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

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.

Veamos un ejemplo práctico

Un requerimiento real

La Asociación de Enoturismo quiere hacer una aplicación web para mostrar sus actividades de enoturismo. Estas consisten principalmente en tours por las distintas viñas, que están repartidas por todas las regiones del país.

Se quiere que los usuarios puedan ver las viñas de cada región, los tours, los vinos que ofrece cada viña y los que se puedan degustar en cada tour, etc.

Un requerimiento real

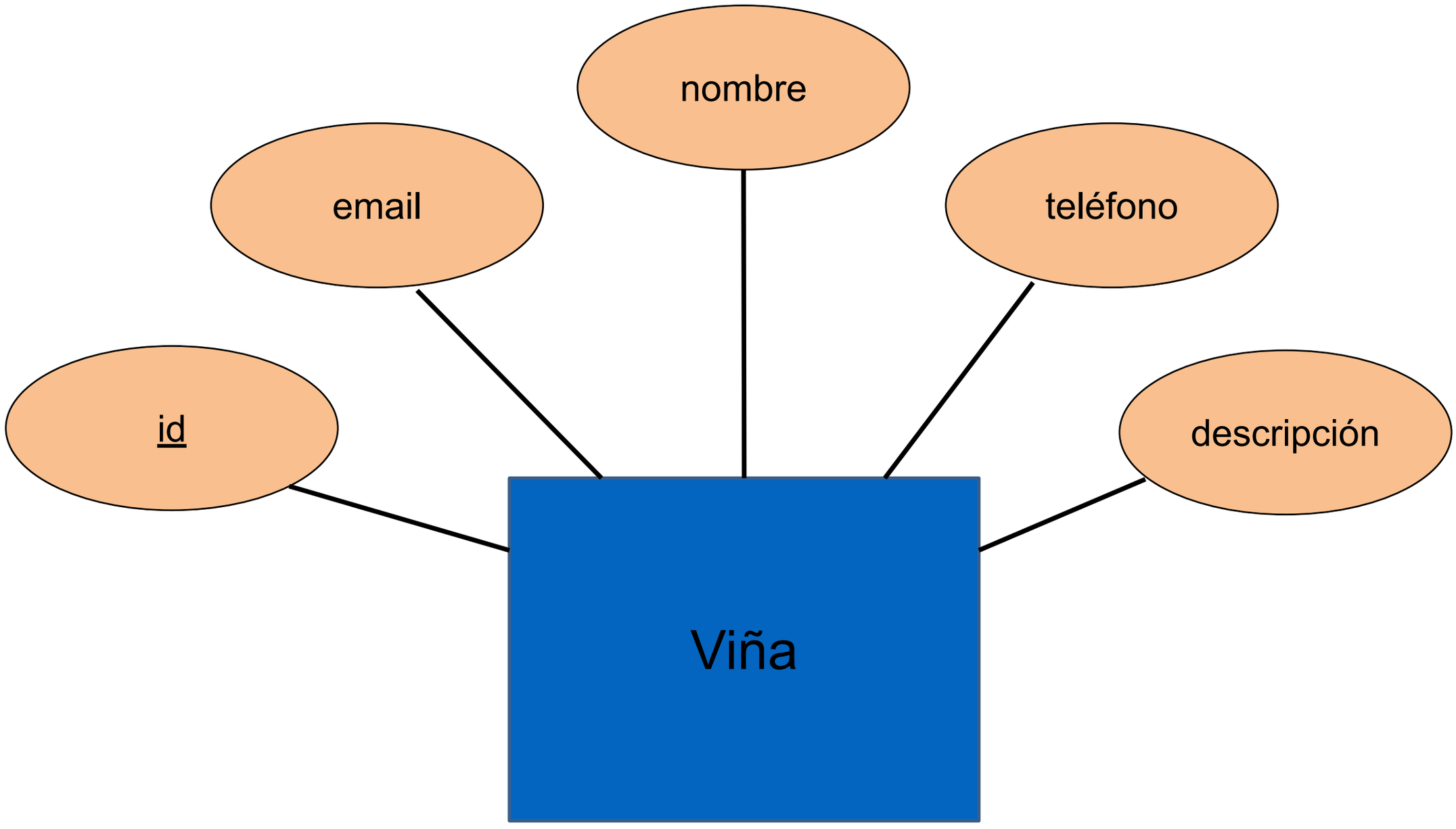
Queremos tener información sobre las siguientes entidades:

- **Regiones**: nombre, reseña
- **Viñas**: su región, nombre, teléfono, descripción
- **Vinos**: nombre, cepa, precio, descripción
- **Tours:** nombre, viñas visitadas, vinos degustados, precio.

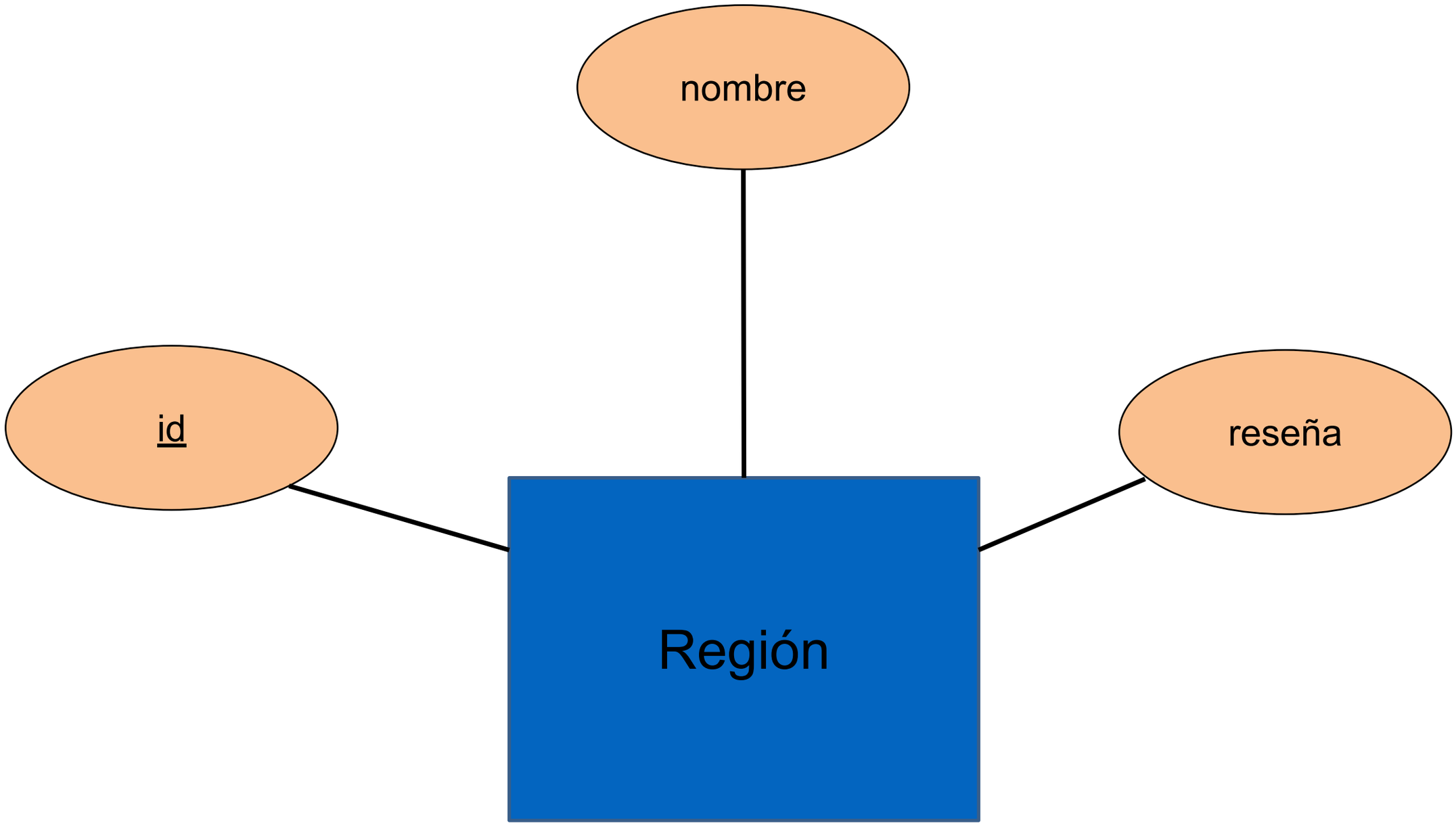
Podríamos agregar **muchas** cosas más pero para un inicio del proyecto es razonable.

Partamos modelando las entidades

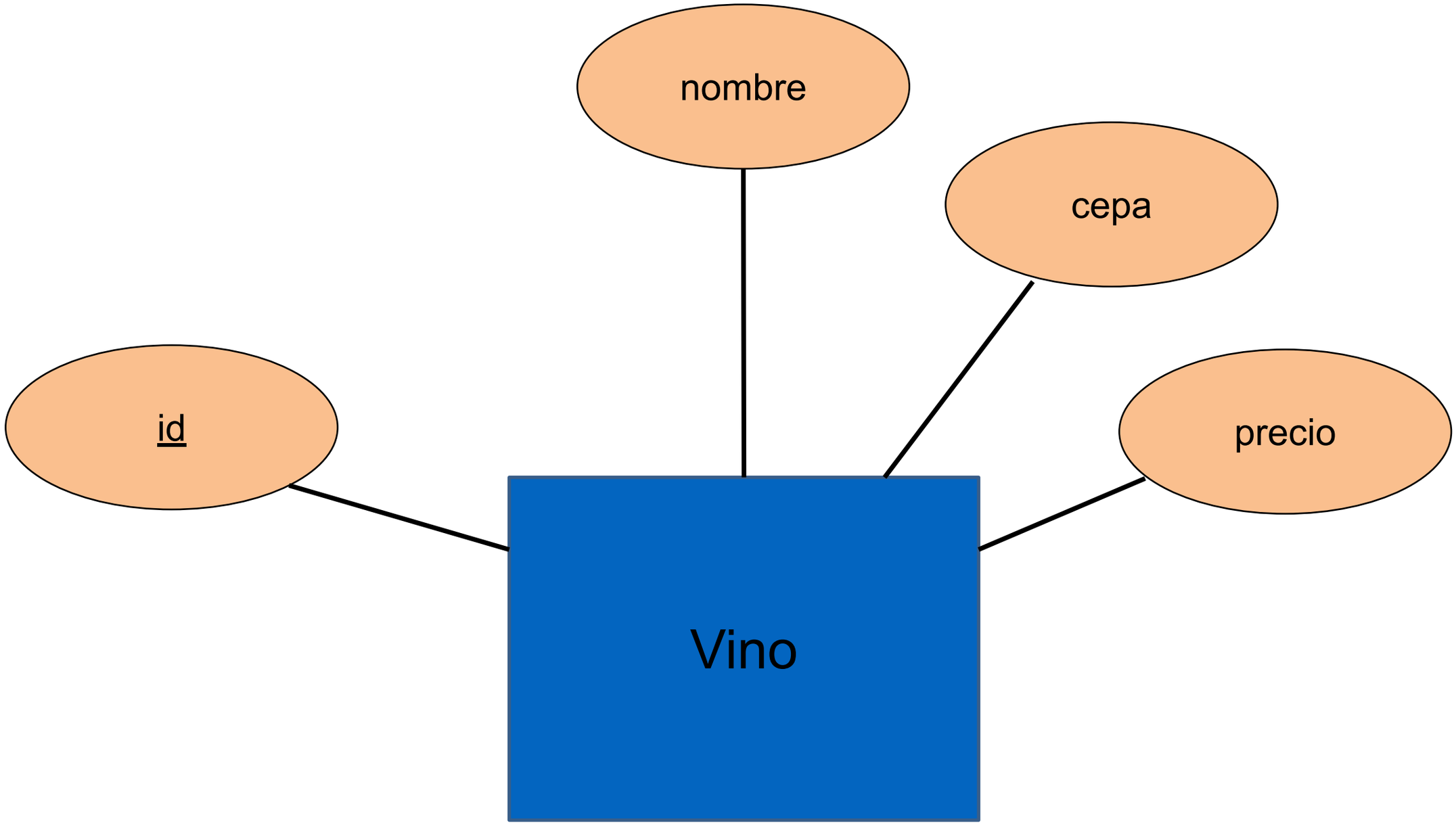
Viña:



