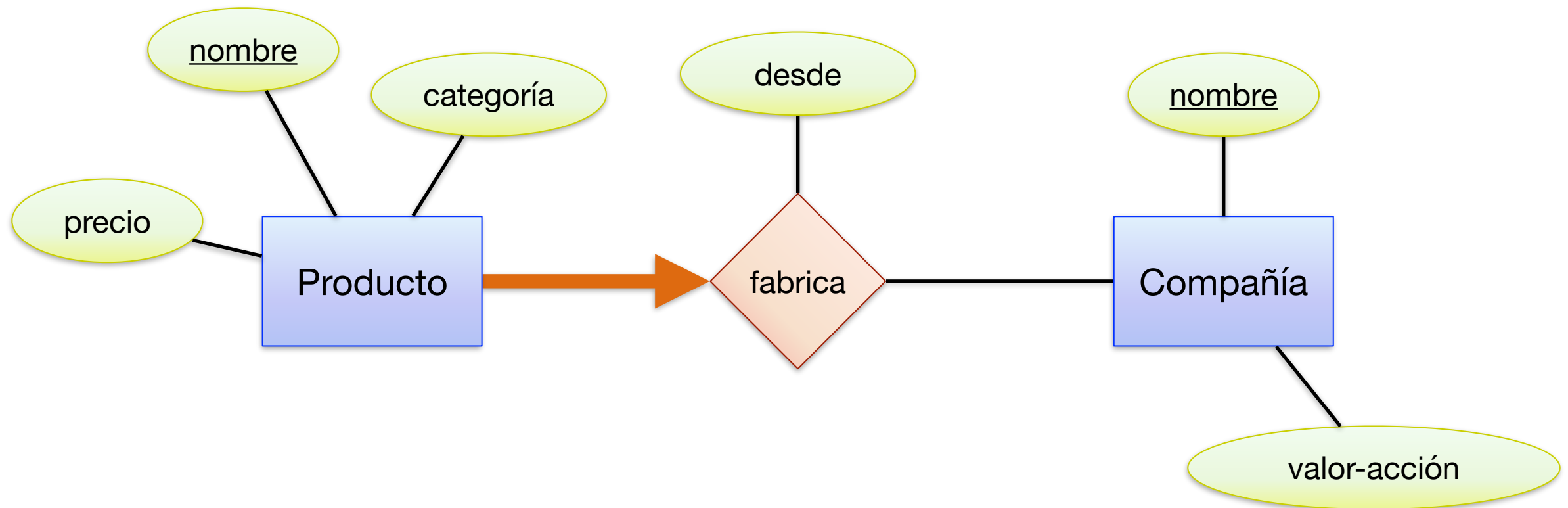
Compañía(nombre: string, valor-acción: int)

Fabrica(Producto.nombre: string, Compañía.nombre, desde: date)

Podemos hacer llave a Producto.nombre o
a Compañía.nombre

M. E/R → M. Relacional

¿Y ahora?



Producto(nombre: string, precio: int, categoría: string,
Compañía.nombre: string, desde: date)

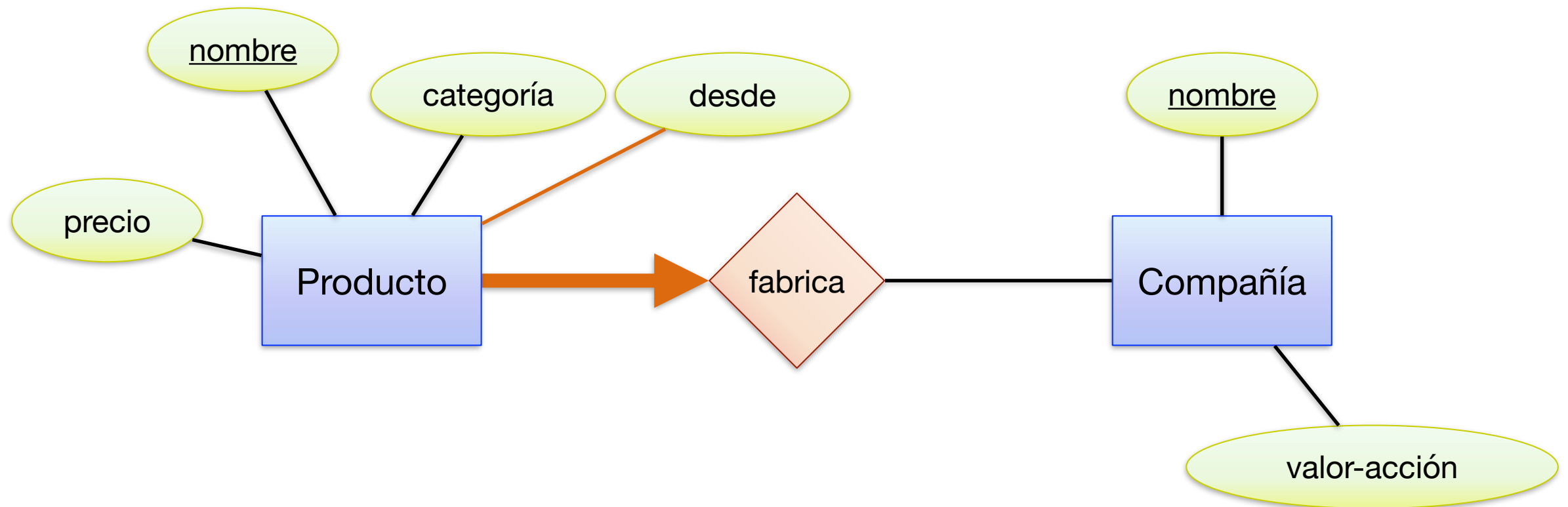
Compañía(nombre: string, valor-acción: int)

Sólo necesitamos una llave foránea en Producto.

Agregamos también el atributo de la relación.

