Subclases, integridad referencial y relaciones de apoyo

Ayudantía 6 · IIC2413 Bases de Datos

1. Subclases (`isa`)

El problema

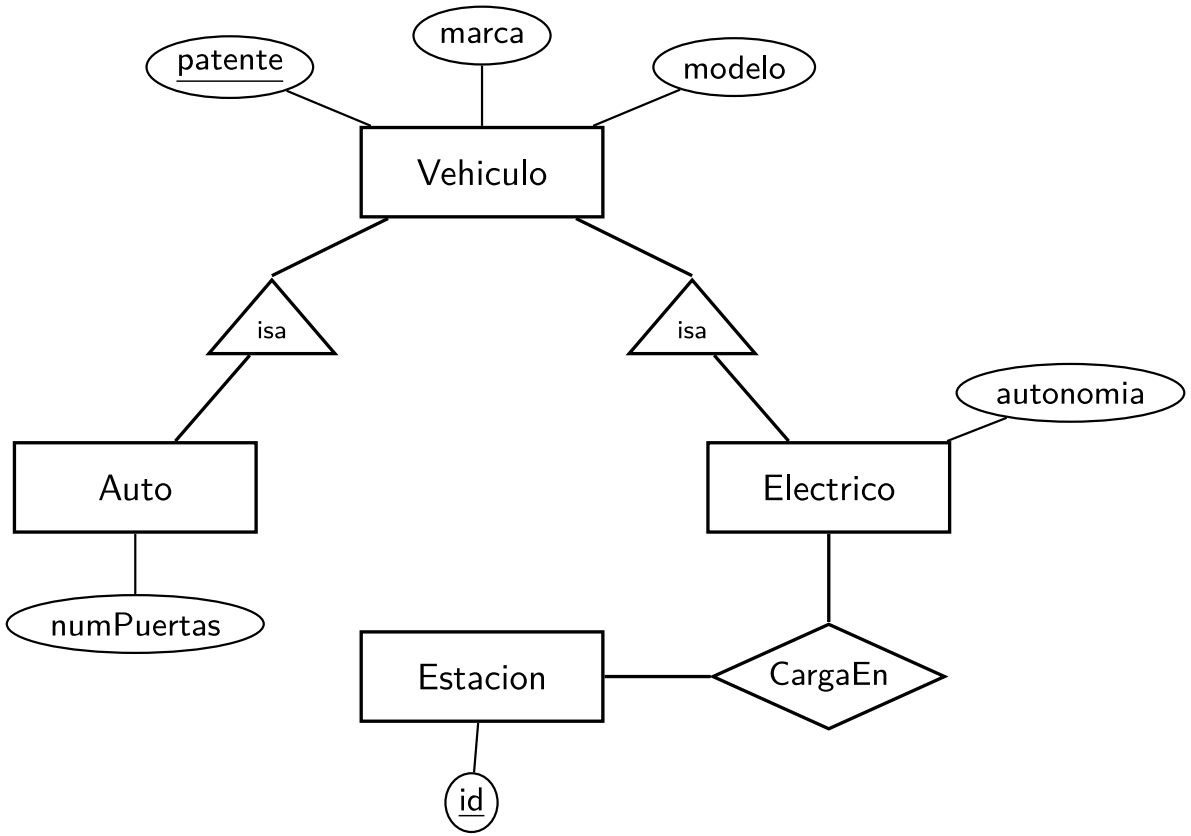
Una empresa de arriendo guarda sus **vehículos**: patente, marca y modelo.

- Los **autos** además tienen número de puertas.
- Los **eléctricos** además tienen autonomía y se cargan en ciertas **estaciones**.
- Una camioneta diésel no tiene ninguna de esas cosas.

> Si le ponemos *numPuertas* y *autonomía* a **todos** los vehículos, casi siempre quedan vacíos, y la relación con estaciones no tiene sentido para un auto a bencina.

> **Idea:** separar los casos especiales en **subclases**, cada una con lo que le es propio.

Cómo se dibuja



Un **triángulo** `isa` : la **punta apunta a la superclase** (la más general). Se lee *"un Auto **es un** Vehículo"*.

Qué significa `isa`

- Toda entidad de la subclase **también es** de la superclase.
- La subclase **hereda todo**: la llave, los atributos y las relaciones (un Auto también tiene patente, marca y modelo).
- La subclase **no tiene llave propia**: usa la de la raíz (*patente*).
- La subclase **agrega lo suyo**: atributos (*numPuertas*) o relaciones (*CargaEn*).
- `isa` es uno-a-uno, pero **no se dibujan flechas**: el triángulo ya lo dice.
- Las jerarquías son **árboles**: una sola raíz, y una subclase puede tener sus propias subclases.

¿Dónde “vive” cada vehículo?

En E/R, una entidad tiene un pedazo en cada caja a la que pertenece, **siempre incluyendo la raíz**.