Región:



Vino:



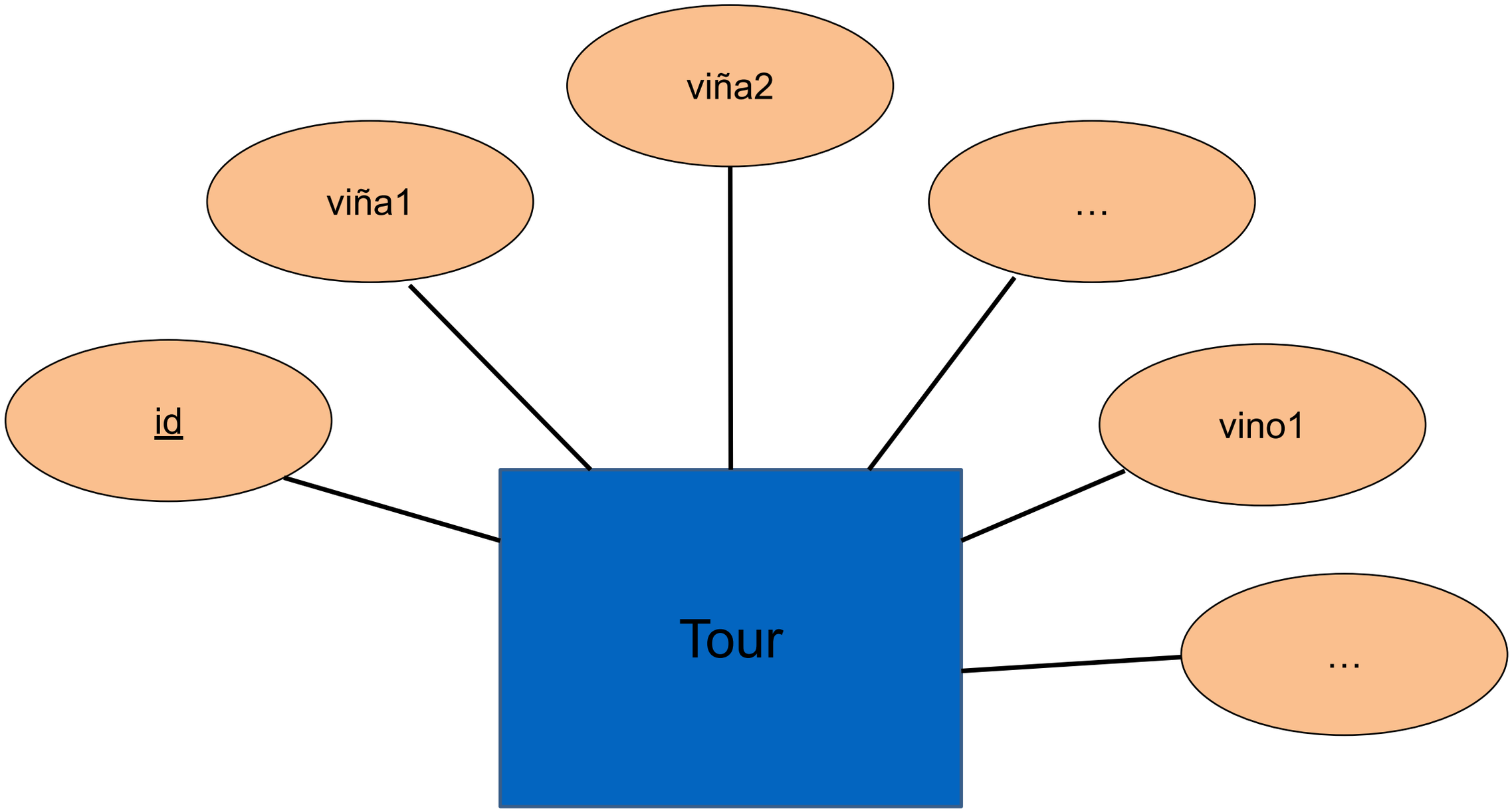
...y los tours?

Recordemos nuestro caso:

- **Regiones**: nombre, reseña
- **Viñas**: su región, nombre, teléfono, descripción
- **Vinos**: nombre, cepa, precio, descripción
- **Tours:** nombre, viñas visitadas, vinos degustados, precio.

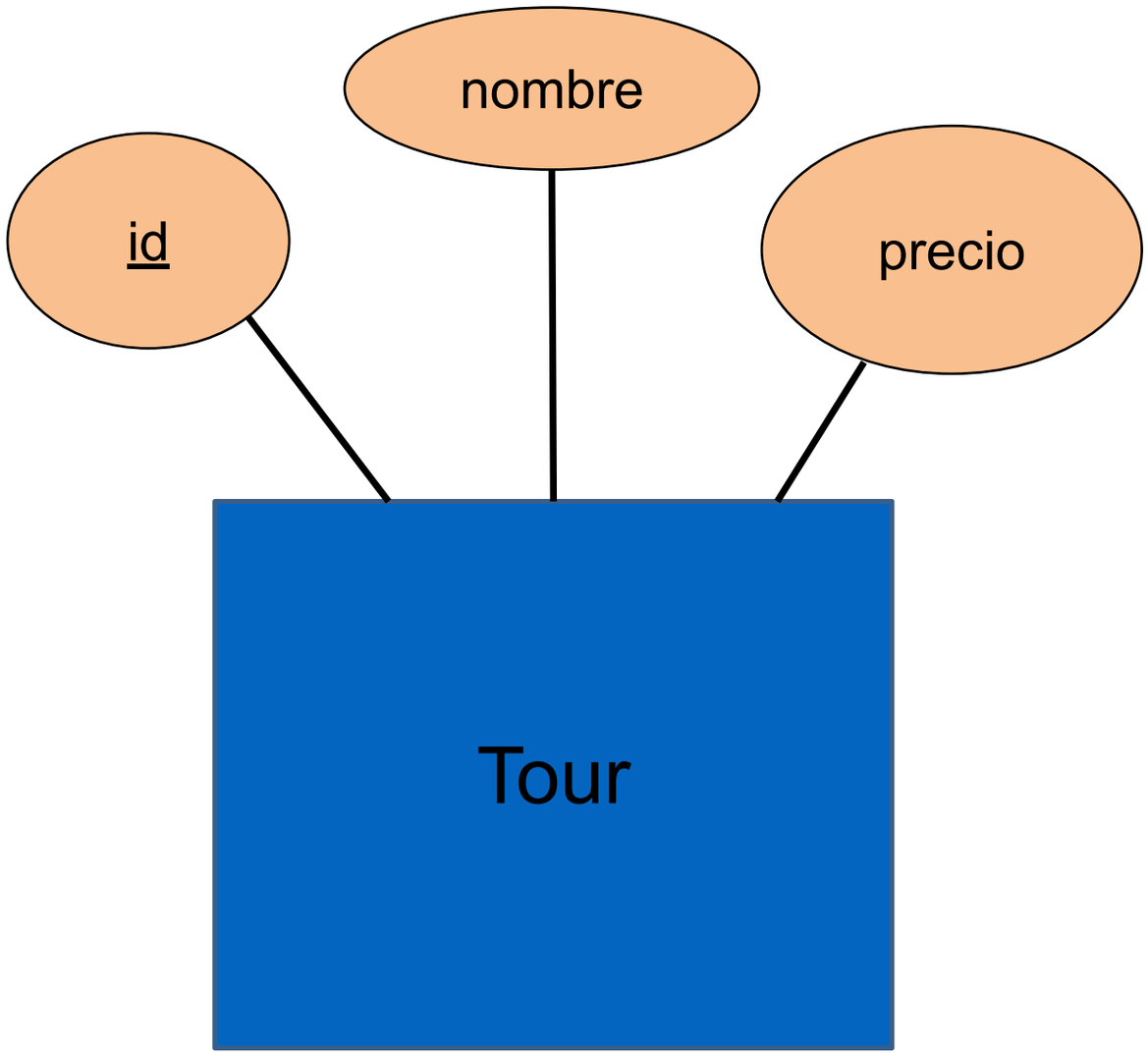
Tenemos que un tour puede tener *n* viñas y *m* vinos.

Tour:



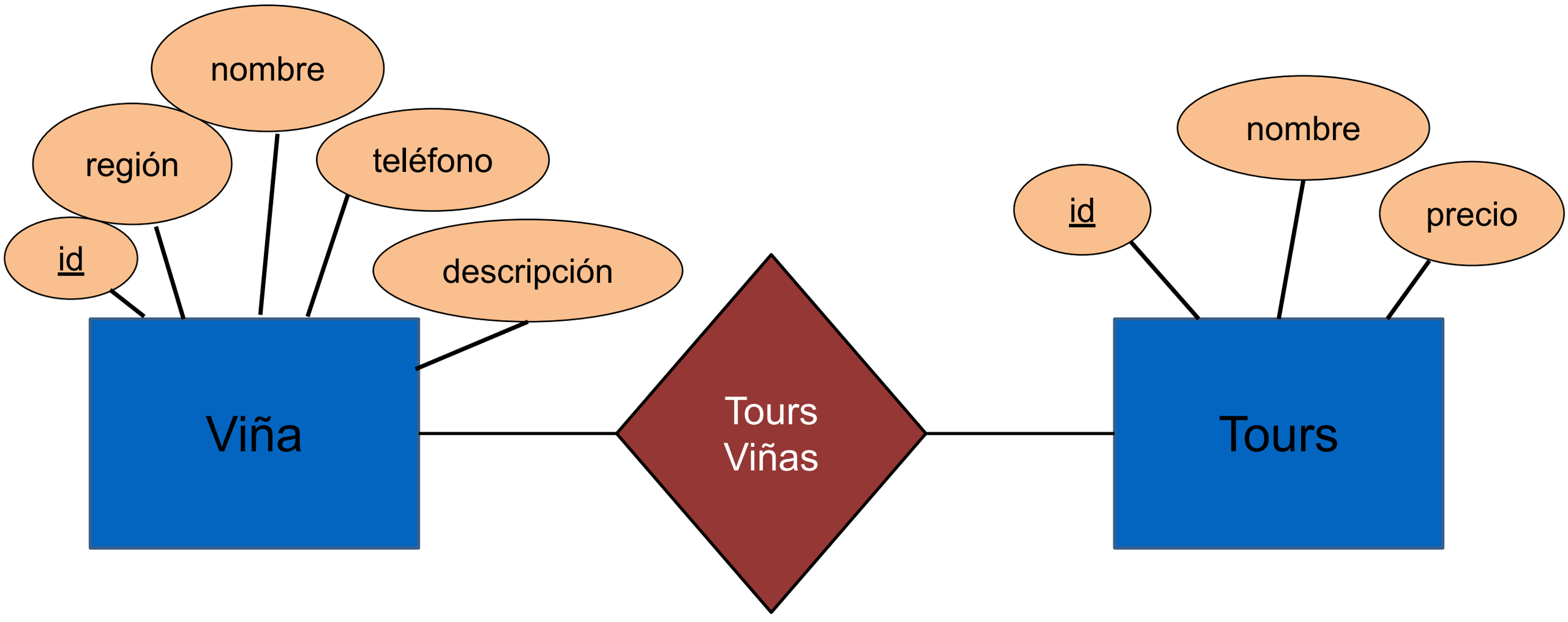
Esto claramente no es una forma muy práctica de modelarlo.

