


UNIVERSIDAD NACIONAL DE INGENIERÍA

Bases de Datos

# El modelo entidad-relación

## Diseño de bases de datos

**Gerson Garrido**

Semana 6

# Hoy

**Sesión A: el modelo E-R**

1. El proceso de diseño
2. Entidades, atributos y relaciones
3. Cardinalidades y participación
4. Claves y entidades débiles
5. Atributos redundantes; el diagrama de Chasqui

**Sesión B: de E-R a tablas**

6. Reglas de traducción (una biblioteca)
7. Lo que las tablas no pueden decir
8. E-R extendido: especialización y agregación
9. Decisiones de diseño y otras notaciones

**Lectura**

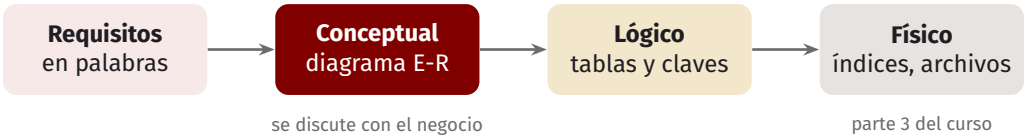
Silberschatz, secs. 6.1 a 6.6 (Sesión A); 6.7, 6.8, 6.9 y 6.10 (Sesión B). Además: Berkeley CS186, Note 13 *DB Design* (*Entity-Relationship Models* y *Relationship Constraints*) y PostgreSQL, *Constraints*.

SECCIÓN 01

# El proceso de diseño

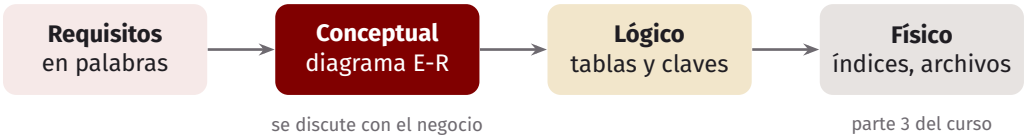
01

# Antes de `CREATE TABLE`




- Dos peligros: redundancia (el mismo dato en dos lugares) e incompletitud (algo del negocio que no se puede representar).
- No hay *el* diagrama de un negocio: hay diagramas mejores o peores para las preguntas que debe responder.

# Antes de `CREATE TABLE`




- Dos peligros: redundancia (el mismo dato en dos lugares) e incompletitud (algo del negocio que no se puede representar).
- No hay *el* diagrama de un negocio: hay diagramas mejores o peores para las preguntas que debe responder.
- Hoy, además, **ingeniería inversa**: reconstruir el diagrama del que salió `schema.sql`.

SECCIÓN 02

# Entidades, atributos y relaciones

02

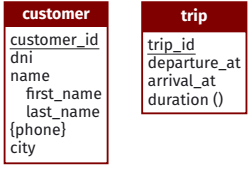
# Entidades y atributos

**Definición**

Una **entidad** es una cosa que se distingue de las demás. Un **conjunto de entidades** *(entity set)* agrupa las del mismo tipo. Un **atributo** es una función que asigna a cada entidad un valor de su **dominio**.

| Eje | Ejemplo |
|---|---|
| simple / **compuesto** | `name (first_name, last_name)` |
| mono / **multivaluado** | `{phone}`: varios teléfonos |
| almacenado / **derivado** | `duration()` de un viaje |
| puede ser **nulo** | `arrival_at` (14 viajes) |

La clave primaria va subrayada.



# Un derivado no se guarda: se calcula

queries.sql

```sql
SELECT trip_id, departure_at, arrival_at, arrival_at - departure_at AS duration
FROM trip
ORDER BY trip_id
LIMIT 3;
```

psql

```
 trip_id |      departure_at      |       arrival_at       | duration
---------+------------------------+------------------------+----------
       1 | 2026-01-05 06:00:00-05 | 2026-01-05 14:30:00-05 | 08:30:00
       2 | 2026-01-05 06:00:00-05 | 2026-01-05 22:45:00-05 | 16:45:00
       3 | 2026-01-05 06:00:00-05 | 2026-01-05 08:15:00-05 | 02:15:00
(3 rows)
```

- Guardar un derivado es guardar dos veces lo mismo: tarde o temprano dejan de coincidir.
- Si se guarda (por rendimiento), alguien tiene que mantenerlo al día: un trigger o la aplicación.

# Conjuntos de relaciones

**Definición**

Sean $E_1, \ldots, E_n$ conjuntos de entidades. Un **conjunto de relaciones** *(relationship set)* $R$ es un subconjunto de $\{(e_1, \ldots, e_n) \mid e_1 \in E_1, \ldots, e_n \in E_n\}$. Cada tupla es una **relación**; $n$ es el **grado**.




- $R$ es un conjunto: la misma combinación de entidades aparece a lo más una vez.
- Una relación vincula entidades, no valores: en el diagrama no hay `sender_id`.

# Conjuntos de relaciones

**Definición**

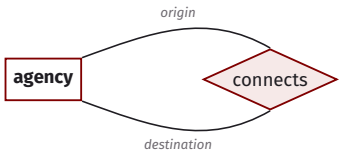
Sean $E_1, \ldots, E_n$ conjuntos de entidades. Un **conjunto de relaciones** *(relationship set)* $R$ es un subconjunto de $\{(e_1, \ldots, e_n) \mid e_1 \in E_1, \ldots, e_n \in E_n\}$. Cada tupla es una **relación**; $n$ es el **grado**.