Vehículo	Vehiculo	Auto	Electrico	Tiene
Toyota Hilux (diésel)	✓			sólo patente, marca, modelo
Toyota Yaris	✓	✓		+ numPuertas
Moto eléctrica	✓		✓	+ autonomía, CargaEn
Nissan Leaf	✓	✓	✓	+ todo lo anterior

> El Leaf es auto **y** eléctrico a la vez. En E/R no hay problema: tiene un pedazo en cada subclase. En programación orientada a objetos habría que inventar una clase *AutoEléctrico*.

Dos preguntas sobre cualquier jerarquía

¿Completa o parcial?

¿Toda entidad de la superclase está en *alguna* subclase?

Vehículos: **parcial** (la Hilux no está en ninguna).

¿Disjunta o con solapamiento?

¿Puede una entidad estar en *dos* subclases a la vez?

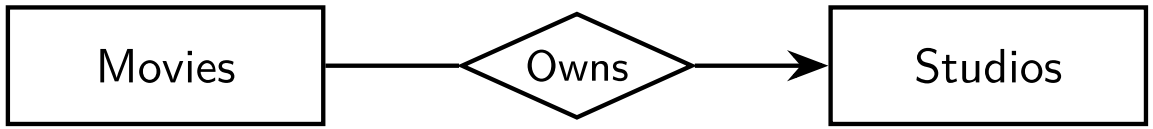
Vehículos: **con solapamiento** (el Leaf).

No cambian el dibujo, pero sí qué estrategia conviene para pasar a tablas.

2. Integridad referencial

RECORDATORIO DE LA CLASE

La flecha de varios-a-uno



Cada película tiene **a lo más un** estudio dueño.

En tablas se absorbe en *Movies* como la columna *nameStudios*, con `FOREIGN KEY` hacia *Studios*.

Lo que la flecha *no* dice



Una app de delivery: cada pedido es de a lo más un cliente, y tiene a lo más un repartidor.

- Pero “a lo más uno” permite **cero**: según el diagrama, podría existir un pedido **sin cliente**.
- Tampoco dice que ese cliente **exista** en la base de datos.

La flecha curva: “exactamente uno, y existe”



> Una **punta curva** hacia F dice que cada E se relaciona con **exactamente un** F, y que ese F **existe** en la base de datos. Eso es **integridad referencial**.

- **HechoPor → Cliente (curva)**: no existe un pedido sin cliente.
- **AsignadoA → Repartidor (normal)**: un pedido recién hecho todavía no tiene repartidor.

¿Curva o normal? La pregunta clave

> ¿Tiene sentido que exista un **E** sin su **F**?

No → curva

- Pedido sin cliente
- Factura sin empresa emisora
- Comentario sin publicación

Frases: *“exactamente un”, “todo”, “siempre”, “obligatorio”, “debe tener”*

Sí → normal

- Pedido sin repartidor (aún)
- Empleado sin jefe (el gerente)
- Alumno sin tutor asignado

Frases: *“puede no tener”, “opcional”, “a lo más”, “si tiene”*

Restricciones de grado

También se puede poner un **número** en una arista: *“un repartidor lleva a lo más 3 pedidos a la vez”*.



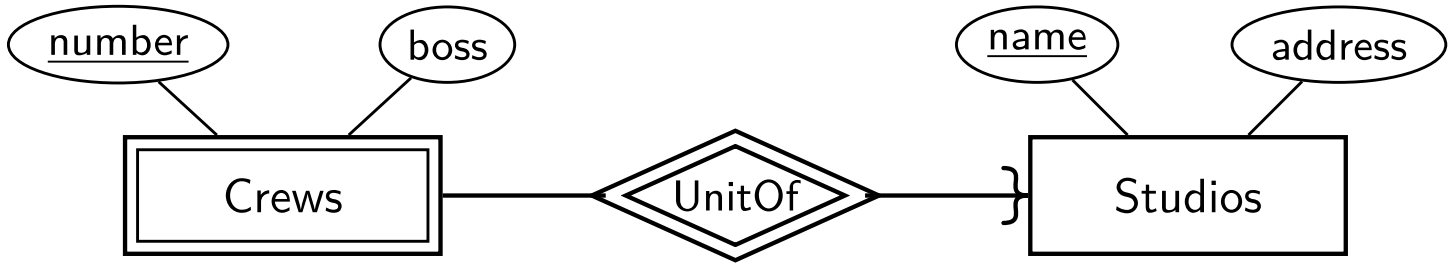
La etiqueta va junto a la arista del lado que se está limitando (aquí, Pedido).