Tour:



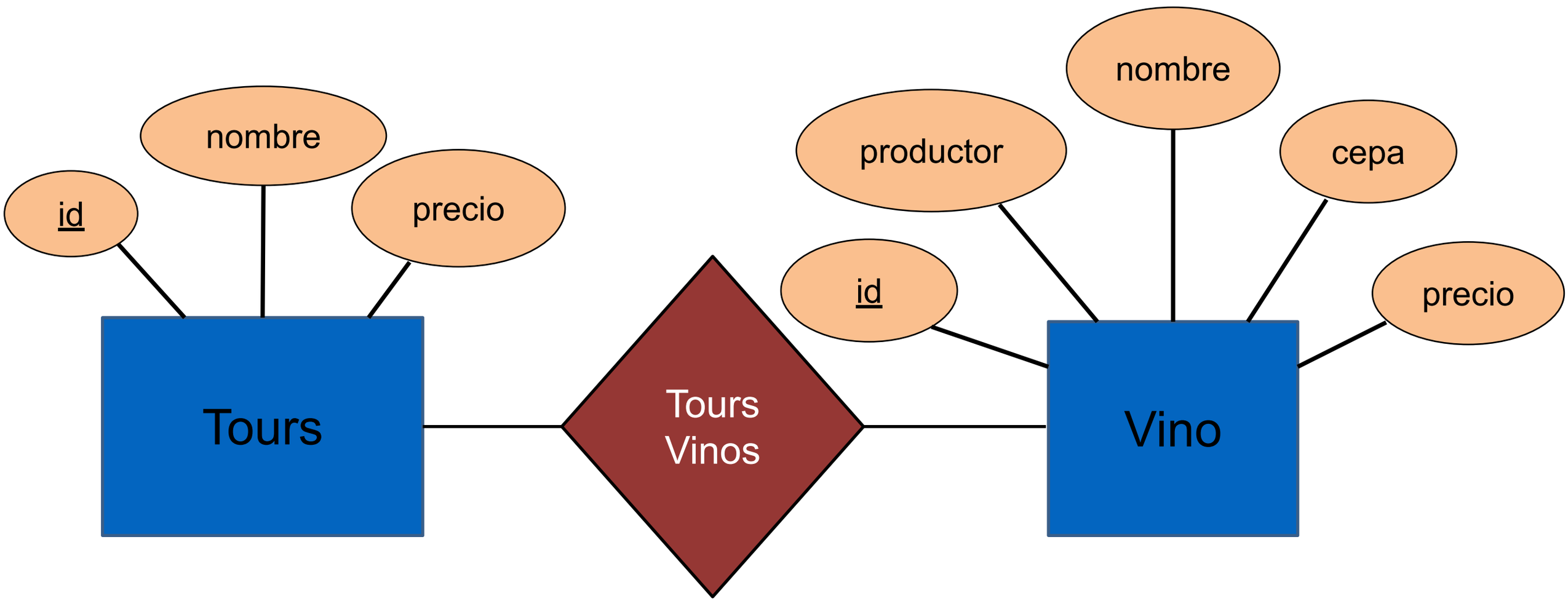
Necesitamos que el tour solo tenga sus atributos específicos, el resto lo modelamos mediante una **relación** (no confundir con las tablas del modelo relacional).

ToursViñas:

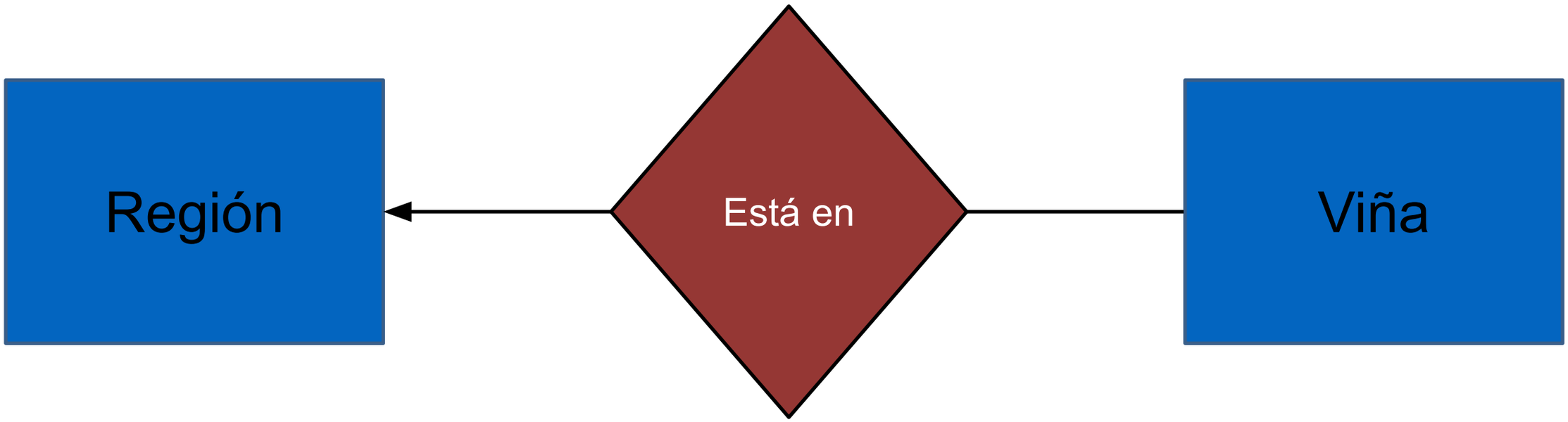


ToursViñas es una relación entre viña y tours.

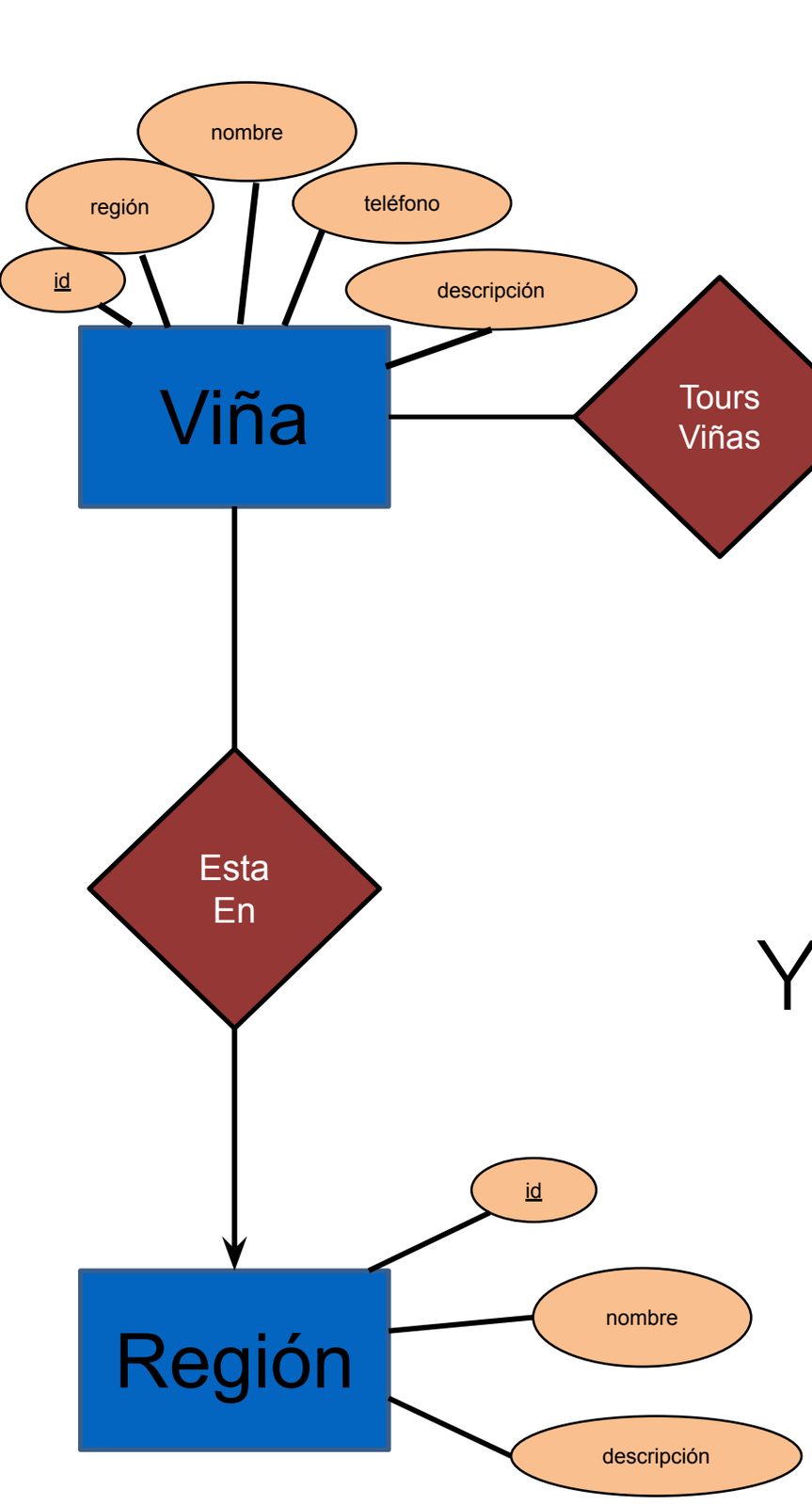
ToursVinos:


nombre
id
precio
Tours
Tours
Vinos
productor
nombre
cepa
id
precio
Vino

Además nos faltaba agregar que una viña pertenece a una región:



La flecha indica que es una relación 1:N, una viña pertenece a una sola región pero una región tiene muchas viñas.



Y así nos queda el diagrama completo.