- $R$ es un conjunto: la misma combinación de entidades aparece a lo más una vez.
- Una relación vincula entidades, no valores: en el diagrama no hay `sender_id`.
- Ojo: *relación* en E-R *(relationship)* es un vínculo (un rombo); en el modelo relacional *(relation)* es una tabla. Hoy, *relación* es la de E-R.

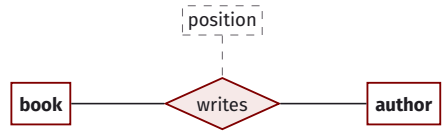
# Roles y atributos descriptivos

**Roles**: el papel de cada participante.




Sin roles, el par (Lima, Cusco) no dice hacia dónde va la ruta. En las tablas se ven en los nombres: `origin_agency_id`, `sender_id`, `recipient_id`.

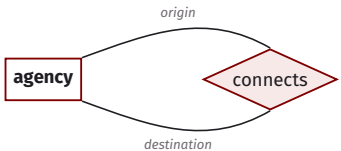
**Atributos descriptivos**: del vínculo, no de un participante.




Silberschatz primero, Korth segundo: es de *ese* autor en *ese* libro.

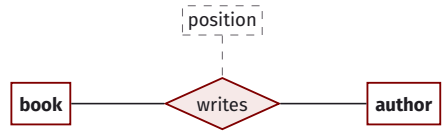
# Roles y atributos descriptivos

**Roles**: el papel de cada participante.




Sin roles, el par (Lima, Cusco) no dice hacia dónde va la ruta. En las tablas se ven en los nombres: `origin_agency_id`, `sender_id`, `recipient_id`.

**Atributos descriptivos**: del vínculo, no de un participante.




Silberschatz primero, Korth segundo: es de *ese* autor en *ese* libro. No distingue relaciones: el mismo par (libro, autor) con otra posición es la misma relación.

SECCIÓN 03

# Cardinalidades y participación

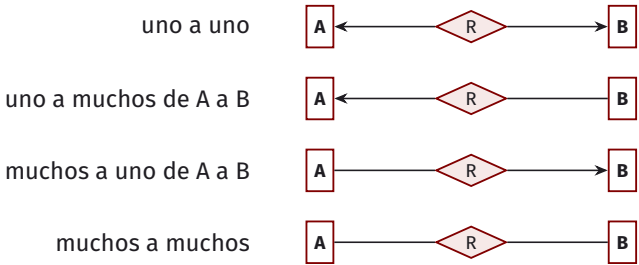
03

# Cardinalidades de mapeo

| Cardinalidad | Cada A, con | Cada B, con |
|---|---|---|
| uno a uno | a lo más un B | a lo más un A |
| uno a muchos de A a B | cero o más B | a lo más un A |
| muchos a uno de A a B | a lo más un B | cero o más A |
| muchos a muchos | cero o más B | cero o más A |

- `sender`: muchos a uno de `shipment` a `customer`.
- `travels_on`: muchos a uno de `shipment` a `trip`.
- `writes`: muchos a muchos entre libros y autores.
- Socio y casillero en una biblioteca: uno a uno.

# Cómo se dibuja: la flecha es “a lo más uno”




- Flecha hacia A: cada B se vincula con a lo más un A.
- En una relación no binaria, a lo más una flecha; con dos, el diagrama tendría dos lecturas.

# Una cardinalidad es una restricción, no una estadística

queries.sql

```sql
SELECT min(n) AS min_shipments,
       max(n) AS max_shipments
FROM (SELECT trip_id, count(*) AS n
      FROM shipment
      WHERE trip_id IS NOT NULL
      GROUP BY trip_id) AS per_trip;
```

psql

```
 min_shipments | max_shipments
---------------+---------------
             1 |            10
(1 row)
```

- “Ningún viaje lleva más de 10” es un dato de esta instancia, no una regla.
- Lo que el esquema impone: `trip_id` es una columna, así que un envío no tiene dos viajes.

**Idea clave**

Una cardinalidad responde “¿qué puede pasar?”, no “¿qué pasa hoy?”. Los datos solo pueden *refutarla*. Se decide con el negocio.

# Participación total y parcial

**Definición**

La participación de *E* en *R* es **total** si toda entidad de *E* participa en al menos una relación de *R*; si no, es **parcial**. Se dibuja con **línea doble**.




- Cada envío: exactamente un remitente (línea doble + flecha del otro lado).
- Cada cliente: cero o más envíos mandados.
- La cardinalidad da el máximo; la participación, el mínimo.

# Participación en Chasqui

shipment_trip_participation

```
 without_trip | shipments
--------------+-----------
          500 |     12000
(1 row)
```

agencies_without_routes

```
 agency_id |       name
-----------+------------------
        29 | Iquitos - Centro
(1 row)
```

trips_without_shipments

```
 trips_without_shipments
-------------------------
                    1038
(1 row)
```

customers_never_sent

```
 never_sent
------------
         82
(1 row)
```

`travels_on`: parcial de los dos lados. `agency` en las rutas y `customer` en `sender`: parciales.

# Participación en Chasqui

shipment_trip_participation

```
 without_trip | shipments
--------------+-----------
          500 |     12000
(1 row)
```

agencies_without_routes

```
 agency_id |       name
-----------+------------------
        29 | Iquitos - Centro
(1 row)
```

trips_without_shipments

```
 trips_without_shipments
-------------------------
                    1038
(1 row)
```

customers_never_sent

```
 never_sent
------------
         82
(1 row)
```

`travels_on`: parcial de los dos lados. `agency` en las rutas y `customer` en `sender`: parciales. Los datos no prueban: `vehicles_without_trips` da 0, pero un camión recién comprado no tiene viajes; `vehicle` en `uses` es parcial.