Dibujo	Restricción	Se lee
———	ninguna	varios
——→	≤ 1	a lo más uno
——→|	= 1	exactamente uno, y existe

3. Relaciones de apoyo

RECORDATORIO DE LA CLASE

Entidades débiles



- Una **entidad débil** (rectángulo doble) no tiene llave propia: el número de escuadrilla sólo es único *dentro de su estudio*.
- La **relación de apoyo** (rombo doble) le presta la llave que le falta.
- Llave de Crews: `(name, number)`.

Pero, ¿**cualquier** relación puede ser de apoyo? **No.** El libro pone 5 condiciones.

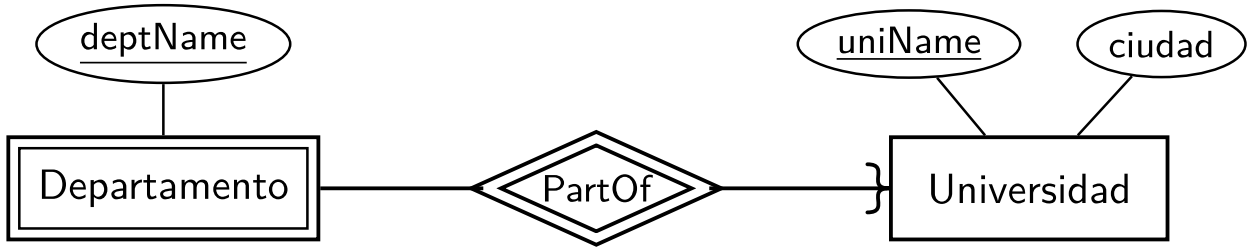
Las 5 condiciones del libro

Para que **R**, que va de la entidad débil **E** a otra entidad **F**, sea relación de apoyo de E:

> **(a)** R es **binaria y varios-a-uno de E a F** (uno-a-uno también cuenta).
> **(b)** R tiene **integridad referencial de E a F**: flecha **curva** hacia F.
> **(c)** Los atributos que F aporta a la llave de E son **atributos de llave de F**.
> **(d)** Si F también es **débil**, parte de lo que aporta viene de *su* apoyo, y así **en cadena**.
> **(e)** Si hay **varias** relaciones de apoyo hacia la misma F, cada una aporta **su propia copia** de la llave de F.

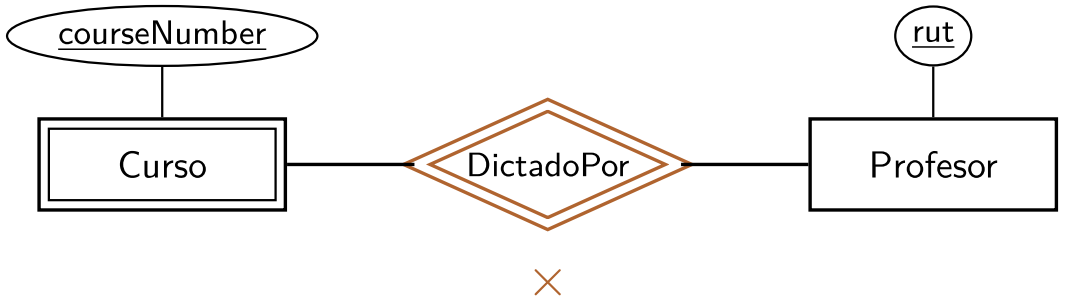
(a), (b) y (c) son requisitos; (d) y (e) explican cómo se arma la llave en casos más complejos.

(a) Binaria y varios-a-uno



Un departamento se llama “Computación” *dentro de* su universidad. Para identificarlo necesitamos **un único** valor de *uniName*.

> **Contraejemplo:** si fuera varios-a-varios, un curso podría estar ligado a **varios** profesores. ¿Cuál de ellos pone su rut en la llave? No hay un valor único.



(b) Integridad referencial y (c) atributos de llave

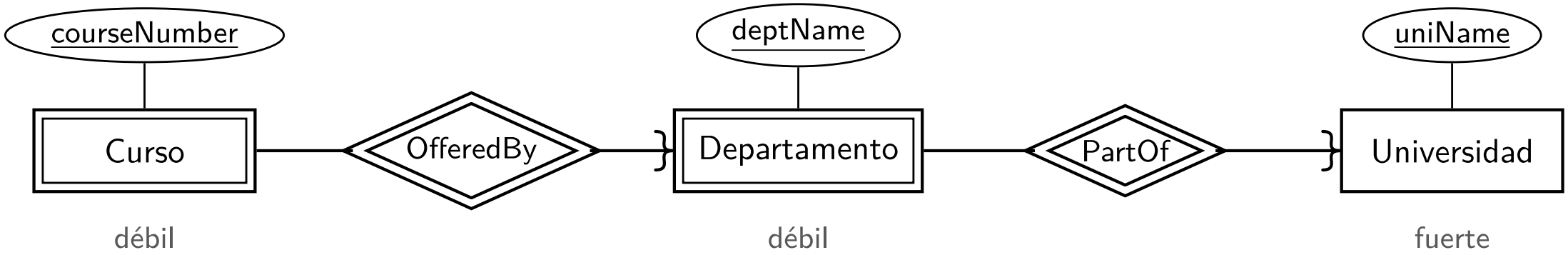
> **(b) Flecha curva**
>
> Todo departamento tiene **exactamente una** universidad, y esa universidad **existe**.
>
> Si pudiera no tener universidad, la llave `(uniName, deptName)` tendría un `NULL`, y una llave nunca puede ser nula.