M. E/R → M. Relacional

Un mejor diagrama



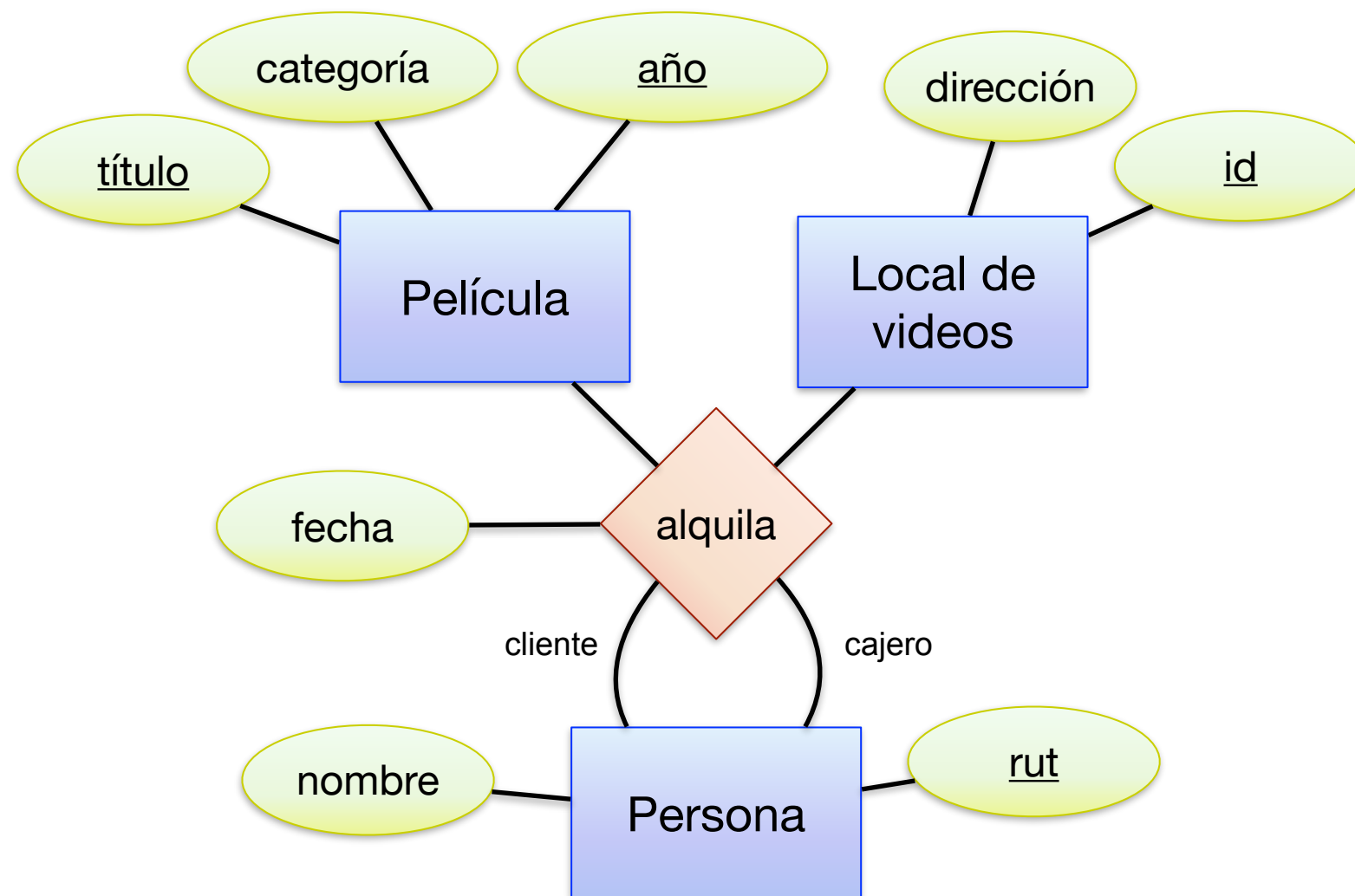
Producto(nombre: string, precio: int, categoría: string,
Compañía.nombre: string, desde: date)

Compañía(nombre: string, valor-acción: int)

Sólo necesitamos una llave foránea en Producto.

Agregamos también el atributo de la relación.

M. E/R → M. Relacional



Pelicula(titulo: string, año: int, categoría: string)

Local de videos(id: int, dirección: string)

Persona(rut: string, nombre: string)

Alquila(Pl.titulo: string, Pl.año: int, Pr.rut-cl: string, Pr.rut-ca: string, L.id: int, fecha: date)

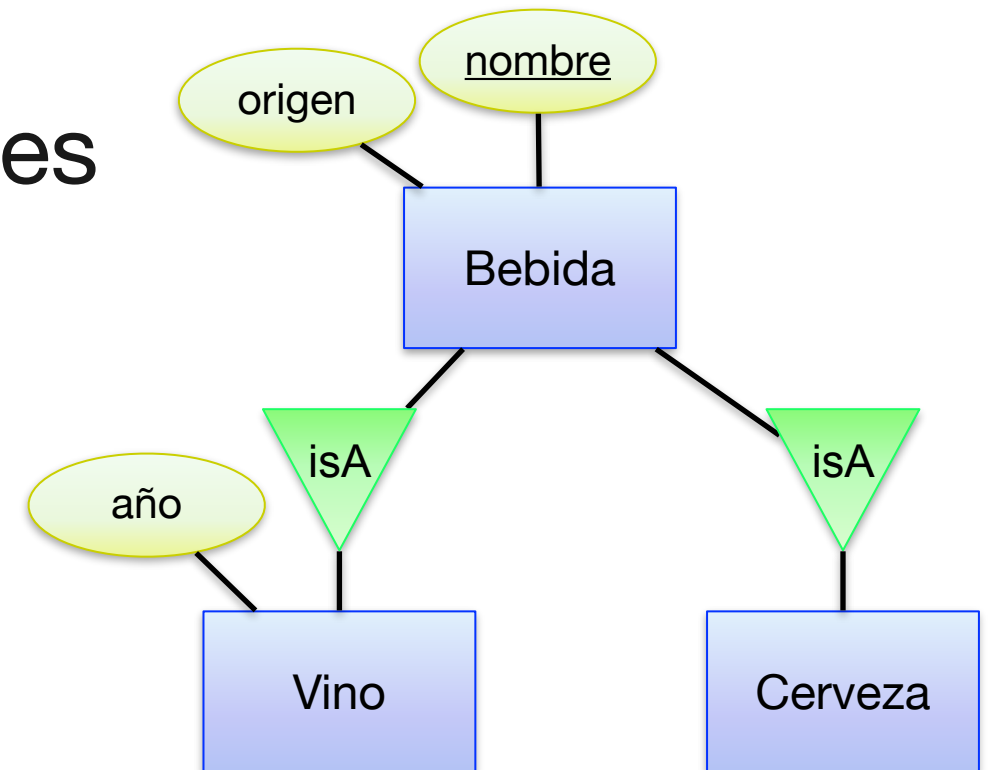
M. E/R → M. Relacional

Jerarquía de clases

Opción 1: Tablas solo para las subclases

Vino(nombre: string, origen: string, año: string)

Cerveza(nombre: string, origen: string)



Opción 2: Tabla para la superclase

Bebida(nombre: string, origen: string)

Vino(nombre: string, año: string)

Cerveza(nombre: string)

Se requieren joins para acceder a todos los datos

¿Cuál es mejor?

Si hay mucho solapamiento: opción 2.
De lo contrario tendríamos mucha repetición de datos.