Es lo que llamamos un **E/R** o Entidad- Relación

Diagramas E/R

Diagramas E/R

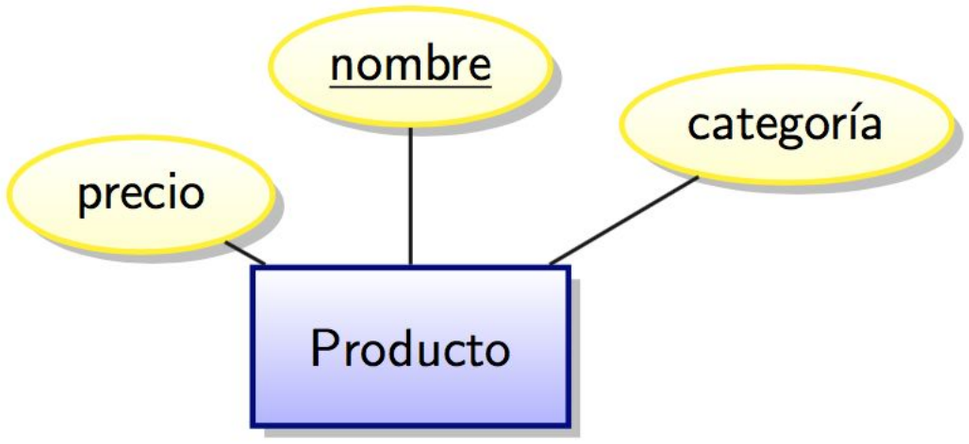
Entidad

Atributo

Relación

Diagramas E/R

Entidad con sus atributos



Obligatorio: cada entidad debe tener una llave

Relaciones

Sean *A* y *B* conjuntos, una relación (binaria) *R* es un subconjunto de *A* x *B*

$$A = \{1, 2, 3\}, B = \{a, b, c, d\}$$

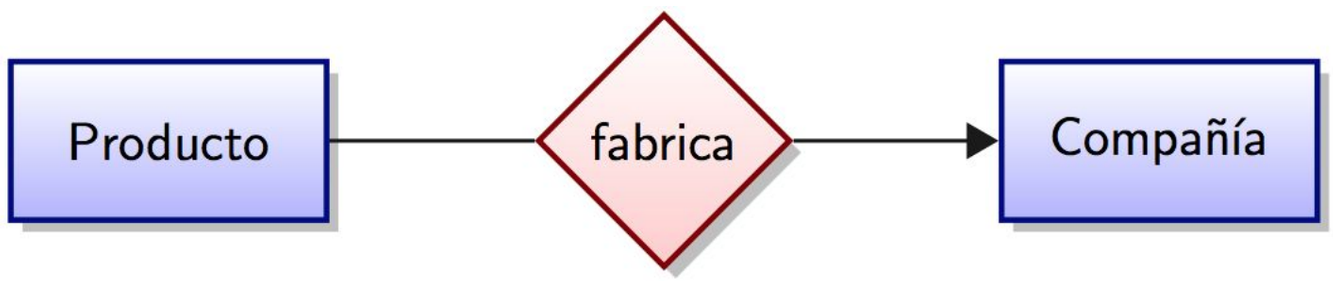
$$R = \{(1, a), (1, c), (2, b)\}$$

Veremos como modelar esto en un esquema relacional.

Relaciones

Multiplicidad

Varios a uno (N:1):



Cada producto tiene **una única** compañía, pero una compañía tiene **más de un** producto

Relaciones

Multiplicidad

Uno a uno (1:1):

Cada producto tiene **una única** compañía, y cada compañía tiene un **único** producto

Relaciones

Multiplicidad

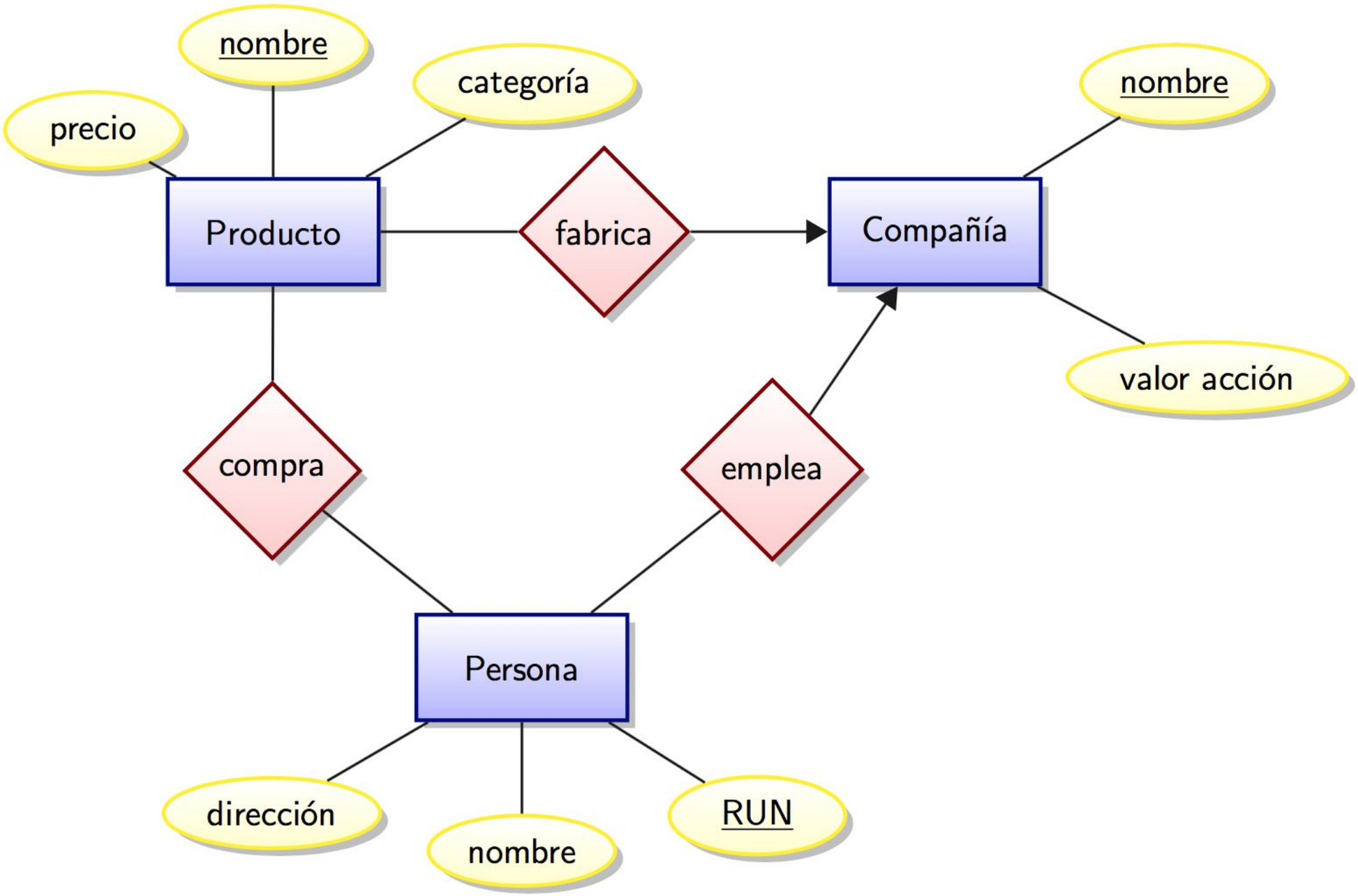
Varios a varios (N:N):

Cada producto tiene **muchas** compañías y cada compañía tiene **muchos** productos

Cuando es N:N la relación es una tabla del esquema. En este caso **Fábrica** se compone de la llave de **Producto** y de **Compañía** (veremos más de eso después).