> **(c) Sólo atributos de llave**
>
> Universidad le aporta *uniName* (su llave), **no** *ciudad*.
>
> Si usáramos la ciudad, dos departamentos de universidades distintas en la misma ciudad podrían chocar.

En tablas: `Departamento(uniName, deptName)`, con uniName como **FK dentro de la PK**. La relación de apoyo **no** genera tabla.

(d) La debilidad se propaga en cadena

Ahora un curso sólo es único *dentro de su departamento*… que a su vez es débil:



- **Curso** necesita la llave de **Departamento**…
- …pero la llave de Departamento **no** es sólo *deptName*: es `(uniName, deptName)`…
- …porque Departamento necesita a **Universidad**, que ya tiene llave propia. Ahí se acaba la cadena.

(d) Cómo quedan las llaves

Entidad	Tipo	Llave completa
Universidad	fuerte	`(uniName)`
Departamento	débil	`(uniName, deptName)`
Curso	débil	`(uniName, deptName, courseNumber)`

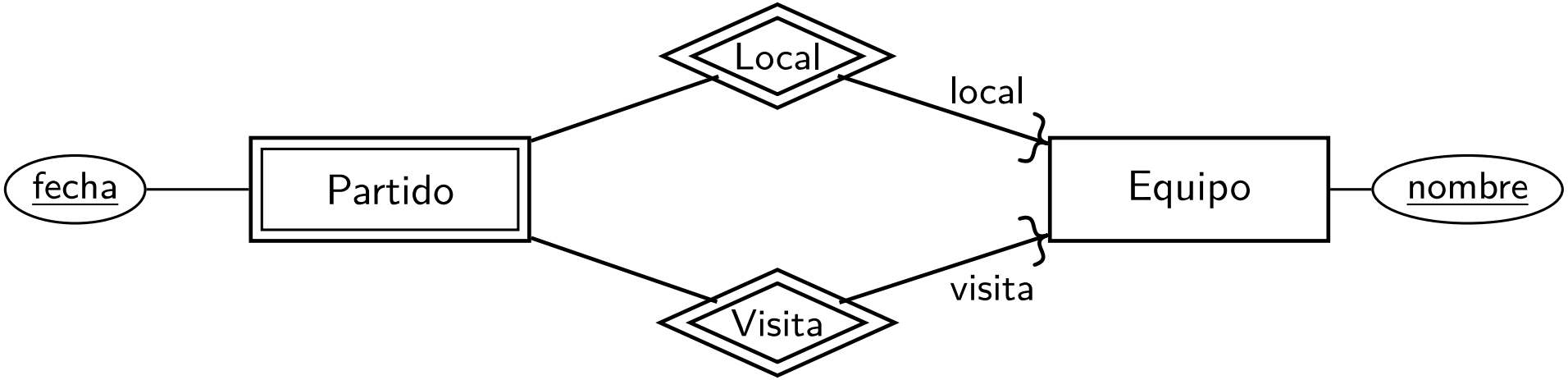
> Una entidad débil hereda la **llave completa** de su dueño, no sólo los atributos dibujados dentro de él. Si esa llave ya trae atributos de más arriba, también pasan hacia abajo.

En SQL basta referenciar al dueño **directo**:

```
FOREIGN KEY (uniName, deptName) REFERENCES Departamento(uniName, deptName)
```

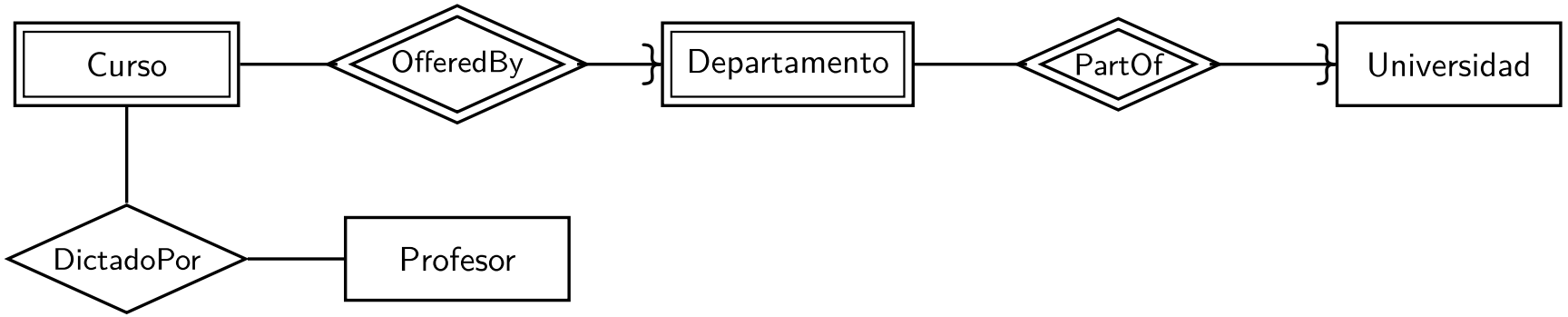
(e) Varias relaciones de apoyo hacia la misma entidad