# La etiqueta $l..h$ del libro

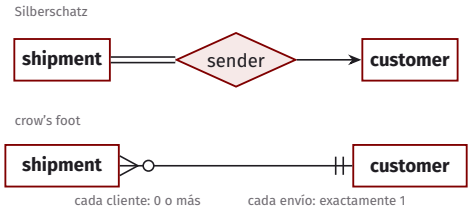



- Mínimo $l$ y máximo $h$ de relaciones de *cada entidad de ese lado*.
- `1..1` junto a `shipment`: cada envío está en exactamente un `sender`.
- `0..*` junto a `customer`: cada cliente, en cero o más.
- $l = 1$ es participación total; $h = 1$ es "a lo más uno".

# La misma relación en crow's foot

| Marca | mín | máx |
|---|---|---|
| dos barras | 1 | 1 |
| círculo y barra | 0 | 1 |
| barra y pata | 1 | muchos |
| círculo y pata | 0 | muchos |

Afuera, junto al rectángulo, el máximo; adentro, el mínimo.




**Error común**

La etiqueta *l*..*h* del libro habla de la entidad **junto a la que está**. En crow's foot y UML la marca junto a `customer` dice cuántos clientes le tocan a cada envío. Primero averigüen la notación.

# Tres notaciones, una relación

LIBRO: FLECHA Y LÍNEA DOBLE




LIBRO: ETIQUETAS L..H




CROW'S FOOT




las marcas junto a shipment hablan de cosas distintas en el libro y en crow's foot

SECCIÓN 04

# Claves y entidades débiles

04

# La clave de un conjunto de relaciones

- $pk(E_1) \cup \cdots \cup pk(E_n)$ es una **superclave** de $R$ (con roles, se renombra).
- Los atributos descriptivos no entran en la clave.
- Que sea minimal depende de la cardinalidad:

| **Cardinalidad de $R$ entre A y B** | **Clave primaria de $R$** |
|---|---|
| muchos a muchos | $pk(A) \cup pk(B)$ |
| muchos a uno de A a B | $pk(A)$ |
| uno a muchos de A a B | $pk(B)$ |
| uno a uno | $pk(A)$ o $pk(B)$ |

`sender`: `shipment_id` basta. `writes`: hacen falta `isbn` y `author_id`. No binaria con una flecha hacia E: la unión de las claves de los demás.

# ¿Qué identifica a un evento?

events_of_shipment_1

```
 shipment_id | seq |          at           | agency_id |   kind
-------------+-----+-----------------------+-----------+-----------
           1 |   1 | 2026-01-28 19:46:00-05 |         9 | received
           1 |   2 | 2026-02-02 12:00:00-05 |         9 | departed
           1 |   3 | 2026-02-02 16:30:00-05 |         8 | arrived
           1 |   4 | 2026-02-04 18:30:00-05 |         8 | delivered
(4 rows)
```

seq_is_not_a_key

```
 events_with_seq_1
-------------------
             12000
(1 row)
```

- `seq = 1` existe en cada envío: los atributos del evento no lo identifican.
- Un evento solo tiene sentido como “el tercer evento *del envío 1*”.

# Entidades débiles

**Definición**

Un conjunto de entidades **débil** depende de un **owner** *(identifying entity set)* por una **relación identificadora**, muchos a uno del débil al owner y **total** del lado débil. El **discriminador** distingue a las débiles de *un mismo* owner. Clave primaria = clave del owner + discriminador.




- El owner no es atributo del débil: `shipment_id` aparece recién en la tabla.
- Rectángulo y rombo dobles; discriminador con subrayado punteado. Clave: (`shipment_id`, `seq`).

# En PostgreSQL: Chasqui no borra en cascada

- El libro traduce la débil con `ON DELETE CASCADE` hacia el owner.
- `schema.sql` no lo declara: se aplica `NO ACTION`.

psql

```
BEGIN;
DELETE FROM shipment WHERE shipment_id = 1;
ROLLBACK;
```

psql

```
ERROR:  update or delete on table "shipment" violates foreign key constraint "shipment_event_shipment_id_fkey" on table "shipment_event"
DETAIL:  Key (shipment_id)=(1) is still referenced from table "shipment_event".
```

Qué debe pasar al borrar cada fila referenciada es una de las decisiones que P1 les pide justificar.

SECCIÓN 05

# Atributos redundantes

05

# Una clave foránea no es un atributo

### Copiado de la tabla

| shipment |
| --- |
| shipment_id |
| sender_id |
| recipient_id |
| origin_agency_id |
| dest_agency_id |
| trip_id |
| weight_kg, price, … |

### En el diagrama

| shipment |
| --- |
| shipment_id |
| weight_kg |
| declared_value |
| price |
| status |
| created_at |
| delivered_at |

- `sender_id` repite lo que dice la relación `sender`: se quita (sec. 6.6).
- La clave foránea aparece sola al traducir a tablas.