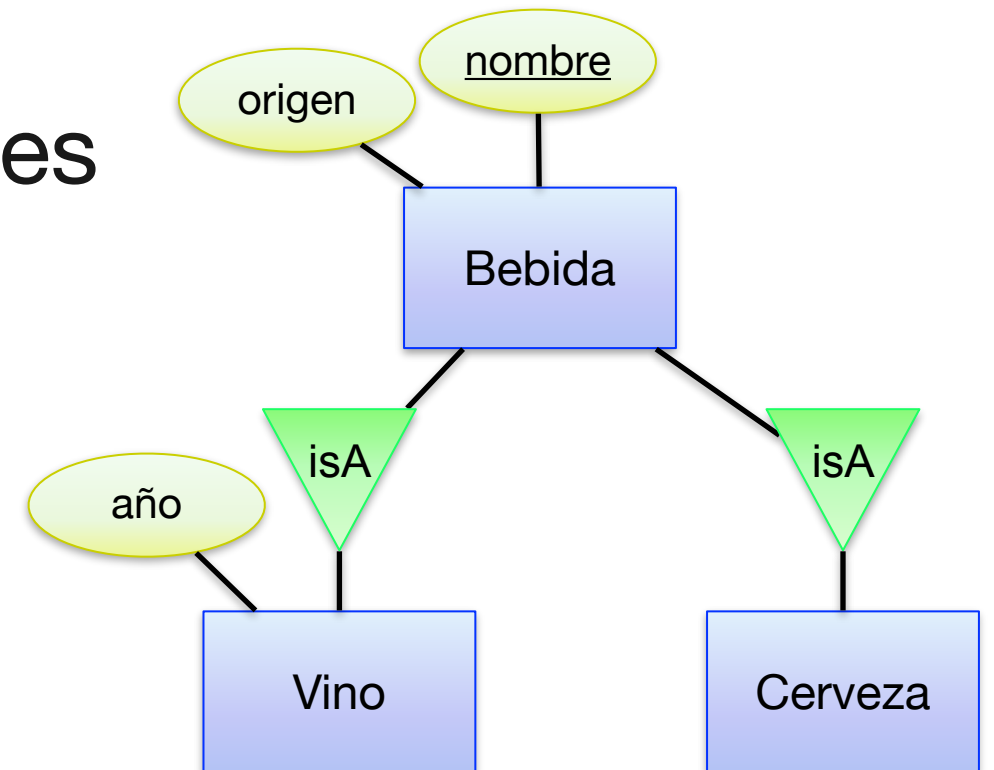
M. E/R → M. Relacional

Jerarquía de clases

Opción 1: Tablas solo para las subclases

Vino(nombre: string, origen: string, año: string)

Cerveza(nombre: string, origen: string)



Opción 2: Tabla para la superclase

Bebida(nombre: string, origen: string)

Vino(nombre: string, año: string)

Cerveza(nombre: string)

Se requieren joins para acceder a todos los datos

¿Cuál es mejor?

Si no hay cobertura: opción 2.
No hay otra opción o no podríamos guardar el whisky :(

M. E/R → M. Relacional

Jerarquía de clases

Opción 1: Tablas solo para las subclases

Vino(nombre: string, origen: string, año: string)

Cerveza(nombre: string, origen: string)

Opción 2: Tabla para la superclase

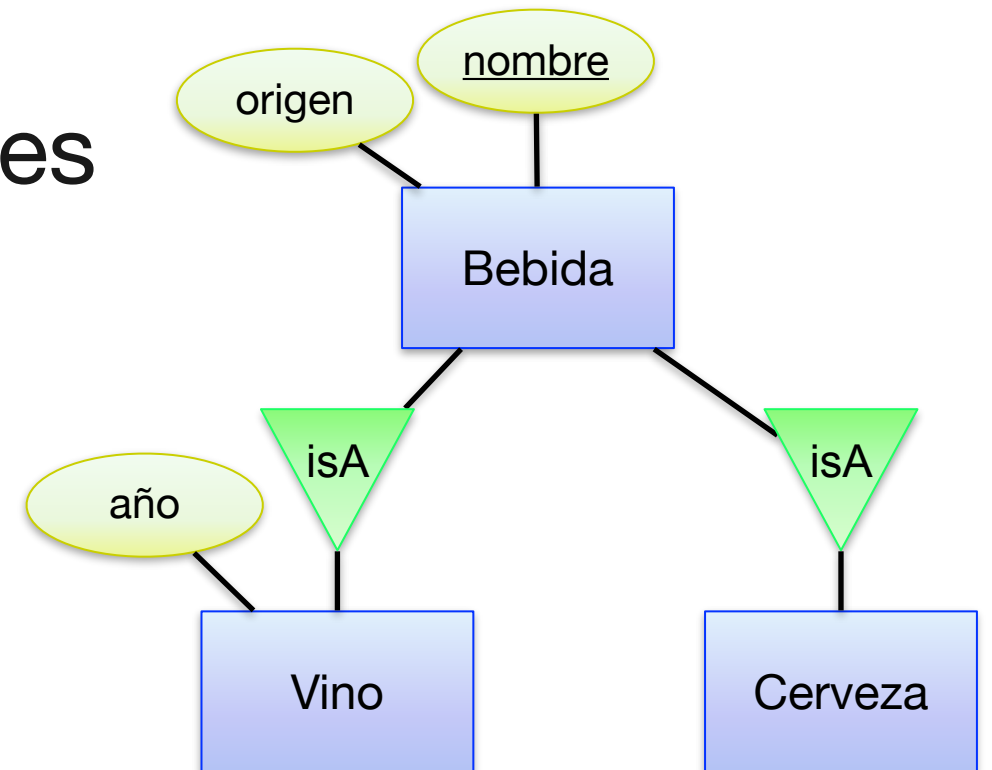
Bebida(nombre: string, origen: string)

Vino(nombre: string, año: string)

Cerveza(nombre: string)

¿Cuál es mejor?

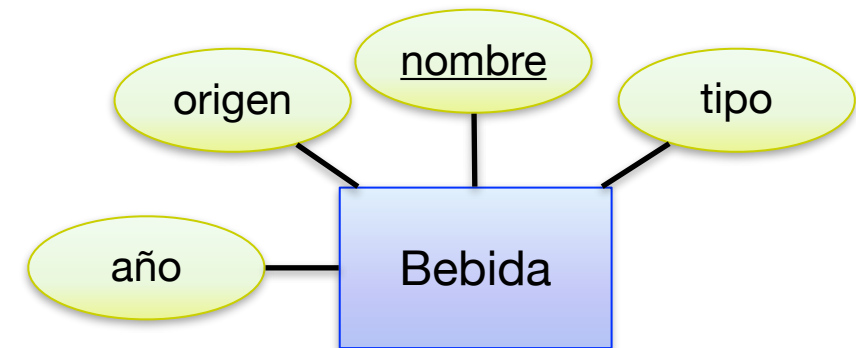
Se requieren joins para acceder a todos los datos



Si hay muchas consultas por **nombre**: opción 2.
Con la opción 1 tendríamos que consultar dos tablas.