Un partido se identifica por su equipo local, su equipo visita y la fecha:



- Hay **dos** relaciones de apoyo hacia **Equipo**: cada una aporta **su propia copia** de *nombre*.
- Un mismo partido está ligado a **dos equipos distintos**, así que en la llave aparecen dos nombres de equipo.
- Llave: `(nombreLocal, nombreVisita, fecha)`. Las columnas se renombran según el **rol**.

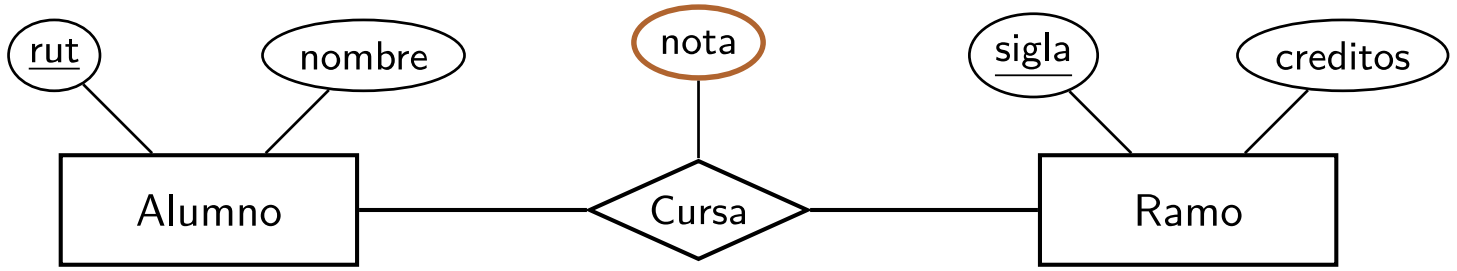
¿Rombo doble ⇔ rectángulo doble?



- **Rombo doble ⇒ rectángulo doble, siempre.** Una relación de apoyo sólo existe para completar la llave de una entidad débil.
- **Rectángulo doble ⇒ al menos un rombo doble.** Una entidad débil siempre necesita a alguien que le preste llave.
- Pero una entidad débil **también puede estar en rombos simples** (*DictadoPor*) que no aportan a su llave.
- Y el dueño puede ser **débil** (Departamento) o **fuerte** (Universidad). Toda cadena termina en uno fuerte.

4. ¿Atributo de la entidad o de la relación?

¿Dónde va cada atributo?



> Pregunta clave: ¿el dato describe a la entidad **por sí sola**, o a la **combinación** de entidades que participan en la relación?

Test: fije una entidad y cambie la otra. Si el valor puede cambiar, es de la **relación**.

Dato	¿Cambia si cambio el ramo?	¿Cambia si cambio el alumno?	Va en
nombre del alumno	no	—	**Alumno**
créditos del ramo	—	no	**Ramo**
nota	sí	sí	**Cursa** (la relación)

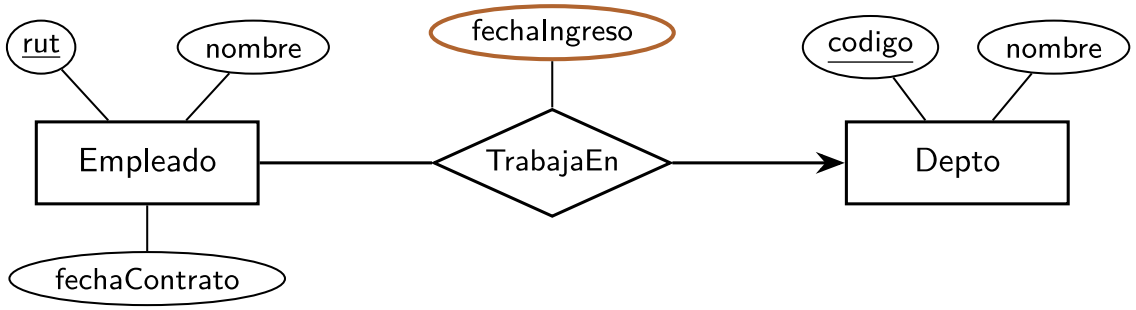
Casos especiales

> **1. El mismo par se repite.** Si un alumno puede cursar el mismo ramo *varias veces* (reprobó y lo repitió, con una nota cada vez), la relación no alcanza: cada par aparece a lo más una vez. Hay que convertirla en **entidad** (débil, identificada también por el semestre).