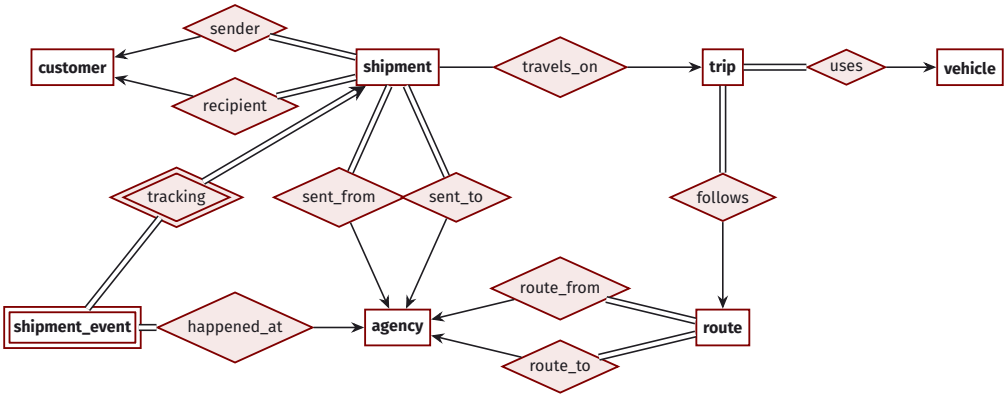
# Redundancias más sutiles: `status`

status_vs_last_event

```
   status   | last_event | shipments
------------+------------+-----------
 delivered  | delivered  |     11115
 in_transit | arrived    |        53
 in_transit | departed   |        13
 registered | received   |       500
 returned   | returned   |       319
(5 rows)
```

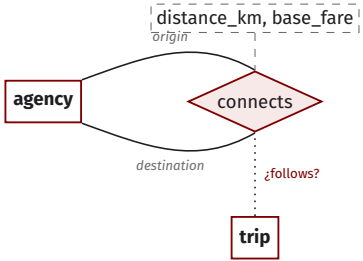
- `status` casi se deriva del último evento del envío (consulta con `JOIN LATERAL` en los apuntes).
- `delivered_at` = el evento `delivered`, en los 11 115 casos.
- Guardarlos es legítimo, pero nada en el esquema los mantiene de acuerdo.

# El diagrama E-R de Chasqui (sin atributos)




Once relaciones, todas muchos a uno. La línea doble del lado “muchos” es exactamente la clave foránea `NOT NULL`; la única parcial es `travels_on` (`trip_id` admite `NULL`).

# ¿Ruta es una entidad o una relación?




- Como relación: muchos a muchos entre agencias, clave = el par. `UNIQUE (origin_agency_id, dest_agency_id)` lo deja ver.
- Pero `trip` se relaciona con una ruta, y una relación no participa en otra.
- Salidas: agregación (Sesión B) o una entidad con `route_id`. Chasqui eligió la segunda.

**Regla práctica**

Cuando algo tiene relaciones propias, tiende a ser una entidad.

SECCIÓN 06

# De E-R a tablas

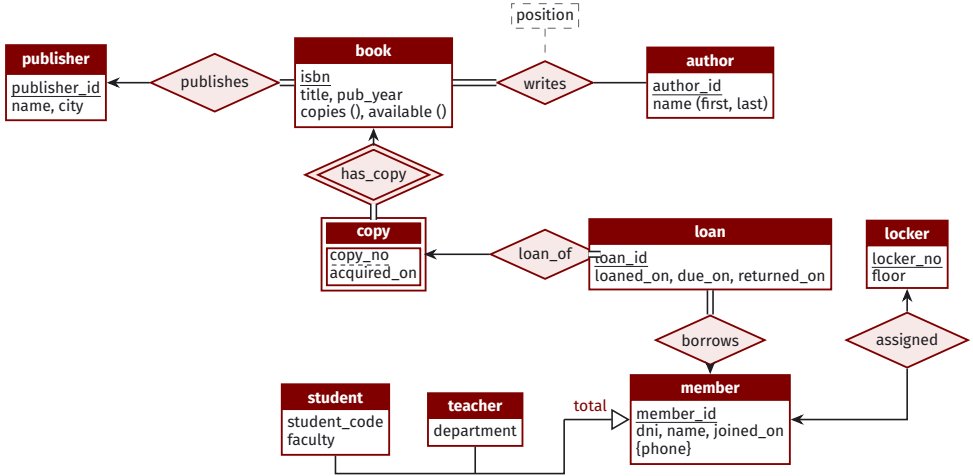
06

# El ejemplo: una biblioteca universitaria

1. **Libro** (ISBN, título, año): una **editorial** y al menos un **autor**; importa el orden de los autores.
2. **Ejemplares** numerados 1, 2, 3 dentro de cada libro. Si el libro se da de baja, sus ejemplares también.
3. **Socios** (DNI, nombre, fecha de alta, teléfonos): **estudiante** (código, facultad) o **docente** (departamento), nunca las dos.
4. **Casilleros**: a lo más uno por socio, a lo más un socio por casillero.
5. **Préstamos**: el mismo socio puede llevarse el mismo ejemplar muchas veces; un ejemplar no está en dos préstamos sin devolver.
6. Cuántos ejemplares tiene cada libro y cuántos están disponibles.

> **Código**
>
> `code/week-06/library.sql`: esquema `library` en `chasqui_w06`, con datos de ejemplo.

# El diagrama de la biblioteca



# Entidades fuertes, compuestos y multivaluados

library.sql

```sql
-- name (first_name, last_name), aplanado
CREATE TABLE author (
  author_id  integer PRIMARY KEY,
  first_name text NOT NULL,
  last_name  text NOT NULL
);
```

- Fuerte → una tabla con la misma clave.
- Compuesto → una columna por componente.

library.sql

```sql
-- {phone}: tabla aparte
CREATE TABLE member_phone (
  member_id integer NOT NULL,
  phone     text NOT NULL,
  CONSTRAINT member_phone_pk
    PRIMARY KEY (member_id, phone),
  CONSTRAINT member_phone_member_fk
    FOREIGN KEY (member_id)
    REFERENCES member ON DELETE CASCADE
);
```

- Multivaluado → tabla (owner, valor); la clave son todas sus columnas.