M. E/R → M. Relacional

Jerarquía de clases



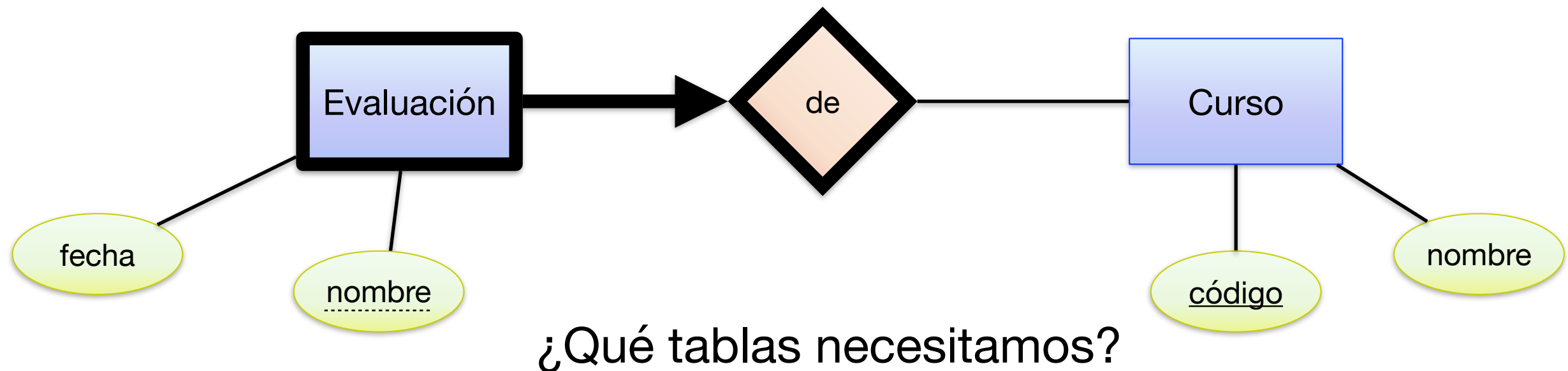
Opción 3: Quitar la jerarquía

Bebida(nombre: string, origen: string, año: string, tipo: string)

- Muchas repeticiones de la columna tipo.
- Puede que no se conozca el tipo (nulls).
- Pero más sencillo (y comprimible)

M. E/R → M. Relacional

Entidades débiles



Curso(nombre: string, origen: string)

Evaluación(nombre: string, C.código: string, fecha: date)

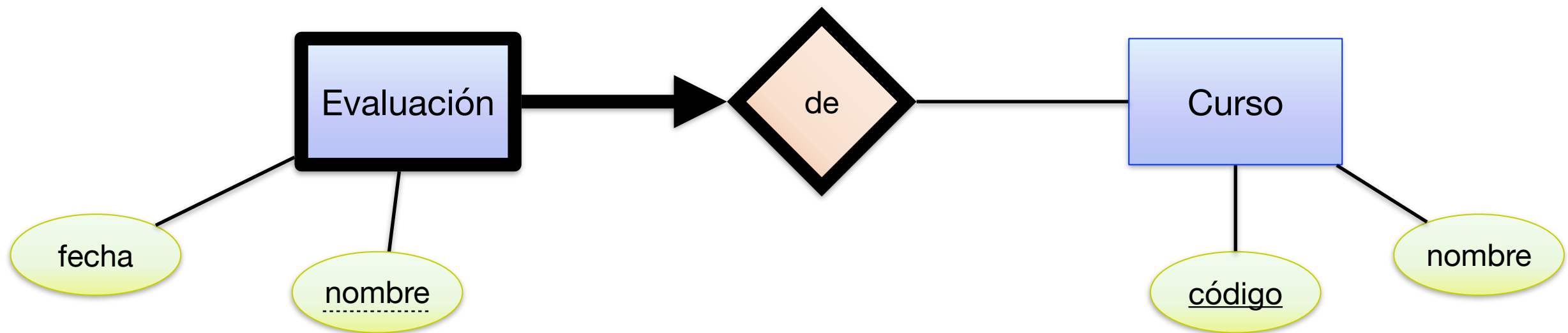
De(E.nombre: string, C.código: string) ❌

¿Está bien esto?

La tabla **De** es redundante (1-a-algo)
(y mal nombre para una tabla)

M. E/R → M. Relacional

Entidades débiles



Curso(nombre: string, origen: string)

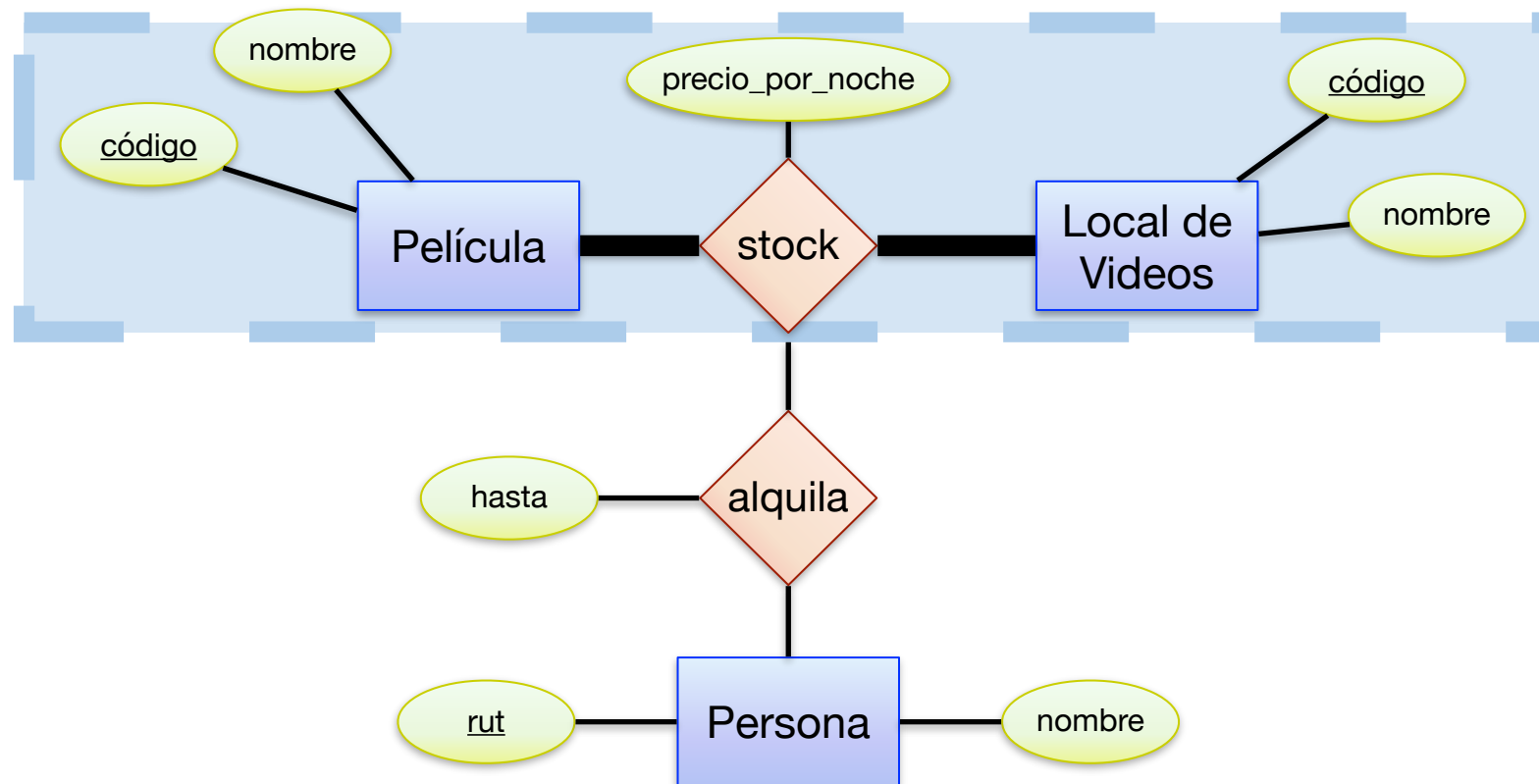
Evaluación(nombre: string, C.código: string, fecha: date)