> **2. Datos derivados.** El promedio de notas de un ramo sí describe al ramo, pero se calcula con `AVG` sobre Cursa. Guardarlo aparte es redundante.

5. Del diagrama a las dependencias funcionales

Al pasar a tablas: el atributo “viaja”



En el diagrama, *fechaIngreso* va en la **relación**. Pero al pasar a tablas, la varios-a-uno se absorbe y la columna termina en **Empleado**: *Empleado(rut, nombre, fechaContrato, codDepto, fechaIngreso)*.

En una **varios-a-varios** el atributo no puede viajar: un alumno tiene muchas notas, así que *nota* se queda en la tabla propia *Cursa(rut, sigla, nota)*.

Cada flecha es una dependencia funcional

> Una relación varios-a-uno **E → F** es una dependencia funcional: **llave(E) → llave(F)**.

¿Por qué? Porque cada E se relaciona con **a lo más un** F. Si dos filas tienen la misma llave de E, se refieren a la misma entidad de E, y entonces tienen que tener el mismo F. Eso es exactamente la definición de dependencia funcional.

Vale igual para la flecha **normal** y la **curva**: ambas dicen “a lo más uno”, y de ahí sale la DF. Lo que la curva agrega (que F exista) es integridad referencial, `NOT NULL` + `FOREIGN KEY`, no una DF.

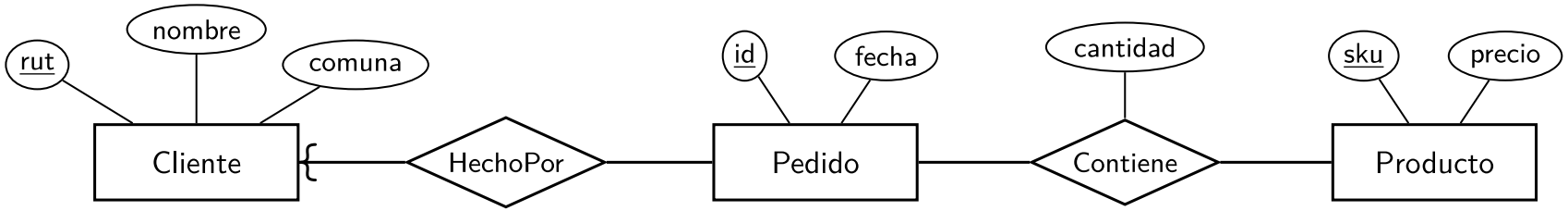
Esto sirve para dos cosas: **(1)** encontrar la llave de la tabla de una relación, y **(2)** revisar si el esquema queda en **BCNF**.

Recordatorio (clase 9): R está en BCNF si, para toda DF no trivial A → B, A es superllave de R.

Del diagrama a las DFs

En el diagrama	Dependencia funcional
La llave de una entidad	llave → resto de sus atributos
Varios-a-uno E → F	llave(E) → llave(F)
Uno-a-uno (flecha hacia ambos lados)	en ambos sentidos: la tabla queda con dos llaves
Flecha de una k-aria hacia E_i	llaves de las otras entidades → llave(E_i)
Atributo de una relación	llaves de las entidades participantes → atributo

Un ejemplo



DFs que se leen del diagrama

- Llaves de entidades: `rut → nombre, comuna`, `id → fecha`, `sku → precio`
- Varios-a-uno HechoPor: `id → rutCliente`
- Atributo de Contiene: `id, sku → cantidad`