# Derivados: una vista

library.sql

```sql
CREATE VIEW book_stock AS
SELECT b.isbn, b.title,
  count(c.copy_no) AS copies,
  count(c.copy_no) FILTER (
   WHERE NOT EXISTS (
    SELECT 1 FROM loan l
    WHERE l.isbn = c.isbn
     AND l.copy_no = c.copy_no
     AND l.returned_on IS NULL)
  ) AS available
FROM book b
  LEFT JOIN copy c
    ON c.isbn = b.isbn
GROUP BY b.isbn, b.title;
```

book_stock

```
      isbn      |          title           | copies | available
----------------+--------------------------+--------+-----------
 9786120000011 | Los ríos profundos       |      2 |         1
 9786120000028 | La ciudad y los perros   |      1 |         1
 9786120000035 | Database System Concepts |      3 |         2
 9786120000042 | Trilce                   |      0 |         0
(4 rows)
```

`copies()` y `available()` no se guardan. `LEFT JOIN` y `count(c.copy_no)`: *Trilce* sale con 0.

# Entidad débil: copy

library.sql

```sql
CREATE TABLE copy (
  isbn        text NOT NULL,
  copy_no     integer NOT NULL,
  acquired_on date NOT NULL,
  CONSTRAINT copy_pk
    PRIMARY KEY (isbn, copy_no),
  CONSTRAINT copy_book_fk
    FOREIGN KEY (isbn) REFERENCES book
    ON DELETE CASCADE
);
```

- Clave del owner + discriminador; clave foránea al owner con `CASCADE`.
- La relación identificadora no tiene tabla: `has_copy(isbn, copy_no)` repetiría a `copy`.
- El discriminador se repite entre libros, no dentro de uno:

library-demo.sql

```
psql:library-demo.sql:27: ERROR:  duplicate key value violates unique constraint "copy_pk"
DETAIL:  Key (isbn, copy_no)=(9786120000028, 2) already exists.
23505
```

# El CASCADE se encuentra con `loan`

library-demo.sql

```
DELETE FROM book WHERE isbn = '9786120000028';
SELECT count(*) AS copies_left FROM copy WHERE isbn = '9786120000028';
DELETE FROM book WHERE isbn = '9786120000035';
```

psql

```
DELETE 1
 copies_left
-------------
           0
(1 row)

psql:library-demo.sql:57: ERROR:  update or delete on table "copy" violates foreign key constraint "loan_copy_fk" on table "loan"
DETAIL:  Key (isbn, copy_no)=(9786120000035, 1) is still referenced from table "loan".
23503
```

El borrado en cascada del ejemplar 1 choca con un préstamo: falla la sentencia entera, nada se borra.

# Muchos a muchos: tabla asociativa

library.sql

```sql
CREATE TABLE book_author (
  isbn      text NOT NULL,
  author_id integer NOT NULL,
  position  integer NOT NULL,
  CONSTRAINT book_author_pk
    PRIMARY KEY (isbn, author_id),
  CONSTRAINT book_author_position_uq
    UNIQUE (isbn, position),
  CONSTRAINT book_author_book_fk
    FOREIGN KEY (isbn) REFERENCES book
    ON DELETE CASCADE,
  CONSTRAINT book_author_author_fk
    FOREIGN KEY (author_id) REFERENCES author
);
```

- Claves de los participantes + atributos descriptivos.
- Clave: la del conjunto de relaciones, `(isbn, author_id)`.

**Error común**

`author_ids text` con `'3,4,5'`, o `author1_id`, `author2_id`: sin clave foránea, sin join limpio, y el cuarto autor no tiene dónde ir.

# Fusionar: muchos a uno en el lado “muchos”

library.sql

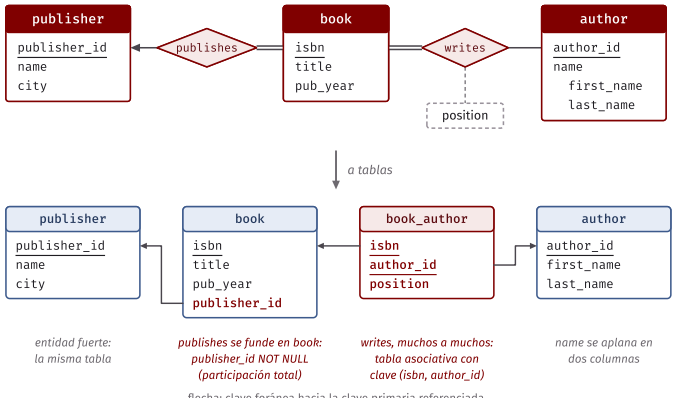
```sql
CREATE TABLE book (
  isbn         text PRIMARY KEY,
  -- title, pub_year
  publisher_id integer NOT NULL,
  CONSTRAINT book_publisher_fk
    FOREIGN KEY (publisher_id)
    REFERENCES publisher
);
```

- `publishes` tendría la clave de `book`: se funde en `book`.
- Total → `NOT NULL`; parcial → `NULL`.
- Uno a uno: con `UNIQUE` (`locker.member_id`).

library-demo.sql

```
psql:library-demo.sql:14: ERROR:  null value in column "publisher_id" of relation "book" violates not-null constraint
23502
psql:library-demo.sql:18: ERROR:  insert or update on table "book" violates foreign key constraint "book_publisher_fk"
23503
psql:library-demo.sql:31: ERROR:  duplicate key value violates unique constraint "locker_member_uq"
23505
```