!Ahora sí!

```
CREATE TABLE evaluacion(  
    nombre string NOT NULL,  
    codigo varchar(30) NOT NULL,  
    fecha date,  
    PRIMARY KEY (nombre, codigo)  
    FOREIGN KEY(codigo) REFERENCES curso(codigo) ON DELETE CASCADE  
)
```

M. E/R → M. Relacional

Agregación



Película(código: string, nombre: string)

LocalDeVideos(código: string, nombre: string)

Stock(P.código: string, L.código: string, precio_por_noche: int)

Persona(rut: string, nombre: string)

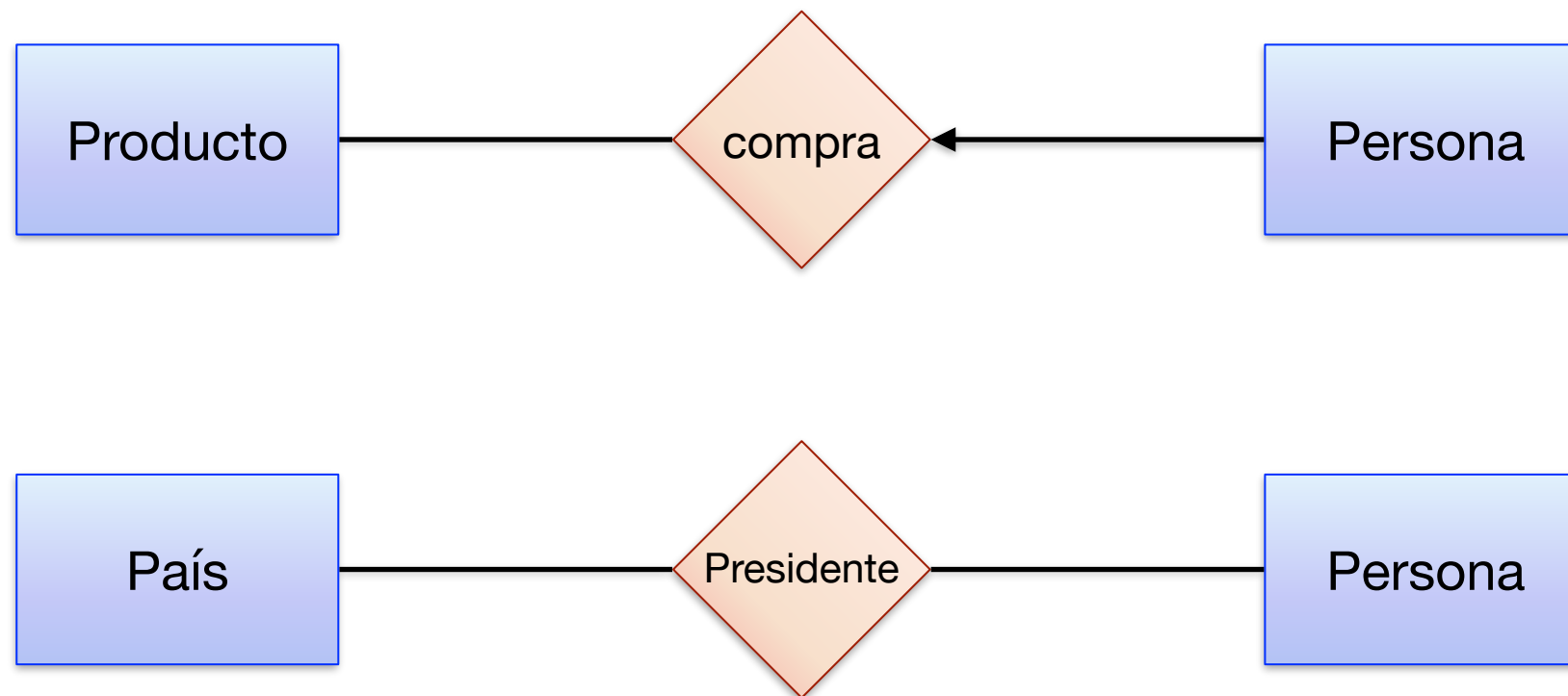
Alquila(S.p código: string, S.l código: string, Pr.rut, hasta: date)

Principios básicos del diseño

Principios básicos del diseño

Fidelidad al problema

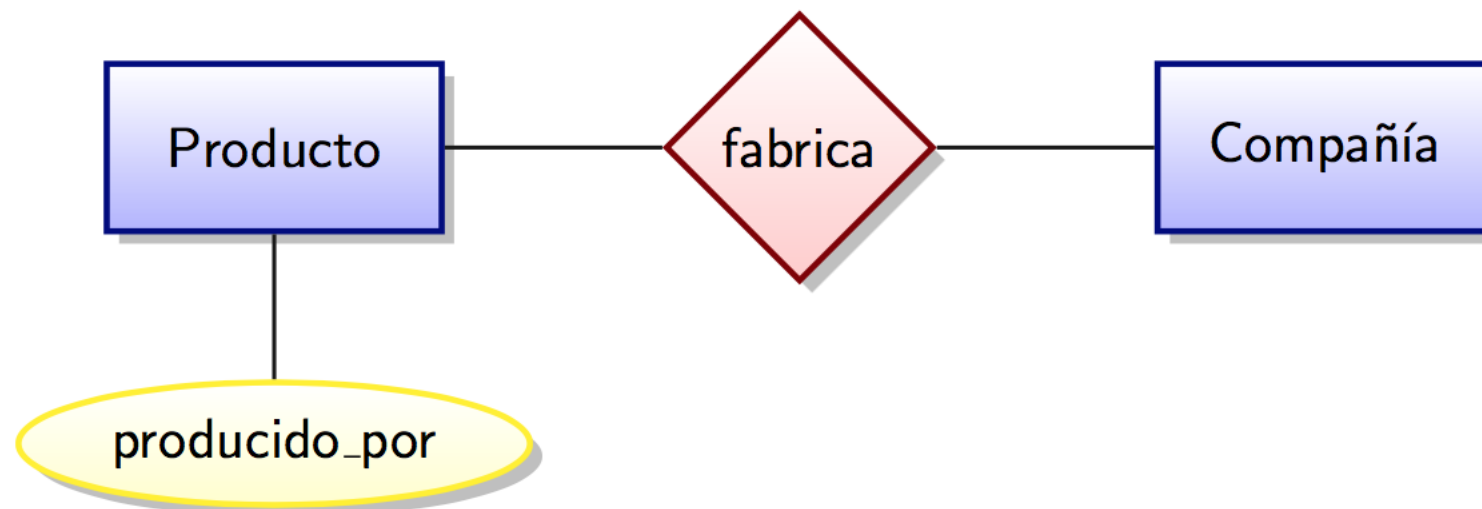
¿Qué está mal?