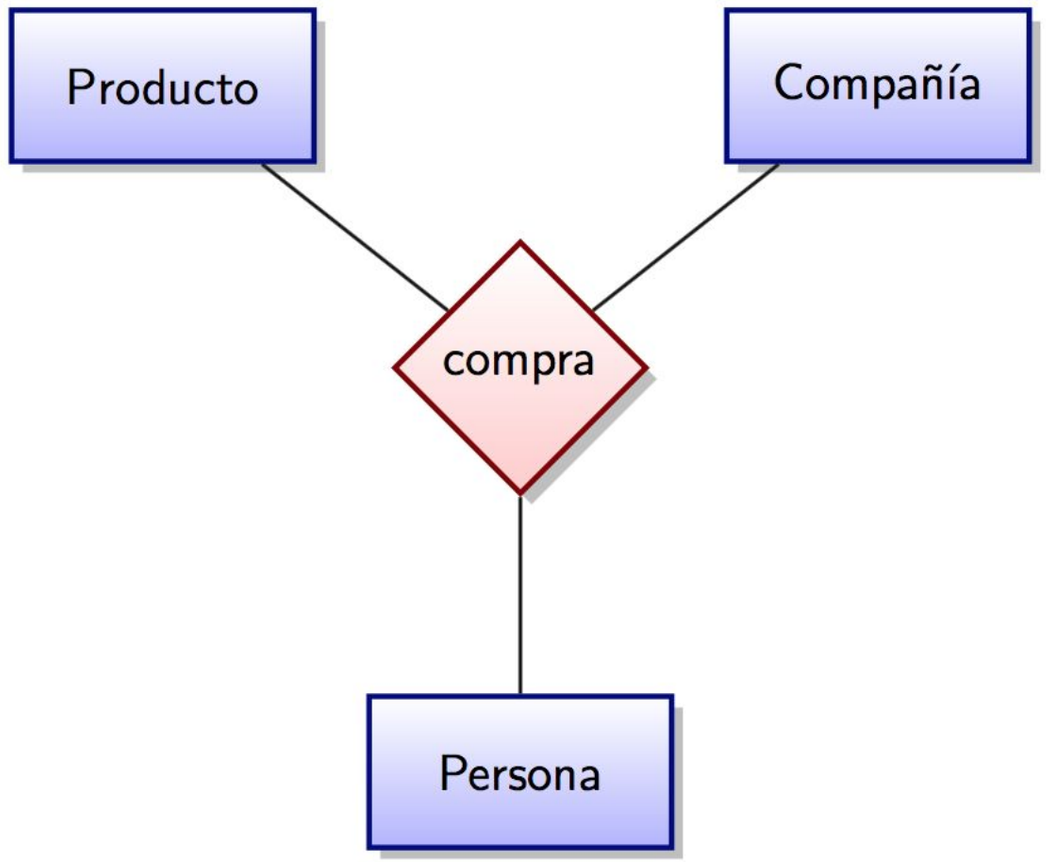
Diagramas E/R

Ejemplo



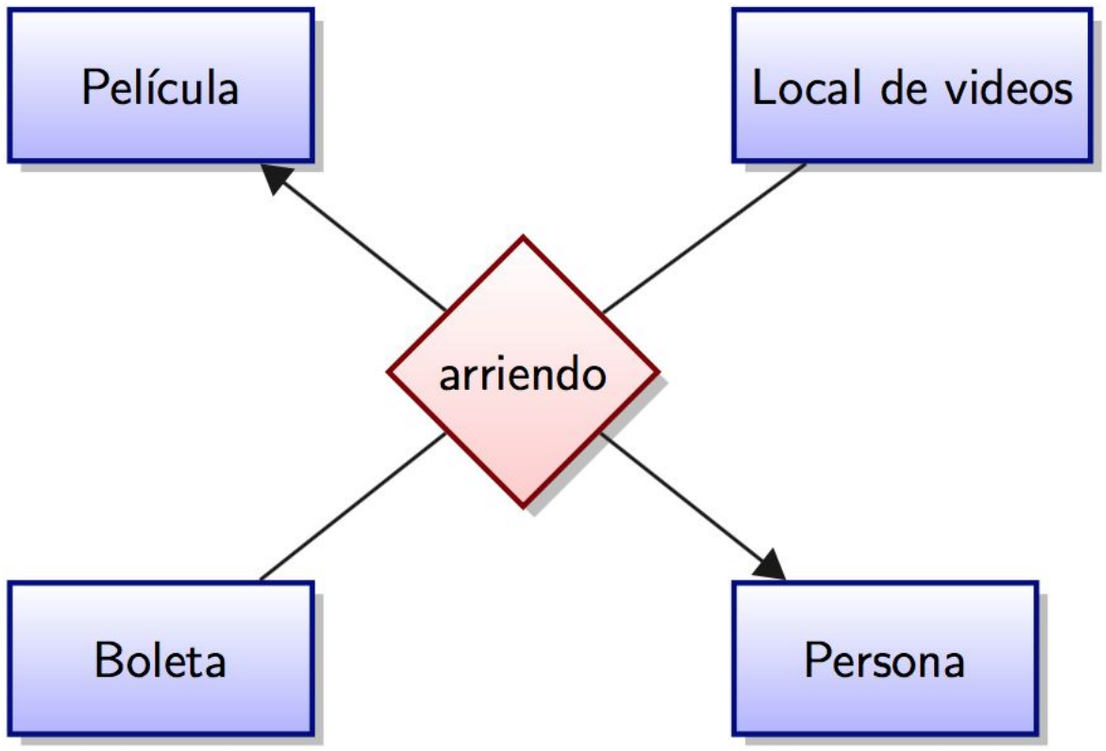
Relaciones múltiples

Una relación puede involucrar a más de 2 entidades

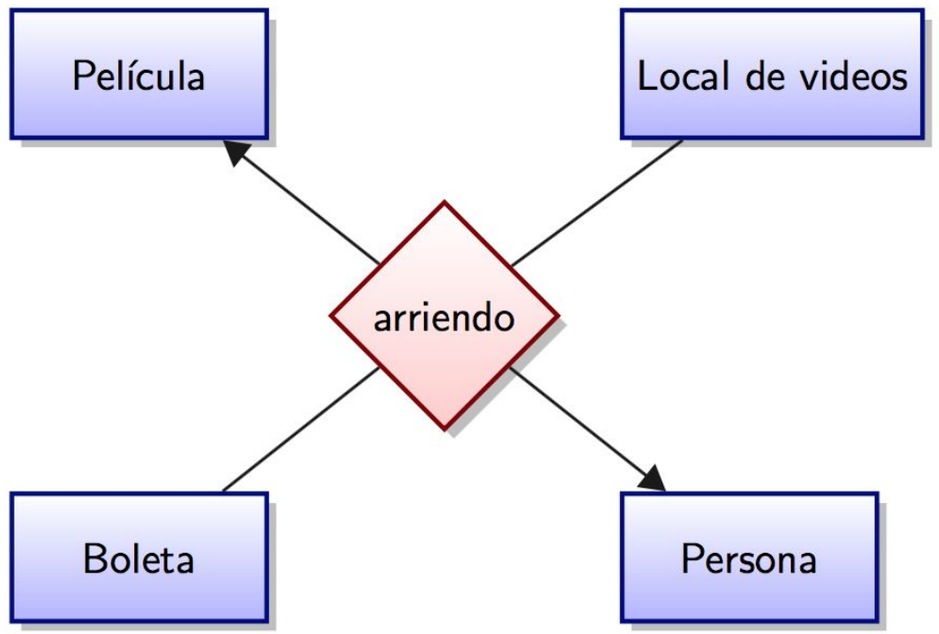


Dependencia multifuncional

¿Qué significa esto?



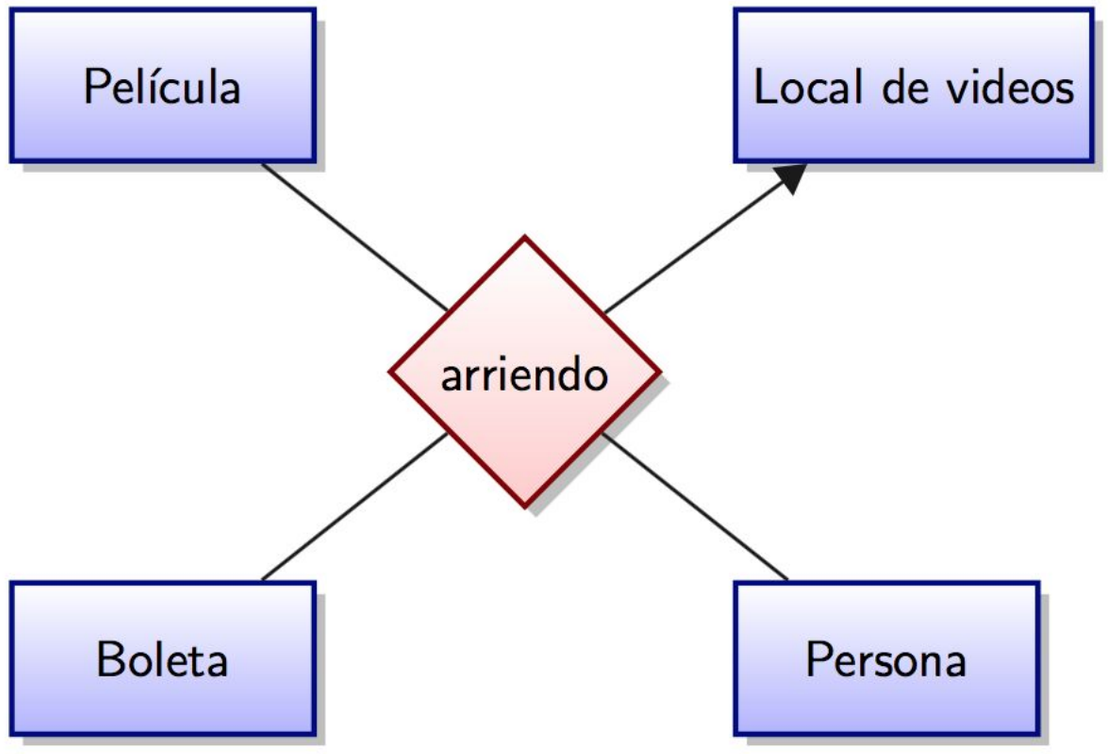
Dependencia multifuncional



- Puedo determinar la película con la boleta, la persona y el local
- Puedo determinar la persona con la película, la boleta y el local

Dependencia multifuncional

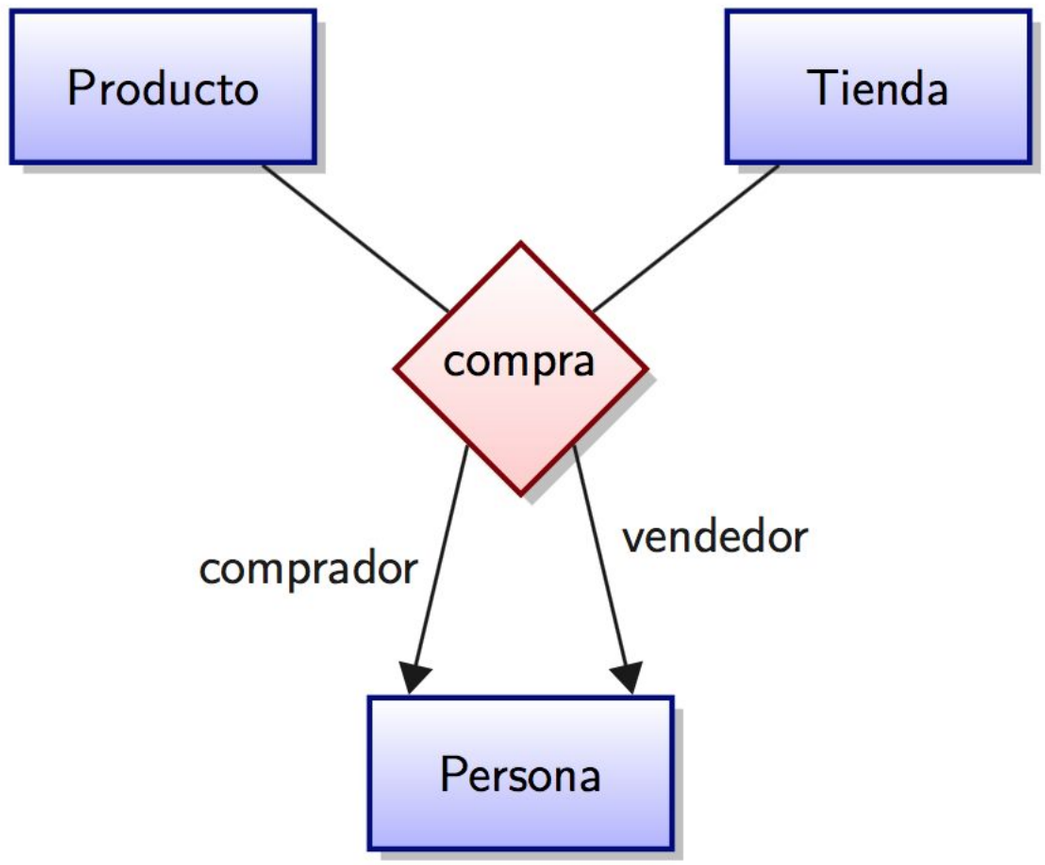
Quiero decir “la boleta determina la tienda”



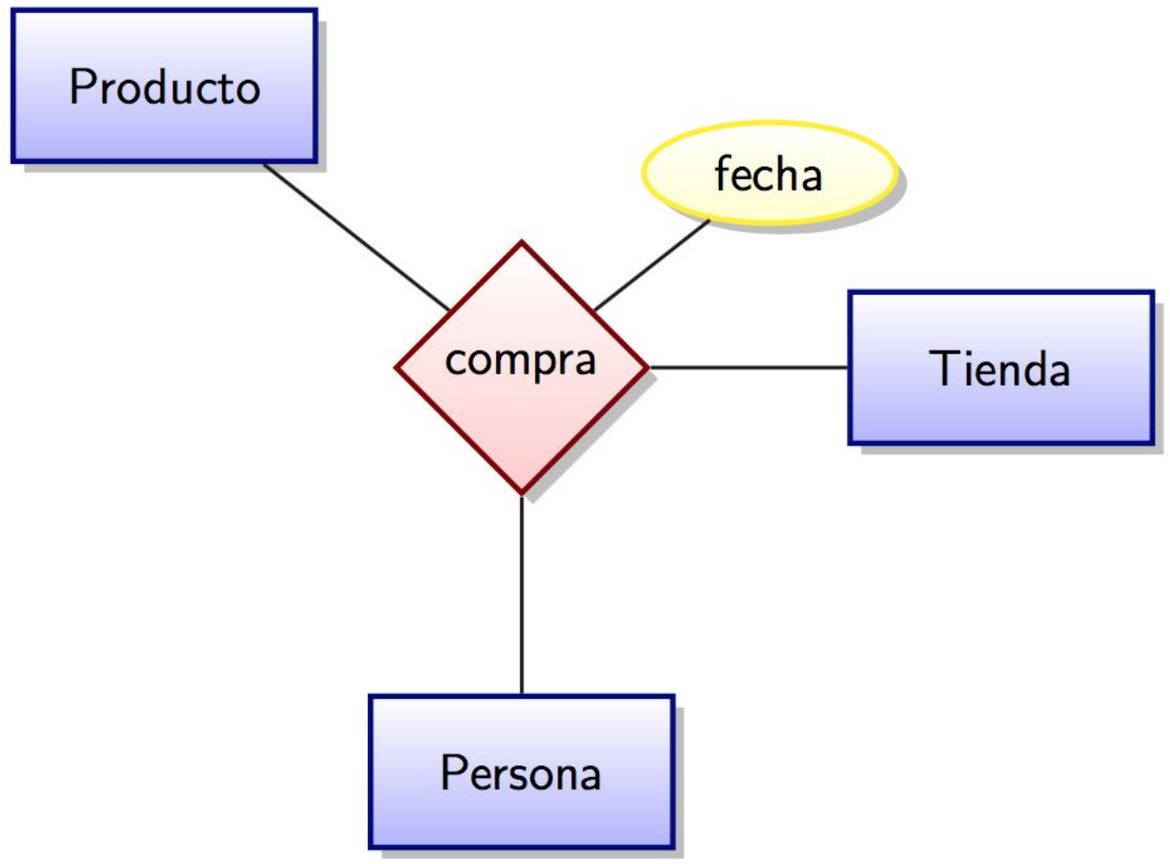
¿Por qué esto está incompleto?