# Dos relaciones, dos destinos



# Las reglas, y Chasqui al revés

| En E-R | En tablas |
|---|---|
| fuerte | tabla, misma clave |
| compuesto | una columna por componente |
| multivaluado | tabla (owner, valor) |
| derivado | no se guarda; vista |
| débil | clave del owner + discriminador; `CASCADE` |
| identificadora | sin tabla |
| muchos a muchos | tabla asociativa |
| muchos a uno | FK en el lado “muchos” |
| uno a uno | FK con `UNIQUE` |
| ternaria | tabla con las tres claves |

| Columna | Relación |
|---|---|
| `shipment.sender_id` | `sender` |
| `shipment.recipient_id` | `recipient` |
| `shipment.origin_agency_id` | `sent_from` |
| `shipment.trip_id` | `travels_on` (parcial) |
| `trip.route_id` | `follows` |
| `trip.vehicle_id` | `uses` |
| `route.origin_agency_id` | `route_from` |
| `shipment_event.agency_id` | `happened_at` |

Todas `NOT NULL` salvo `trip_id`. Ninguna tabla de Chasqui es asociativa.

# Lo que una clave foránea no puede decir

- Total del lado “uno”: “toda editorial tiene al menos un libro”.
- Total en un muchos a muchos: “todo libro tiene al menos un autor”.
- Al menos un valor de un multivaluado: “todo socio tiene un teléfono”.

**El problema**

“Por cada fila de aquí tiene que existir una fila *allá* que la referencie.” Una clave foránea va en el sentido contrario. No hay `CREATE ASSERTION` en PostgreSQL.

Y un trigger común no sirve: al insertar el libro, sus autores todavía no están (necesitan que el libro exista). La revisión tiene que esperar al `COMMIT`.

# Constraint trigger diferido: dos tablas, dos triggers

library.sql

```
CREATE FUNCTION book_has_author() RETURNS trigger
LANGUAGE plpgsql AS $$
DECLARE
  v_isbn text;
BEGIN
  v_isbn := CASE WHEN TG_TABLE_NAME = 'book' THEN NEW.isbn ELSE OLD.isbn END;
  IF EXISTS (SELECT 1 FROM library.book WHERE isbn = v_isbn)
     AND NOT EXISTS (SELECT 1 FROM library.book_author WHERE isbn = v_isbn) THEN
    RAISE EXCEPTION 'el libro % no tiene autores', v_isbn
      USING ERRCODE = 'check_violation', CONSTRAINT = 'book_has_author';
  END IF;
  RETURN NULL;
END;
$$;
CREATE CONSTRAINT TRIGGER book_has_author_ins AFTER INSERT ON book
  DEFERRABLE INITIALLY DEFERRED FOR EACH ROW EXECUTE FUNCTION book_has_author();
CREATE CONSTRAINT TRIGGER book_has_author_del AFTER DELETE OR UPDATE OF isbn ON book_author
  DEFERRABLE INITIALLY DEFERRED FOR EACH ROW EXECUTE FUNCTION book_has_author();
```

# La transacción falla al `COMMIT`

psql

```
BEGIN;
INSERT INTO library.book VALUES ('9786120000066', 'Poemas humanos', 1939, 3);
COMMIT;
SELECT count(*) FROM library.book WHERE isbn = '9786120000066';
```

psql

```
BEGIN
INSERT 0 1
ERROR:  el libro 9786120000066 no tiene autores
CONTEXT:  PL/pgSQL function library.book_has_author() line 10 at RAISE
 count
-------
     0
(1 row)
```

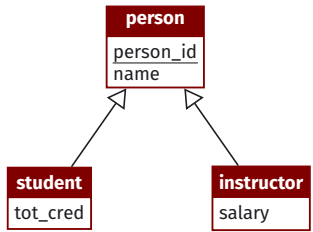
`CREATE CONSTRAINT TRIGGER` es de PostgreSQL. Con el autor en la misma transacción, pasa.

SECCIÓN 07

# E-R extendido

07

# Especialización y generalización



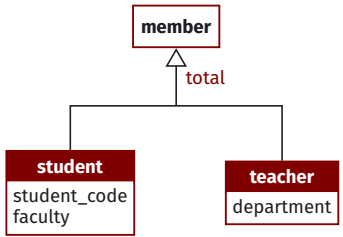

- **Especialización**: de arriba hacia abajo, subgrupos con atributos o relaciones propios.
- **Generalización**: de abajo hacia arriba, juntar lo común. Mismo dibujo.
- Las subclases **heredan** atributos y relaciones: un estudiante tiene `person_id` y `name`.
- Como `Student extends Person` en POO.

# Restricciones de una especialización

- **Disjunta** o **con solapamiento**: ¿una entidad puede estar en dos subclases?
- **Total** o **parcial**: ¿toda entidad está en alguna subclase?

| | **universidad** | **biblioteca** |
|---|---|---|
| ¿dos subclases? | solapamiento | disjunta |
| ¿alguna subclase? | parcial | total |

Solapamiento: una flecha por subclase (diapositiva anterior). Disjunta: una sola flecha.



# Método 1: una tabla por nivel

extended.sql

```sql
CREATE TABLE isa_levels.person (
  person_id integer PRIMARY KEY,
  name      text NOT NULL
);
CREATE TABLE isa_levels.student (
  person_id integer PRIMARY KEY,
  tot_cred  integer NOT NULL,
  CONSTRAINT student_person_fk
    FOREIGN KEY (person_id)
    REFERENCES isa_levels.person
    ON DELETE CASCADE
);
-- instructor, igual, con salary
```

levels_everyone

```
 person_id |    name    | tot_cred | salary
-----------+------------+----------+---------
         1 | Carla Ríos |      120 |
         2 | Jorge Paz  |          | 6500.00
         3 | Elena Soto |       48 | 1800.00
(3 rows)
```

- Sirve siempre: Elena está en las dos subclases.
- Ver todo: un `LEFT JOIN` por subclase.