Principios básicos del diseño

Evitar redundancia

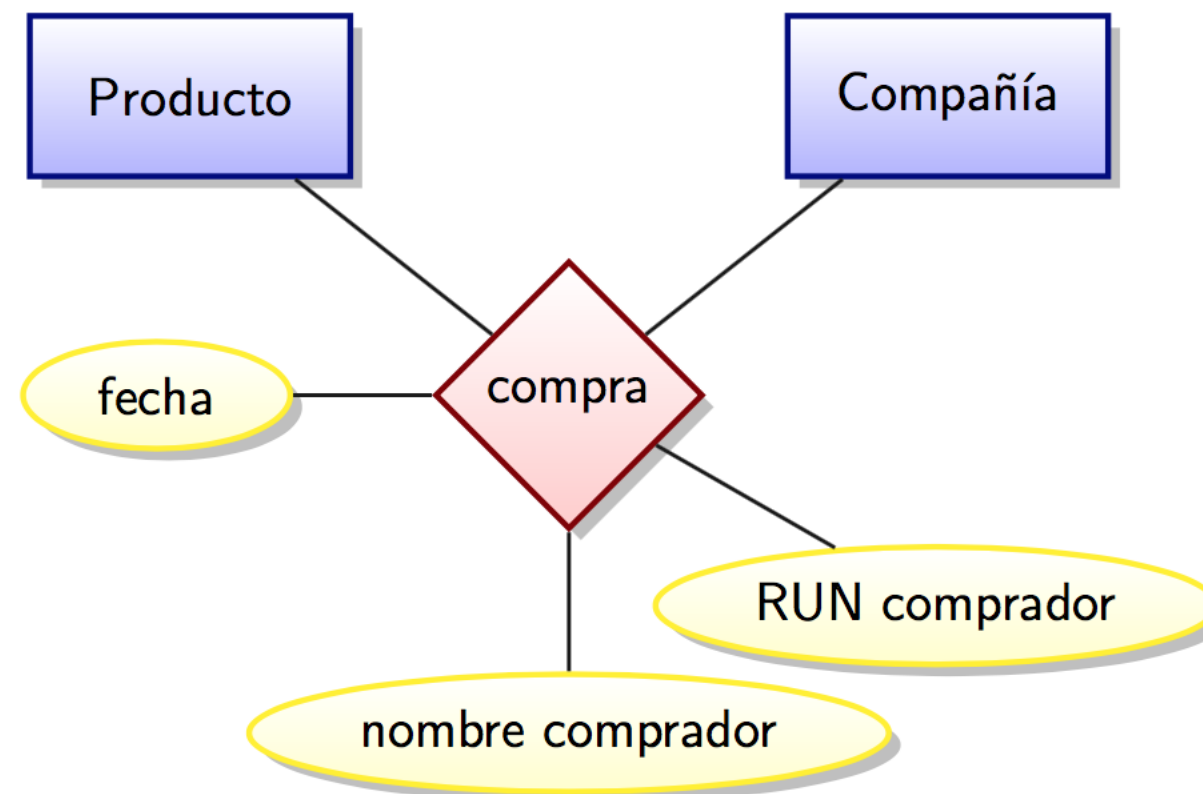
Algo como esto, puede generar **anomalías**



Principios básicos del diseño

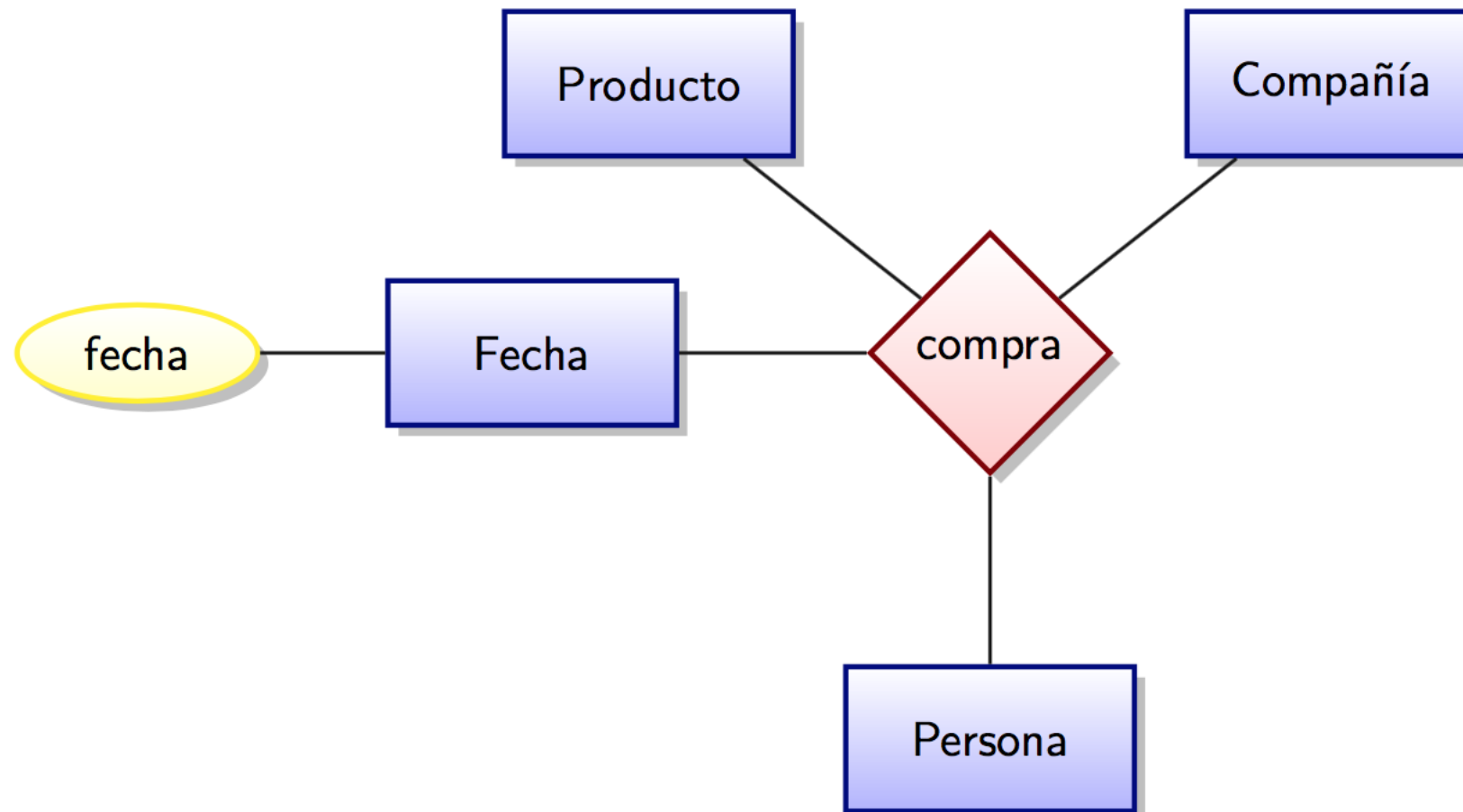
Elegir entidades y relaciones correctamente

¿Qué está mal?