Roles

Una entidad puede participar más de una vez en la relación



Atributos en relaciones

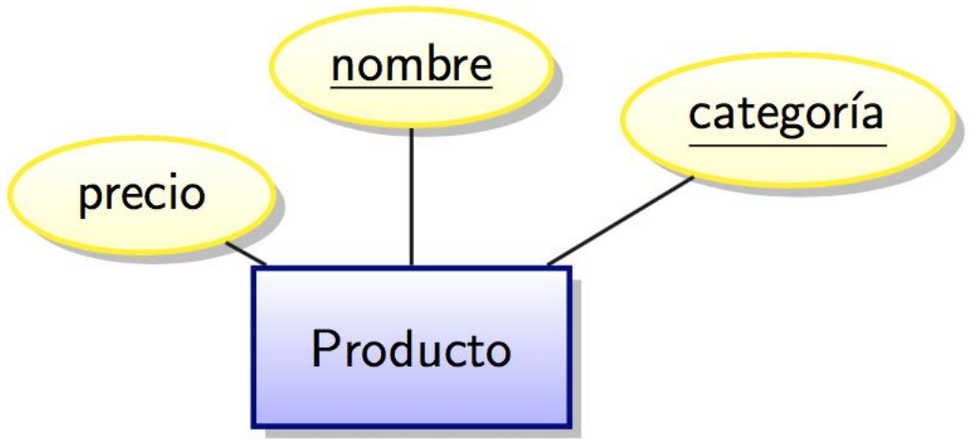


De E/R a esquema relacional

- Algunas cosas son directas: Cada entidad es una tabla y cada atributo una columna
- Las relaciones se ven caso a caso.

De E/R a esquema relacional

De entidad a tabla



Producto(<u>nombre, categoría</u>, precio)

nombre	categoría	precio
Polaroid SX-70	Fotografía	$100.000
Trípode	Fotografía	$20.000

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

Y las relaciones?

Llaves Foráneas

Restringimos una columna de una tabla a que sus elementos pertenezcan a la llave de otra tabla.

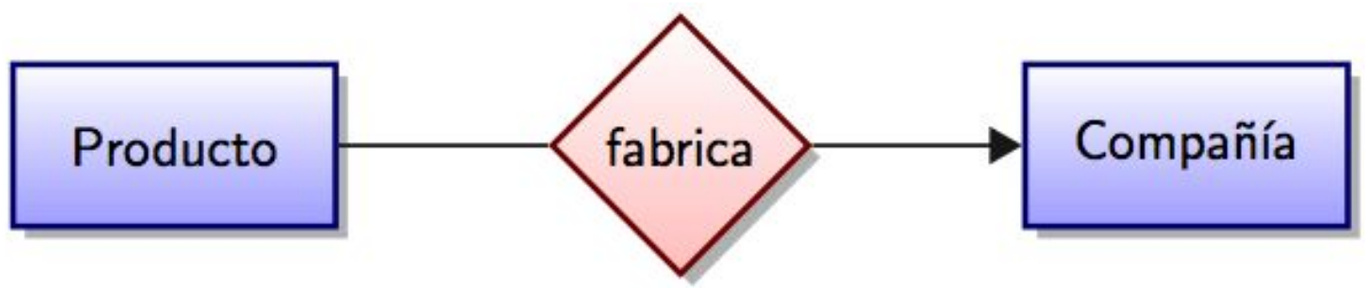
En otras palabras: **Producto.idCompañía ⊆ Compañía.id**

Asumiendo que id es llave de Compañía

Permiten de forma práctica modelar las relaciones y hacer joins entre tablas.

Llaves Foráneas

Relación N:1



¿Cómo modelamos con llaves foráneas?

Colocamos en la tabla que “pertenece” a la otra, una llave foránea que la referencia. En SQL:

```
CREATE TABLE Producto(
    id int,
    nombre varchar(30),
    categoria varchar(30),
    precio int,
    id_compañia int,
    PRIMARY KEY (id),
    FOREIGN KEY(id_compañia) REFERENCES Compañia(id)
)
```

Llaves Foráneas

Relación N:N

¿Cómo modelamos con llaves foráneas?

No podemos directamente. Necesitamos usar una tabla auxiliar para representar la relación. En SQL:

```sql
CREATE TABLE Fabrica(
    id int,
    id_producto int NOT NULL,
    id_compañia int NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY(id_producto) REFERENCES Producto(id)
    FOREIGN KEY(id_compañia) REFERENCES Compañia(id)
)
```

Llaves Foráneas

Relación 1:1

¿Cómo modelamos con llaves foráneas?

Sólo necesitamos una llave foránea en cualquiera de las dos tablas (Por qué??).

Agregamos además una restricción de integridad de unicidad a la llave, para que la relación se mantenga 1:1. Hablaremos más de restricciones de integridad en un rato. En SQL:

```
CREATE TABLE Producto(
    id int,
    nombre varchar(30),
    categoria varchar(30),
    precio int,
    id_compañia int UNIQUE,
    PRIMARY KEY (id),
    FOREIGN KEY(id_compañia) REFERENCES Compañia(id)
)
```

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, FOREIGN KEY(a) REFERENCES R)
INSERT INTO R VALUES(1, 1)
INSERT INTO S VALUES(1)
```

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, FOREIGN KEY(a) REFERENCES R)
INSERT INTO R VALUES(1, 1)
INSERT INTO S VALUES(1)
INSERT INTO S VALUES(2)
```

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

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, FOREIGN KEY(a) REFERENCES R)
```

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

Llaves Foráneas en SQL

Eliminar con llaves foráneas

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

S	A	C
	1	3
	2	2

R	A	B
	1	2
	2	3

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

Respuesta: obtenemos error

Llaves Foráneas en SQL

Eliminar con llaves foráneas

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