# Método 2: solo las subclases

- `student(person_id, name, tot_cred)`, `instructor(person_id, name, salary)`.
- El libro lo admite solo si es disjunta y total. Con solapamiento:

extended-demo.sql

```sql
INSERT INTO isa_leaves.instructor
  VALUES (1, 'Carla Rios', 2500);
SELECT person_id, name FROM isa_leaves.student
UNION
SELECT person_id, name FROM isa_leaves.instructor
ORDER BY person_id;
```

psql

```
INSERT 0 1
 person_id |    name
-----------+------------
         1 | Carla Rios
         1 | Carla Ríos
         2 | Jorge Paz
         3 | Elena Soto
(4 rows)
```

Además: ninguna clave foránea puede apuntar a “una persona”, y nada impide el mismo id en las dos tablas.

# Método 3: una sola tabla

extended.sql

```sql
CREATE TABLE isa_single.person (
  person_id     integer PRIMARY KEY,
  name          text NOT NULL,
  is_student    boolean NOT NULL,
  is_instructor boolean NOT NULL,
  tot_cred      integer CHECK (tot_cred >= 0),
  salary        numeric(8, 2) CHECK (salary > 0),
  CONSTRAINT person_student_ck CHECK (is_student = (tot_cred IS NOT NULL)),
  CONSTRAINT person_instructor_ck CHECK (is_instructor = (salary IS NOT NULL))
);
```

extended-demo.sql

```
psql:extended-demo.sql:14: ERROR:  new row for relation "person" violates check constraint "person_instructor_ck"
DETAIL:  Failing row contains (4, Raúl Vega, f, t, null, null).
```

No está en el libro, pero es muy común. Listar a todos es fácil; la tabla se llena de `NULL`.

# Qué conserva cada traducción

| | 1: por nivel | 2: solo subclases | 3: una tabla |
|---|---|---|---|
| Solapamiento | sí | no (datos repetidos) | sí, un indicador por subclase |
| Parcial | sí | no | sí |
| Total | constraint trigger | sí, por construcción | `CHECK (is_student OR is_instructor)` |
| Disjunta | `kind` + FK compuesta | no | columna `kind` |
| `NOT NULL` de una subclase | sí | sí | un `CHECK` por atributo |
| FK hacia la superclase | sí | no | sí |
| FK hacia una subclase | sí | sí | no, salvo con clave compuesta |
| Listar a todos | `LEFT JOIN` por subclase | `UNION` | un `SELECT` |

**En PostgreSQL: `INHERITS` no alcanza**

`CREATE TABLE student (...) INHERITS (person)` parece hecho para esto, pero la documentación (*Inheritance*, *Caveats*) advierte que `UNIQUE`, claves primarias y claves foráneas no se heredan. En el curso usamos tablas normales.

# Disjunta con el método 1: `(member_id, kind)`

library.sql

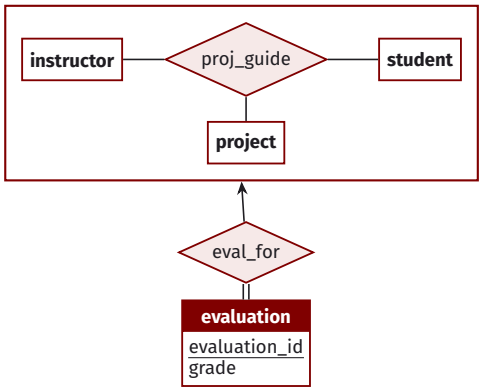
```
CREATE TABLE teacher (
  member_id  integer PRIMARY KEY,
  kind       text NOT NULL DEFAULT 'teacher',
  department text NOT NULL,
  CONSTRAINT teacher_kind_ck
    CHECK (kind = 'teacher'),
  CONSTRAINT teacher_member_fk
    FOREIGN KEY (member_id, kind)
    REFERENCES member (member_id, kind)
    ON DELETE CASCADE
);
```

- El socio tiene un solo `kind`: solo entra en la subclase que le toca.
- `member` tiene `kind` y `UNIQUE (member_id, kind)`, para que la FK pueda apuntar ahí.
- La totalidad necesita un constraint trigger.

library-demo.sql

```
psql:library-demo.sql:39: ERROR:  insert or update on table "teacher" violates foreign key constraint "teacher_member_fk"
DETAIL:  Key (member_id, kind)=(1, teacher) is not present in table "member".
```

# Agregación: una relación que participa en otra




- Lo que se evalúa es *la guía*, no las tres entidades por separado.
- Una cuaternaria permitiría evaluar una combinación que nunca existió.
- La **agregación** trata a `proj_guide` y sus participantes como una entidad.
- Cada evaluación es de exactamente una guía; una guía, cero o más evaluaciones.

# Agregación en tablas: la FK apunta a la relación

extended.sql

```sql
CREATE TABLE university.evaluation (
  evaluation_id integer PRIMARY KEY,
  instructor_id integer NOT NULL,
  student_id    integer NOT NULL,
  project_id    integer NOT NULL,
  evaluated_on  date NOT NULL,
  grade         integer NOT NULL CHECK (grade BETWEEN 0 AND 20),
  CONSTRAINT evaluation_proj_guide_fk FOREIGN KEY (instructor_id, student_id, project_id)
    REFERENCES university.proj_guide
);
```

extended-demo.sql

```
psql:extended-demo.sql:25: ERROR:  insert or update on table "evaluation" violates foreign key constraint "evaluation_proj_guide_fk"
DETAIL:  Key (instructor_id, student_id, project_id)=(11, 21, 31) is not present in table "proj_guide".
```

Los tres existen por separado, pero esa guía no. Con tres FK separadas, la fila habría entrado.