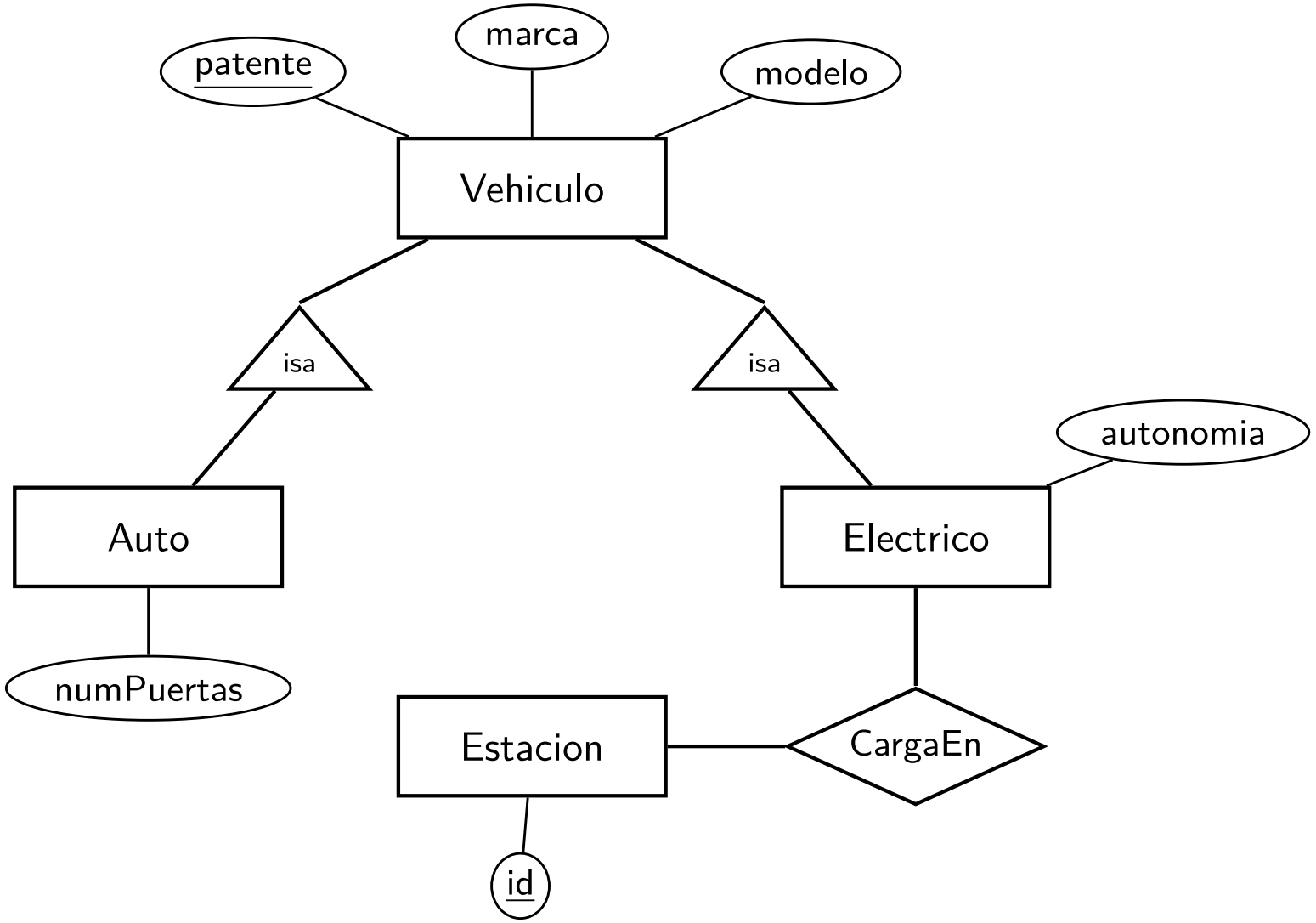
Tablas resultantes

- *Cliente(rut, nombre, comuna)*
- *Pedido(id, fecha, rutCliente)*: HechoPor se absorbe
- *Producto(sku, precio)*
- *Contiene(id, sku, cantidad)*: varios-a-varios, tabla propia

En cada una, la única DF tiene la llave a la izquierda: todas están en **BCNF**.

6. De `isa` a tablas

Tres estrategias, cada una buena para algo distinto.


marca
patente
modelo
Vehiculo
isa
isa
autonomia
Auto
Electrico
numPuertas
Estacion
CargaEn
id

Estrategia 1: estilo E/R

> Una tabla **por cada caja**. Cada tabla lleva la **llave de la raíz** + sólo sus atributos **propios**.

patente	marca	modelo
HJKL34	Toyota	Hilux
ABCD12	Toyota	Yaris
ZXCV78	Super Soco	TC
PQRS56	Nissan	Leaf

patente	numPuertas
ABCD12	5
PQRS56	5

patente	autonomia
ZXCV78	80
PQRS56	270

Tablas: **Vehiculo**, **Auto** y **Electrico**. El Leaf aparece en las **tres**; entre ellas sólo se repite la llave. `isa` **no** genera tabla. *CargaEn(patente, idEstacion)* apunta a **Electrico**, así que sólo los eléctricos pueden cargar.

Estrategia 1 en SQL

La llave de la subclase es, a la vez, llave foránea hacia la superclase:

```
CREATE TABLE Electrico ( patente CHAR(6), autonomia INT,
    PRIMARY KEY (patente),
    FOREIGN KEY (patente) REFERENCES Vehiculo(patente) ON DELETE CASCADE );
```

Si se borra el vehículo, también debe desaparecer su pedazo “eléctrico”: `CASCADE`.

Estrategia 2: orientada a objetos

> Una tabla **por cada combinación posible** de clases, con **todos** sus atributos. Cada entidad está en **una sola** tabla.