```
CREATE TABLE R(a int, b int, PRIMARY KEY(a))
CREATE TABLE S(a int, FOREIGN KEY(a) REFERENCES R ON DELETE CASCADE)
```

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

Llaves Foráneas en SQL

Eliminar con llaves foráneas

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

S	A	C
	1	3
	2	2

R	A	B
	1	2
	2	3

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

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

Llaves Foráneas en SQL

Eliminar con llaves foráneas

Opción 3: dejar en nulo

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

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

Llaves Foráneas en SQL

Eliminar con llaves foráneas

Opción 3: dejar en nulo

S	A	C
	1	3
	2	2

R	A	B
	1	2
	2	3

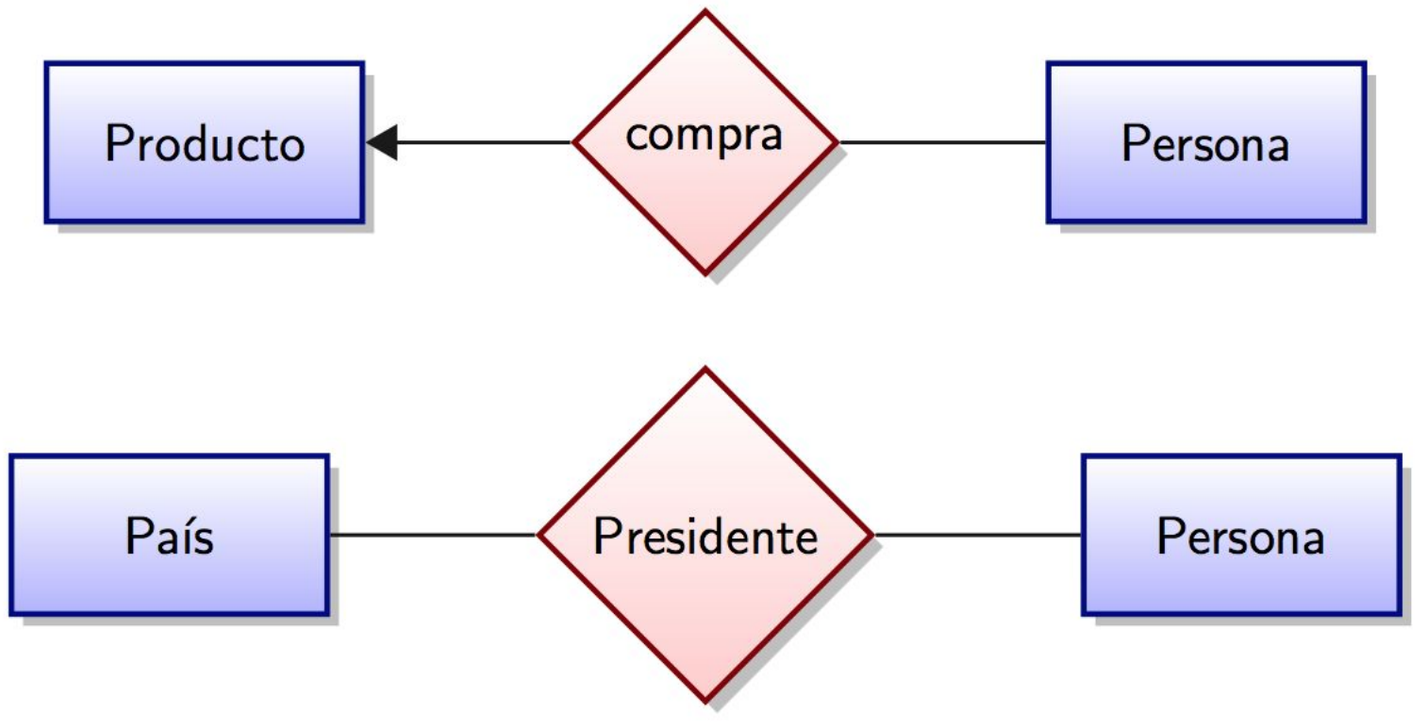
Qué ocurre al eliminar (1, 2) en **R**?

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

Principios básicos del diseño

Fidelidad al problema

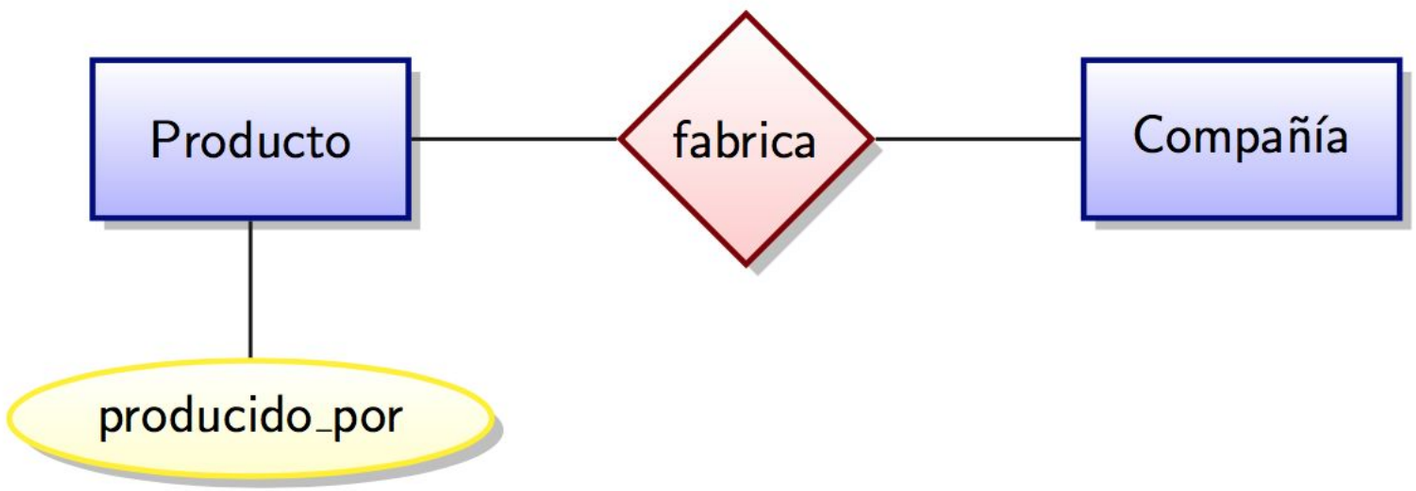
¿Qué está mal?



Principios básicos del diseño

Evitar redundancia

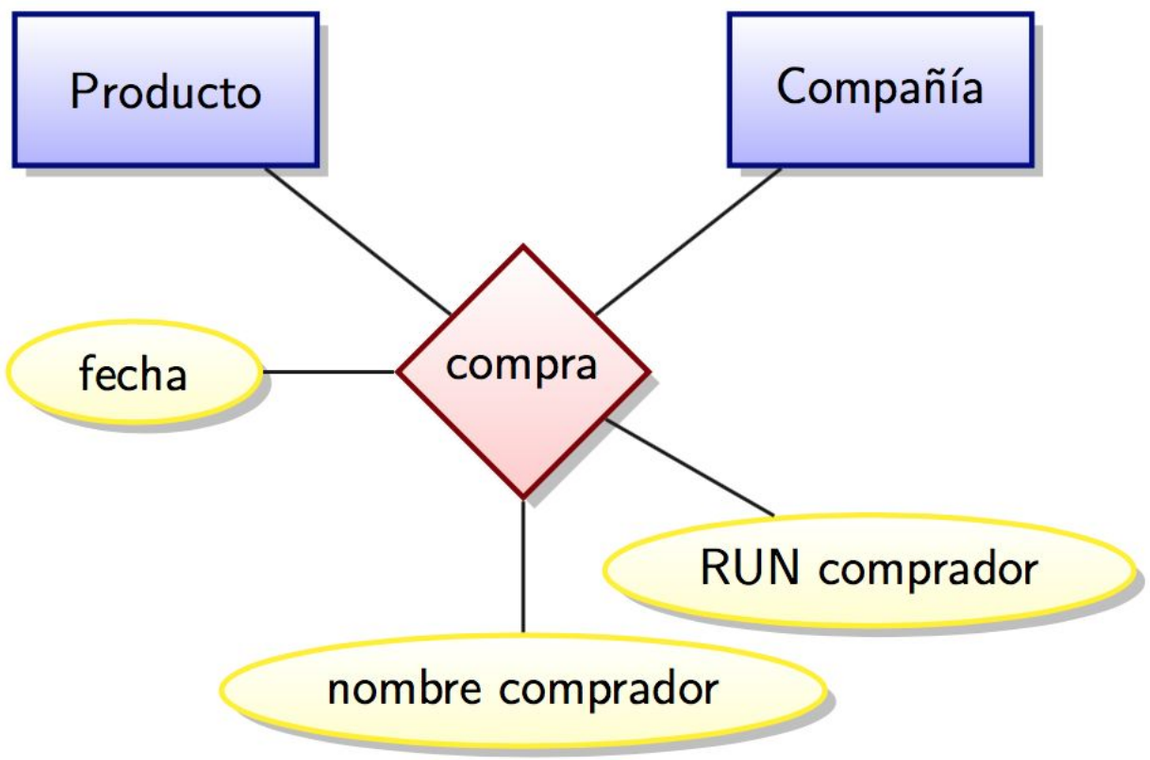
Algo como esto, puede generar **anomalías**



Principios básicos del diseño

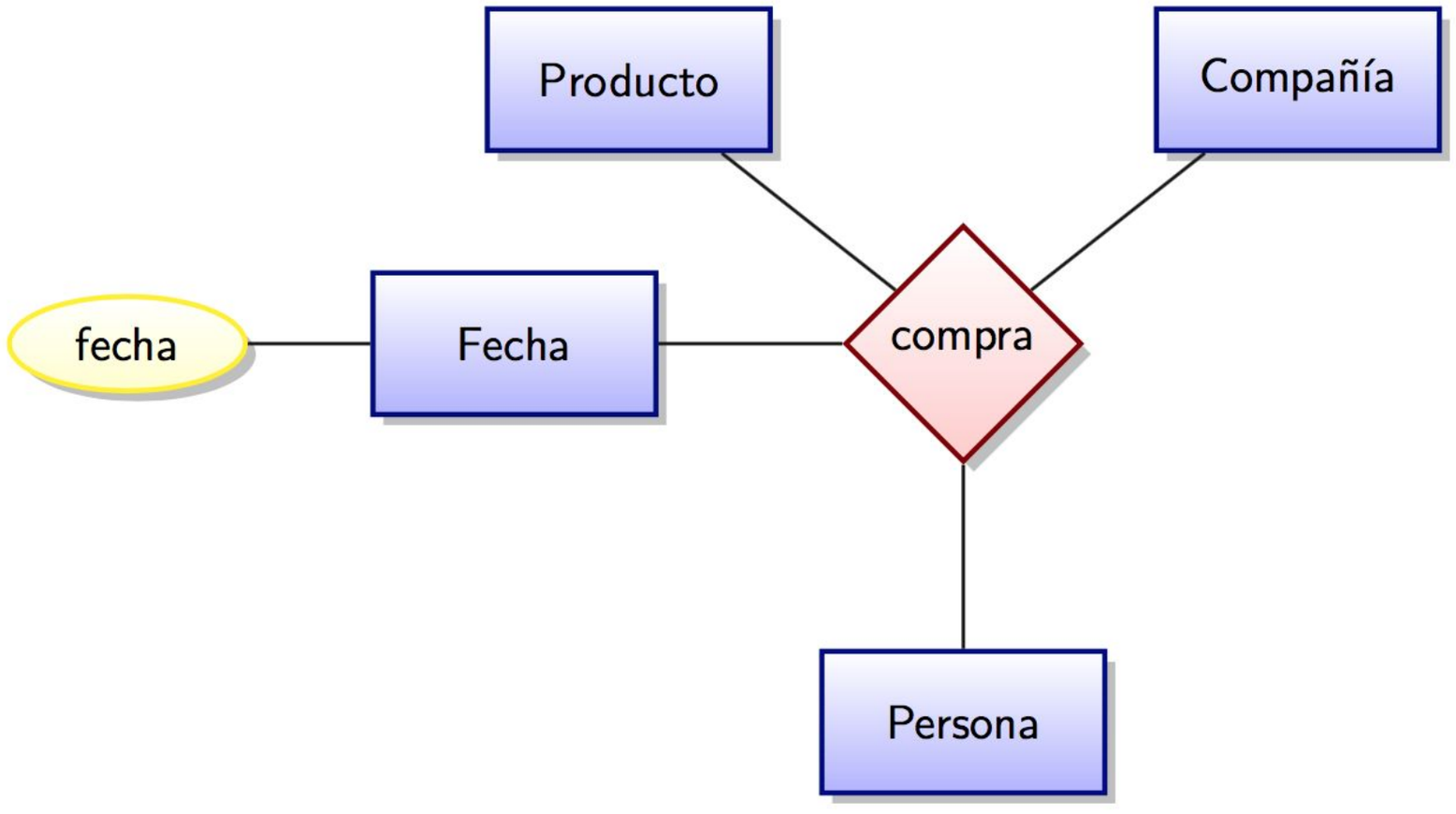
Elegir entidades y relaciones correctamente

¿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:

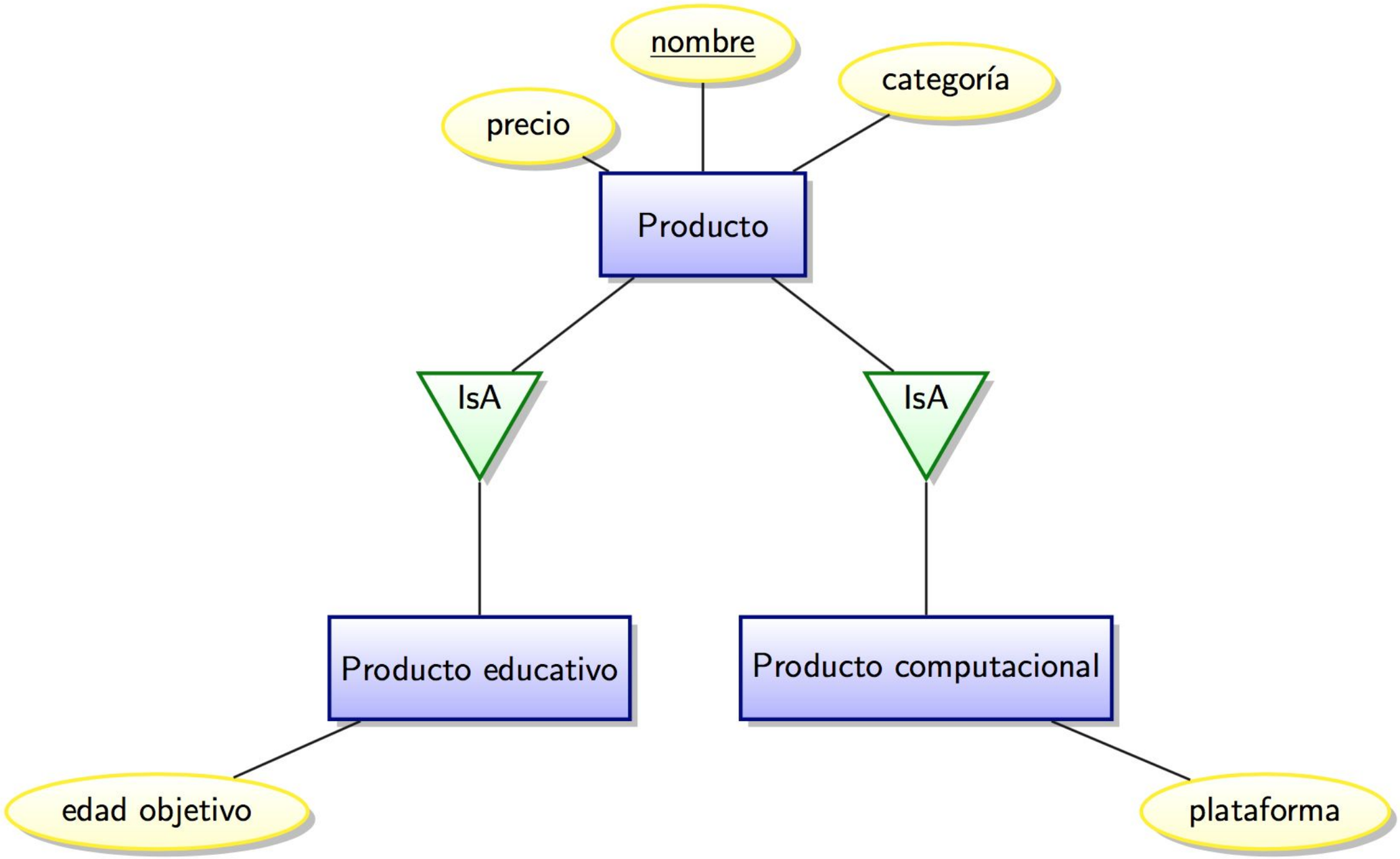
```
CREATE TABLE Usuario(
    id SERIAL,
    email nombre varchar(30) UNIQUE NOT NULL,
    RUT varchar(30) UNIQUE NOT NULL,
    fecha date,
    PRIMARY KEY (id)
)
```

Subclases (IsA)

- Algunas entidades son casos especiales de otra
- Similar a la herencia en orientación a objetos
  - Ej. Todo estudiante es también una persona

- En E/R usamos **IsA** (EsUn en Inglés)

Subclases (IsA)



Subclases (IsA)

Sin herencia

Mantener sólo las entidades hijo con todos los atributos de la entidad padre:

ProductoEducativo(id, nombre, edad_objetivo, precio, categoría)

ProductoComputacional(id, nombre, plataforma, precio, categoría)

Ventajas: No se necesitan joins para acceder todas las columnas
Desventaja: No puedo tener productos genéricos ni solapados (en la práctica si podríamos querer un producto educacional y computacional a la vez)

Subclases (IsA)

Herencia con tablas múltiples.

- Se hacen tablas para todas las entidades y las entidades hijo tienen una referencia a la entidad padre.
- Las entidades hijo tienen sólo los atributos adicionales
- Para obtener la versión completa hago un join

Ventaja: puedo tener productos genéricos y solapados.

Desventaja: se requieren joins para acceder a todos los datos.

Subclases (IsA)

Herencia con tablas múltiples.

Producto(id, nombre, precio, categoría)

ProductoEducativo(id, nombre, edad_objetivo, id_producto)

ProductoComputacional(id, nombre, plataforma, id_producto)

Subclases (IsA)

Herencia con una sola tabla.

- Se hace una sola tabla que tiene las columnas tanto del padre como de los hijos.
- Se puede agregar un columna tipo para guardar a qué entidad pertenece cada fila.

Ventaja: No se requieren joins y se pueden tener productos solapados.

Desventaja: Pueden quedar muchos nulos en las tablas y estos no se pueden evitar, aunque la entidad hija a la que corresponda el atributo nunca lo tenga vacío.

Subclases (IsA)

Herencia con una sola tabla.

Producto(nombre, precio, categoría, edad_objetivo, plataforma, tipo)

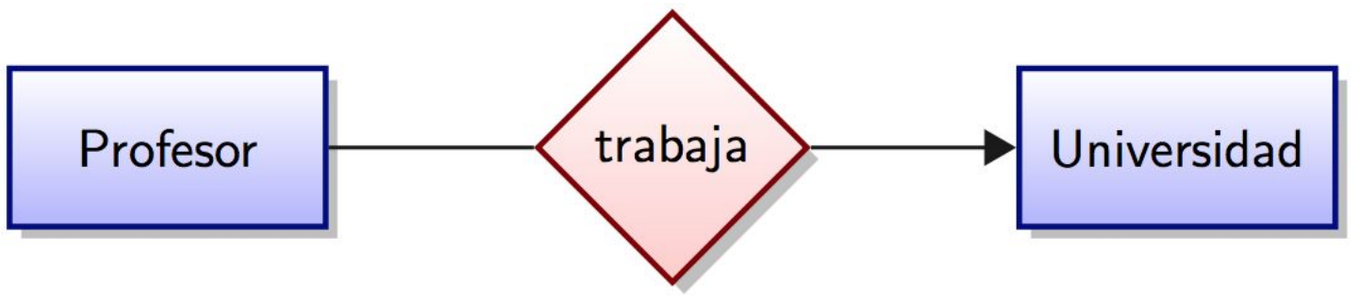
Restricciones de integridad

Son restricciones que permiten que la base de datos se mantenga íntegra en el tiempo. Algunas son:

- Integridad de la entidad: el RUT es valor único y no puede ser nulo.
- Participación: toda compañía debe tener al menos un empleado
- Referenciales: si se trabaja en una compañía, esta debe existir (Llaves foráneas).
- Del dominio: la edad de las personas debe estar entre 0 y 150 años.

Restricciones de integridad

Integridad de la entidad



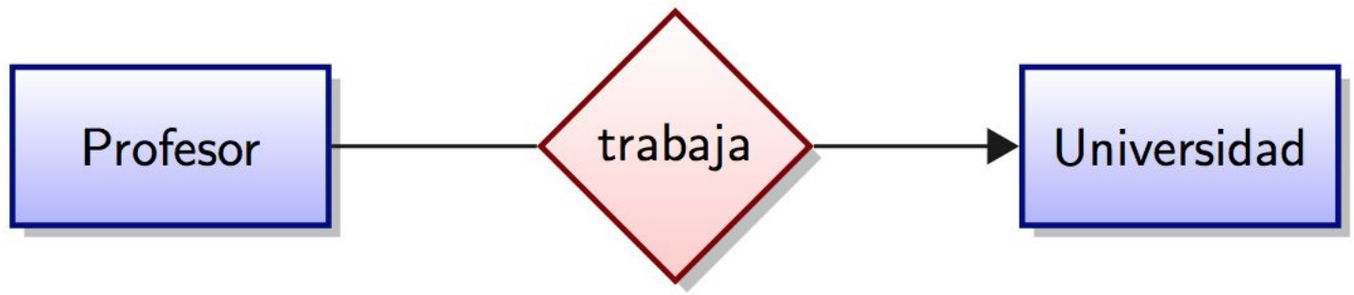
Un ejemplo de tabla de profesor con algunas restricciones:

```
CREATE TABLE Profesor(
    id int PRIMARY KEY,
    nombre varchar(30) NOT NULL,
    apellidos varchar(30) NOT NULL,
    telefono varchar(30) NOT NULL,
    id_universidad int,
    nivel varchar(20) DEFAULT 'Pregrado'
    FOREIGN KEY(id_universidad) REFERENCES Universidad(id)
)
```

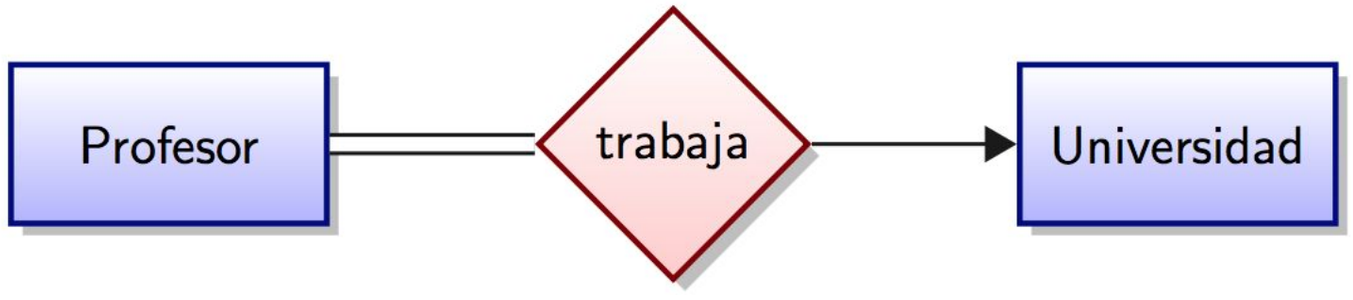
Restricciones de integridad

Participación

Cada profesor puede trabajar en una única universidad (pero puede estar sin trabajo!):



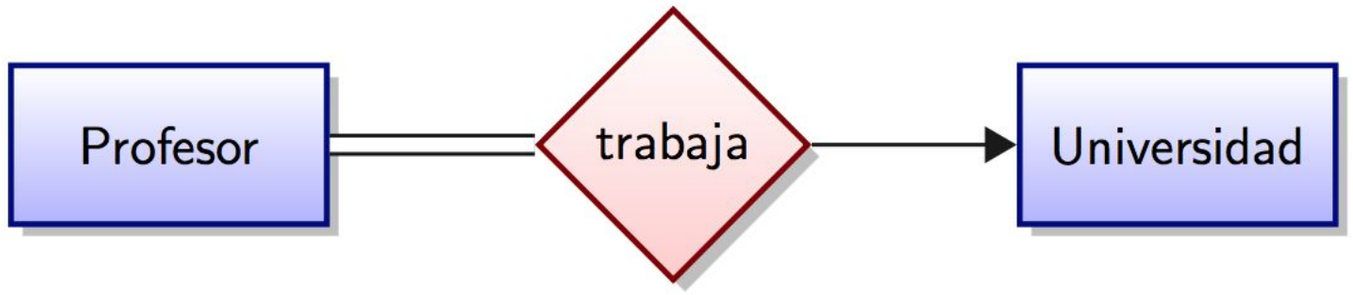
Cada profesor necesariamente trabaja en una única universidad:



Restricciones de integridad

Participación

Cada profesor necesariamente trabaja en una única universidad

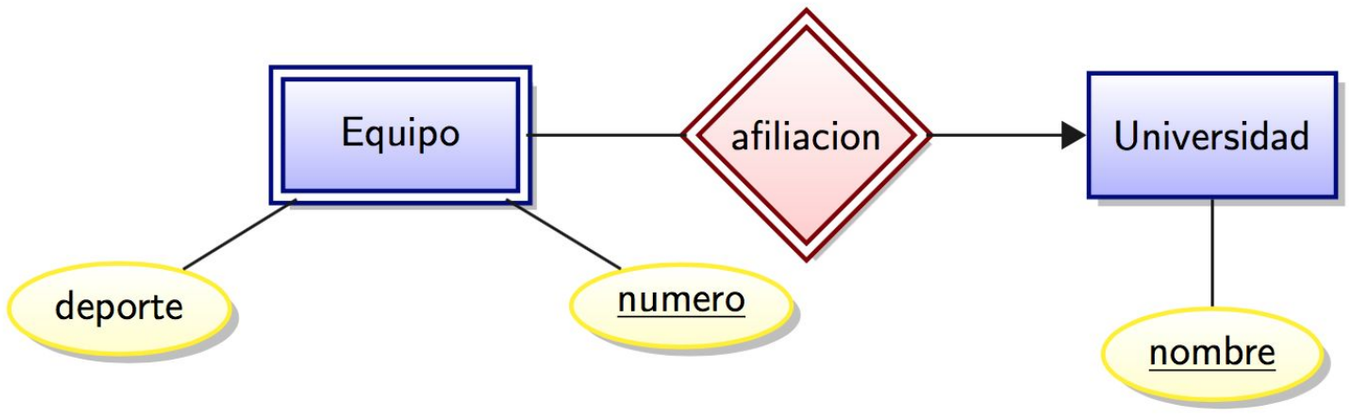



```
CREATE TABLE Profesor(
    id int PRIMARY KEY,
    nombre varchar(30) NOT NULL,
    apellidos varchar(30) NOT NULL,
    telefono varchar(30) NOT NULL,
    id_universidad int NOT NULL,
    nivel varchar(20) DEFAULT 'Pregrado'
    FOREIGN KEY(id_universidad) REFERENCES Universidad(id)
)
```

Restricciones de integridad

Entidades débiles

Le llamamos entidad débil a aquellas que necesitan de una relación para existir. En este caso, el equipo no puede existir sin estar asociado a una universidad.




```
CREATE TABLE Equipo(
    id int PRIMARY KEY,
    nombre varchar(30) NOT NULL,
    id_universidad int NOT NULL,
    FOREIGN KEY(id_universidad) REFERENCES Universidad(id) ON DELETE CASCADE
)
```

Restricciones de integridad

Dominio

Queremos restringir el dominio de las columnas. Una forma simple de hacer esto en SQL es con CHECK:

```
CREATE TABLE Festival(
    id int PRIMARY KEY,
    nombre varchar(30) NOT NULL,
    fecha_inicio date NOT NULL,
    fecha_fin date NOT NULL,
    precio int NOT NULL,
    CHECK( precio BETWEEN 10000 AND 1000000 ),
    CHECK( fecha_fin > fecha_inicio)
)
```