Principios básicos del diseño

No complicar más de lo necesario



Principios básicos del diseño

Elección de llave primaria.

Al momento de diseñar siempre queremos identificar todas los atributos de las entidades que son candidatos a ser llave de la tabla, a estos les llamamos *natural key*, porque son columnas que naturalmente tienen el comportamiento de una llave. Por ejemplo de la siguiente tabla:

Usuario(email, rut, username, nombre, tipo, fecha_de_inscripcion)

rut, email y username son posibles *natural keys*.

...pero en la práctica el 99% de las veces es mejor usar una columna inventada, sin significado que sea autogenerada por el RDBMS. A esto le llamamos [surrogate key](#).

Principios básicos del diseño

Elección de llave primaria.

Bueno en realidad es algo medio opinionado...

Surrogate vs. natural/business keys [closed]

Asked 12 years, 7 months ago Active 2 months ago Viewed 71k times



Closed. This question is [opinion-based](#). It is not currently accepting answers.

Principios básicos del diseño

Elección de llave primaria.

Bueno en realidad es algo medio opinionado...

Pero en la práctica los frameworks de desarrollo web modernos esperan una *surrogate key* llamada *id* como llave primaria e incluso la generan por defecto.

La tabla anterior deberíamos generarla así:

```
CREATE TABLE usuario(  
    id SERIAL,  
    email nombre varchar(30) UNIQUE NOT NULL,  
    RUT varchar(30) UNIQUE NOT NULL,  
    fecha date,  
    PRIMARY KEY (id)  
)
```

Llaves foráneas

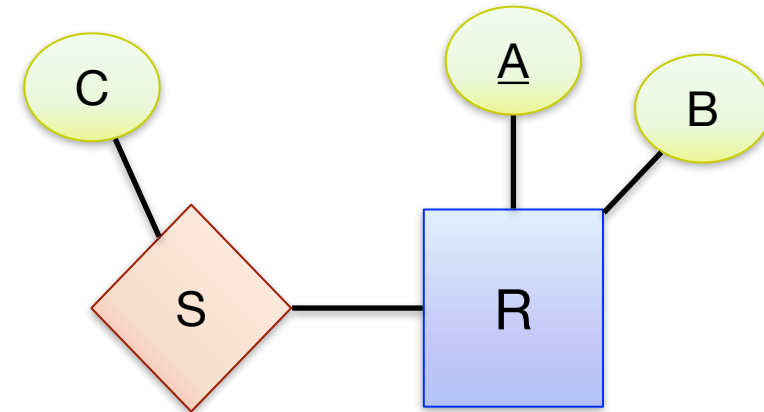
Llaves Foráneas en SQL

Inserciones con llaves foráneas

¿Qué pasa en este caso?

```
CREATE TABLE R(a int, b int, PRIMARY KEY(a));  
CREATE TABLE S(a int, c int, FOREIGN KEY(a) REFERENCES R, ...);  
INSERT INTO R VALUES(1, 1);  
INSERT INTO S VALUES(1, 2);
```

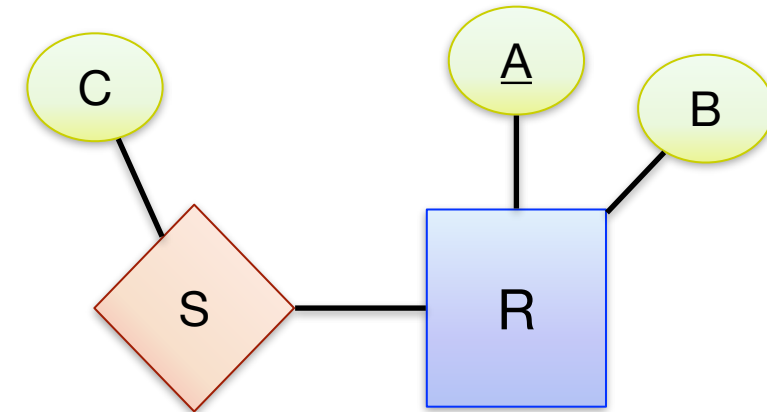
Todo bien hasta ahora...



Llaves Foráneas en SQL

Inserciones con llaves foráneas

¿Qué pasa en este caso?



```
CREATE TABLE R(a int, b int, PRIMARY KEY(a));  
CREATE TABLE S(a int, c int, FOREIGN KEY(a) REFERENCES R, ...);  
INSERT INTO R VALUES(1, 1);  
INSERT INTO S VALUES(1, 2);  
INSERT INTO S VALUES(2, 3); ❌
```

ERROR!

La base de datos no permite que se agreguen filas en que la llave foránea no está en la tabla referenciada!

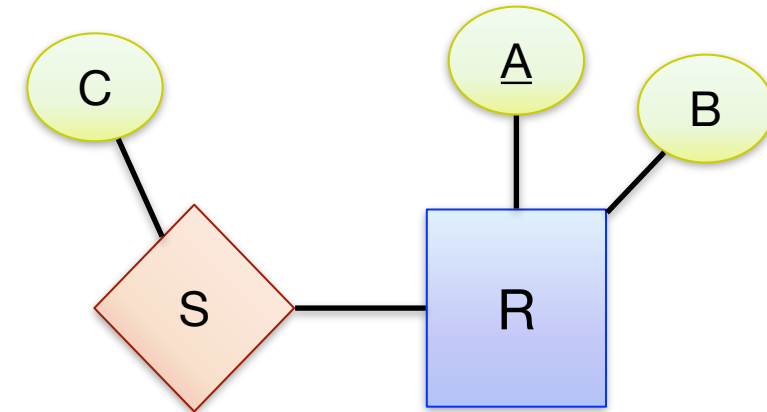
Llaves Foráneas en SQL

Eliminar con llaves foráneas

Tenemos $S[a] \subseteq R[a]$ (llave foránea)

S	A	C
	1	3
	2	2

R	A	B
	1	2
	2	3

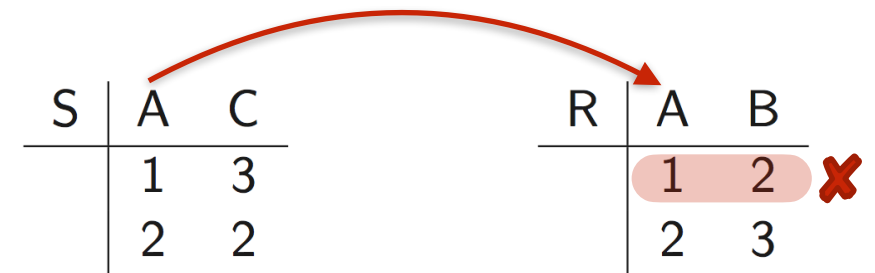


Qué ocurre al eliminar (1, 2) en **R**?

Llaves Foráneas en SQL

Eliminar con llaves foráneas

Qué ocurre al eliminar (1, 2) en **R**?