SECCIÓN 08

# Decisiones de diseño

08

# ¿Entidad o atributo? ¿Entidad o relación?

## Entidad o atributo

- El teléfono es atributo mientras sea solo un número; si tiene datos propios (tipo, acepta mensajes), es entidad.
- `city`: atributo; no se guarda nada *de* la ciudad.
- Error común: la clave de otra entidad como atributo (`sender_id` en el diagrama).

## Entidad o relación

- `borrows(member, copy)` con fechas no puede registrar el segundo préstamo del mismo par.
- Por eso `loan` es una entidad, con dos muchos a uno totales.
- Un envío de Chasqui tiene viaje, eventos y agencias: entidad, no relación entre remitente y destinatario.

**Regla**

Si el vínculo se repite entre las mismas entidades, o tiene relaciones propias: entidad.

# En PostgreSQL: un índice único parcial

library.sql

```sql
CREATE UNIQUE INDEX loan_open_copy_uq ON loan (isbn, copy_no) WHERE returned_on IS NULL;
```

library-demo.sql

```
psql:library-demo.sql:35: ERROR:  duplicate key value violates unique constraint "loan_open_copy_uq"
DETAIL:  Key (isbn, copy_no)=(9786120000035, 1) already exists.
23505
```

- “A lo más un préstamo abierto por ejemplar”: una cardinalidad sobre un subconjunto de filas.
- No es SQL estándar ni se escribe como `CONSTRAINT ... UNIQUE`: es un índice, con el mismo 23505.
- Al devolver, la fila sale del índice y el ejemplar se puede volver a prestar.

# ¿Binaria o ternaria?

`prescribes(doctor, patient, drug)` o tres binarias. Salas atiende a Pedro y a Lucía; Pedro toma ibuprofeno; Salas recetó ibuprofeno (a Lucía).

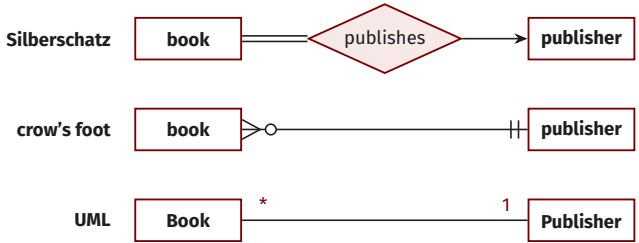
| `treats` | `takes` | `uses` |
|---|---|---|
| (Salas, Pedro)<br>(Salas, Lucía) | (Pedro, ibuprofeno) | (Salas, ibuprofeno) |

- Las binarias son compatibles con “Salas le recetó ibuprofeno a Pedro”, que no pasó.
- La ternaria guarda la tripleta `(Salas, Lucía, ibuprofeno)`.
- Reemplazarla por una entidad artificial con tres binarias es posible, con un id inventado y restricciones extra.

**La pregunta**

¿El hecho necesita a todos los participantes a la vez? Si son dos hechos independientes, la ternaria es un error.

# Una relación, tres dibujos




- El libro anota cada restricción junto a la entidad de la que habla; crow's foot y UML, en el extremo opuesto.
- Crow's foot dibuja un muchos a muchos ya como tabla asociativa: está más cerca del diseño lógico.
- dbdiagram.io y DBML: `publisher_id integer [not null, ref: > publisher.publisher_id]`.

SECCIÓN 09

# Cierre

09

# El código de la semana

```
make test
```

```
$ make test
chasqui-load chasqui_w06
Base de datos chasqui_w06 lista.
...
✓ método 1, 23503: un estudiante que no es una persona
...
✓ 23514 al COMMIT: un libro sin autores
...
ℹ tests 55
ℹ pass 55
ℹ fail 0
```

- `library.sql`: la biblioteca traducida.
- `extended.sql`: tres traducciones de la especialización y la agregación.
- `*-demo.sql`: cada restricción violada, con su SQLSTATE.
- `*.test.mjs`: cardinalidad y participación, por `err.code`.

# En resumen

- Entidades, atributos (compuestos, multivaluados, derivados) y relaciones (conjuntos de tuplas de entidades).
- Cardinalidad = máximo (flecha); participación = mínimo (línea doble). Restricciones, no estadísticas.
- Clave de una relación según la cardinalidad. Débil: clave del owner + discriminador.
- En el diagrama, una relación; nunca la clave del otro como atributo.
- Muchos a muchos → tabla; muchos a uno → FK en el lado "muchos" (`NOT NULL` si es total); uno a uno → FK `UNIQUE`.
- Total del lado "uno": constraint trigger diferido.
- Especialización: por nivel, solo subclases o una tabla; cada una pierde algo.
- Agregación: FK hacia la tabla de la relación agregada.

# Para la próxima clase

- Laboratorio: modelar un dominio con un diagrama E-R y traducirlo a `model.sql`.
- P1 (Esquema y consultas) se entrega esta semana; P2 (Diseño y aplicación) se publica.
- Leer Silberschatz, secs. 7.1 a 7.3 (dependencias funcionales y BCNF).
- Correr `make test` en `code/week-06` y leer los apuntes de la semana.

**Pregunta**

En la biblioteca, ¿por qué `loan` es una entidad y no una relación entre `member` y `copy`? ¿Qué restricción del requisito 5 perderían si la clave fuera `(member_id, isbn, copy_no)`?