Vehiculo

patente	marca	modelo
HJKL34	Toyota	Hilux

VehiculoA

patente	marca	modelo	numPuertas
ABCD12	Toyota	Yaris	5

VehiculoE

patente	marca	modelo	autonomia
ZXCV78	Super Soco	TC	80

VehiculoAE

patente	marca	modelo	numPuertas	autonomia
PQRS56	Nissan	Leaf	5	270

Con **n** subclases hay **2^n** combinaciones: con 2 son 4 tablas; con 5, ¡32! Si la jerarquía es **disjunta** (ninguna entidad está en dos subclases a la vez), desaparecen las combinaciones con dos subclases; si además es **completa** (toda entidad está en alguna subclase, ninguna queda sólo en la raíz), basta una tabla por hoja.

Estrategia 3: con nulos

> Una **única** tabla con **todos** los atributos de la jerarquía. Donde no aplica: `NULL`.

patente	marca	modelo	numPuertas	autonomia
HJKL34	Toyota	Hilux	NULL	NULL
ABCD12	Toyota	Yaris	5	NULL
ZXCV78	Super Soco	TC	NULL	80
PQRS56	Nissan	Leaf	5	270

> Si no sabemos la autonomía de un eléctrico, su fila queda igual que la de un auto a bencina: un `NULL` puede significar “no aplica” o “no se sabe”. En la práctica se agrega una columna de tipo (*esElectrico*).

¿Cuál conviene?

	E/R	OO	Nulos
Tablas (n subclases directas)	n + 1	2^n	1
Filas por entidad	una por caja	1	1
Espacio	repite la llave	mínimo	muchos NULL
“Patentes de todos los Toyota”	1 tabla	unir 4 tablas	1 tabla
“Modelo y autonomía de los eléctricos”	join de 2 tablas	unir 2 tablas	1 tabla
FK que apunte sólo a eléctricos	**se puede**	**repartidos en 2 tablas**	**no se distingue**

- **Solapamiento**, o relaciones que apuntan a una subclase: **E/R**.
- **Disjunta** (y mejor aún si es completa): **OO**.
- Pocas tablas y consultas simples, aceptando nulos: **nulos**.

7. Recordatorio: `ON DELETE`

RECORDATORIO DE LA CLASE

`ON DELETE`: qué pasa al borrar

Se escribe en cada `FOREIGN KEY` de la tabla **hija** (la que apunta), y se activa al borrar una fila de la tabla **padre** (a la que apunta).

Opción	Al borrar la fila padre…	Úsela cuando la hija…
`NO ACTION` (default)	**Error**: no se borra mientras haya hijas	no debería perderse
`CASCADE`	Se borran **también** las hijas	no tiene sentido sin la padre (débil, subclase)
`SET NULL`	Las hijas quedan, con la FK en `NULL`	puede quedar “sin” la padre (flecha normal)

> `SET NULL` falla si la columna es parte de la **llave primaria** o es `NOT NULL`.

`ON DELETE`: un ejemplo

```
CREATE TABLE Publicacion ( id INT, titulo VARCHAR(100),
    autor     VARCHAR(12) NOT NULL,     categoria VARCHAR(30),
    PRIMARY KEY (id),
    FOREIGN KEY (autor)     REFERENCES Usuario(rut)      ON DELETE NO ACTION,
    FOREIGN KEY (categoria) REFERENCES Categoria(nombre) ON DELETE SET NULL );

CREATE TABLE Comentario ( idPublicacion INT, numero INT, texto VARCHAR(500),
    PRIMARY KEY (idPublicacion, numero),
    FOREIGN KEY (idPublicacion) REFERENCES Publicacion(id) ON DELETE CASCADE );
```

Si borro…	Resultado
un usuario con publicaciones	**Error** (`NO ACTION`): no se borra
una categoría	Sus publicaciones quedan con `categoria = NULL`
una publicación	Se borran también sus comentarios

8. La flecha curva en SQL

Flecha curva = `NOT NULL` + `FOREIGN KEY`

La flecha curva se traduce en **llave foránea** + `NOT NULL`:

```
CREATE TABLE Pedido ( id INT, fecha DATE,
    cliente    VARCHAR(12) NOT NULL,   -- curva: tiene que estar
    repartidor VARCHAR(12),            -- normal: puede ser NULL
    PRIMARY KEY (id),
    FOREIGN KEY (cliente)    REFERENCES Cliente(rut)    ON DELETE NO ACTION,
    FOREIGN KEY (repartidor) REFERENCES Repartidor(rut) ON DELETE SET NULL );
```

- `FOREIGN KEY`: el cliente **existe**. `NOT NULL`: el cliente **está**.
- Con flecha curva, **nunca** `SET NULL`: chocaría con el `NOT NULL`.
- El grado “≤ 3” **no** se puede expresar con PK/FK: se anota aparte.