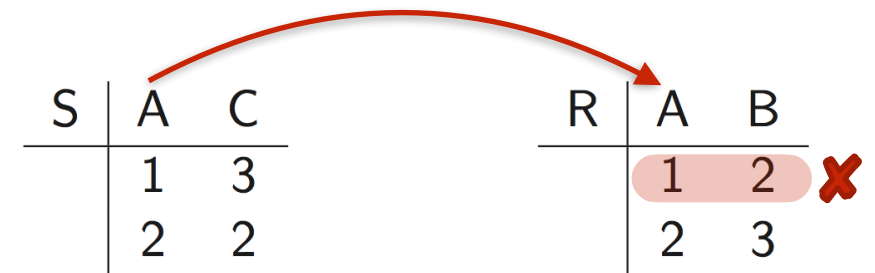
Tenemos las siguientes opciones:

- No permitir eliminación
- Propagar la eliminación y también borrar (1,3) de S
- Mantener la tupla en **S** pero dejar en la llave foránea el valor en null.

Llaves Foráneas en SQL

Eliminar con llaves foráneas

Qué ocurre al eliminar (1, 2) en **R**?



Opción 1: no permitir la eliminación. Es el default en SQL

```
CREATE TABLE R(a int, b int, PRIMARY KEY(a))
```

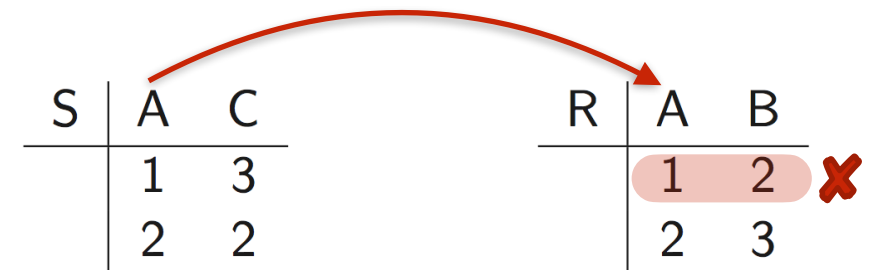
```
CREATE TABLE S(a int, c int, FOREIGN KEY(a) REFERENCES R, ...)
```

Respuesta: obtenemos error

Llaves Foráneas en SQL

Eliminar con llaves foráneas

Qué ocurre al eliminar (1, 2) en **R**?



S	A	C
	1	3
	2	2

R	A	B
	1	2
	2	3

Opción 2: Propagar la eliminación. (Cascada de eliminaciones)

```
CREATE TABLE R(a int, b int, PRIMARY KEY(a))
```

```
CREATE TABLE S(a int, c int,
```

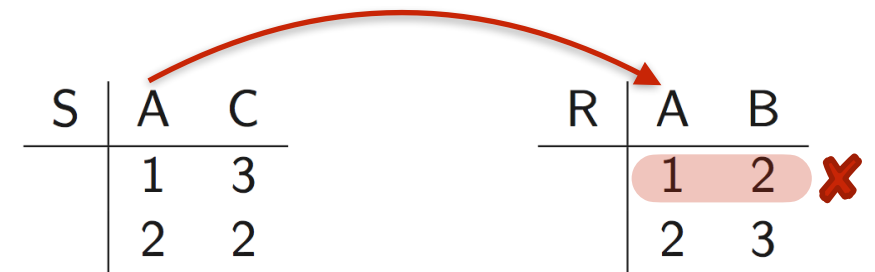
```
FOREIGN KEY(a) REFERENCES R ON DELETE CASCADE, ...)
```

Respuesta: se elimina también (1, 3) en S

Llaves Foráneas en SQL

Eliminar con llaves foráneas

Qué ocurre al eliminar (1, 2) en **R**?



S	A	C
	1	3
	2	2

R	A	B
	1	2
	2	3

Opción 3: dejar en nulo

```
CREATE TABLE R(a int, b int, PRIMARY KEY(a))  
CREATE TABLE S(a int, c int,  
                FOREIGN KEY(a) REFERENCES R ON DELETE SET NULL, ...)
```

Respuesta: la tupla (1, 3) en S ahora es (null, 3)

Restricciones de integridad

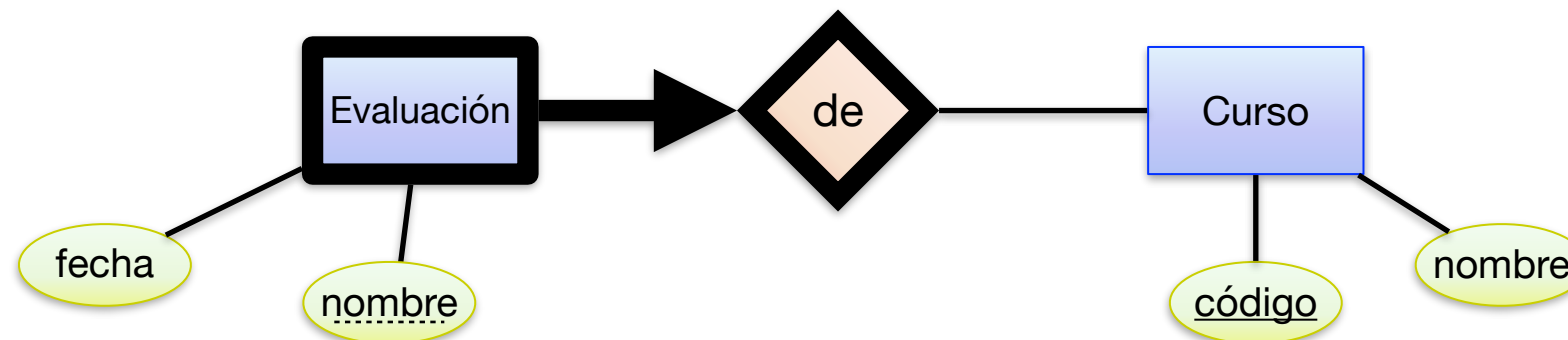
Restricciones de integridad

Son **restricciones** formales que imponemos a **un esquema** que todas **sus instancias** deben satisfacer. Algunas son:

- **De valores nulos:** El valor puede o no ser nulo.
- **Unicidad:** Dado un atributo, no pueden haber dos tuplas con el mismo valor.
- **De llave:** el valor es único y no puede ser null.
- **De referencia:** si se trabaja en una compañía, esta debe existir (Llaves foráneas).
- **De dominio:** la edad de las personas debe estar entre 0 y 150 años.

M. E/R → M. Relacional

Entidades débiles



Curso(nombre: string, origen: string)

Evaluación(nombre: string, C.código: string, fecha: date)

No puede ser nulo

```
CREATE TABLE evaluacion(
```

```
nombre string NOT NULL,
```

Tiene que tener a lo más 30 caracteres

```
codigo varchar(30) NOT NULL,
```

No puede ser nulo

```
fecha date DEFAULT NOW(),
```

Tiene que ser una fecha, y su valor por defecto es la fecha actual

```
PRIMARY KEY (nombre, codigo)
```

La llave son "nombre" y "codigo"

```
FOREIGN KEY(codigo) REFERENCES curso(codigo) ON DELETE CASCADE
```

"codigo" es una llave foránea a la tabla curso

Borrar las tuplas de evaluación que dependan de un codigo en la tabla curso que fue eliminado.