

UNIVERSIDAD NACIONAL DE INGENIERÍA

Bases de Datos

Dependencias funcionales y normalización

Diseño de bases de datos

Gerson Garrido

Semana 7

Hoy

Sesión A: dependencias funcionales y BCNF

1. Anomalías de un mal diseño
2. Dependencias funcionales
3. Descomposición sin pérdida
4. BCNF y su algoritmo

Sesión B: teoría, 3NF

5. Axiomas de Armstrong y clausura
6. Cubierta canónica
7. Preservación de dependencias
8. 3NF y la síntesis; BCNF o 3NF
9. (Opcional) 4NF

Lectura

Silberschatz, secs. 7.1–7.3 (sesión A; complementaria 7.8) y 7.3–7.5 (sesión B; opcional 7.6). Berkeley CS186, Note 13 *DB Design*.

SECCIÓN 01

Anomalías

01

La hoja de viajes

Una oficina lleva los viajes en una hoja de cálculo; alguien la importa tal cual:

```
trip_sheet(trip_id, departure_at, route_id, origin_agency, origin_city,
dest_agency, dest_city, distance_km, base_fare, plate, kind, capacity_kg)
```

- Una fila por viaje; `trip_id` es la clave primaria.
- Cada fila copia los datos de la ruta, de las dos agencias y del vehículo.
- Se construye con un join de `trip`, `route`, `agency` (dos veces) y `vehicle`.

> **Pregunta**
>
> Sin joins, todo se lee en una fila. ¿Qué podría salir mal?

La misma información, muchas veces

```
psql

 trip_id | route_id | distance_km |  plate  | kind | capacity_kg
---------+----------+-------------+---------+------+-------------
      47 |      116 |          31 | K2U-314 | van  |        1400
     126 |      113 |          22 | K2U-314 | van  |        1400
     153 |      119 |          18 | K2U-314 | van  |        1400
     206 |      115 |          22 | K2U-314 | van  |        1400
(4 rows)
```

```
psql

 filas | rutas | vehiculos
-------+-------+-----------
  5914 |   120 |        24
(1 row)
```

- 24 vehículos y 120 rutas, pero sus datos aparecen 5914 veces.
- **Redundancia** *(redundancy)*: el mismo hecho en muchos lugares, y nada obliga a que digan lo mismo.

Anomalía de actualización

examples.sql

```sql
BEGIN;
UPDATE trip_sheet SET capacity_kg = 1500
WHERE trip_id = 47;
SELECT plate,
       array_agg(DISTINCT capacity_kg)
         AS capacidades
FROM trip_sheet
WHERE plate = 'K2U-314'
GROUP BY plate;
ROLLBACK;
```

psql

```
  plate   | capacidades
----------+-------------
 K2U-314 | {1400,1500}
(1 row)
```

Anomalía de actualización

examples.sql

```sql
BEGIN;
UPDATE trip_sheet SET capacity_kg = 1500
WHERE trip_id = 47;
SELECT plate,
       array_agg(DISTINCT capacity_kg)
         AS capacidades
FROM trip_sheet
WHERE plate = 'K2U-314'
GROUP BY plate;
ROLLBACK;
```

psql

```
  plate  | capacidades
---------+-------------
 K2U-314 | {1400,1500}
(1 row)
```

Anomalía

La van tiene 91 filas; se cambió una. ¿Cuánto carga `K2U-314`? Depende de qué fila se lea.

Anomalías de inserción y de borrado

Inserción: un vehículo sin viajes no tiene dónde ir.

psql

```
INSERT INTO trip_sheet (plate, kind, capacity_kg) VALUES ('Z9Z-999', 'van', 1200);
ERROR:  null value in column "trip_id" of relation "trip_sheet" violates not-null constraint
```

Anomalías de inserción y de borrado

Inserción: un vehículo sin viajes no tiene dónde ir.

psql

```
INSERT INTO trip_sheet (plate, kind, capacity_kg) VALUES ('Z9Z-999', 'van', 1200);
ERROR:  null value in column "trip_id" of relation "trip_sheet" violates not-null constraint
```

Borrado: al borrar los viajes de la ruta 113, se pierde la ruta.

examples.sql

```
1  BEGIN;
2  DELETE FROM trip_sheet WHERE route_id = 113;
3  SELECT count(DISTINCT route_id) AS rutas, ...
```

psql

```
DELETE 75
 rutas | viajes_113
-------+------------
   119 |          0
```

Una causa, una solución

Causa

La tabla guarda hechos sobre cosas distintas (viajes, rutas, agencias, vehículos), y cada hecho sobre un vehículo depende de que exista un viaje que lo lleve.

Una causa, una solución

Causa

La tabla guarda hechos sobre cosas distintas (viajes, rutas, agencias, vehículos), y cada hecho sobre un vehículo depende de que exista un viaje que lo lleve.

Descomponer *(decomposition)*: reemplazar R por $R_1, \ldots, R_n$ con $R = R_1 \cup \cdots \cup R_n$, y r por sus proyecciones $\Pi_{R_i}(r)$.

Una causa, una solución

Causa

La tabla guarda hechos sobre cosas distintas (viajes, rutas, agencias, vehículos), y cada hecho sobre un vehículo depende de que exista un viaje que lo lleve.

Descomponer *(decomposition)*: reemplazar R por $R_1, \ldots, R_n$ con $R = R_1 \cup \cdots \cup R_n$, y r por sus proyecciones $\Pi_{R_i}(r)$.

1. ¿Se recupera r desde las partes? **sin pérdida**
2. ¿Las partes ya no tienen redundancia? **BCNF, 3NF**
3. ¿Las reglas se verifican sin joins? **preservación de dependencias**

Las tres preguntas se responden con dependencias funcionales.

SECCIÓN 02

Dependencias funcionales

02

Dependencia funcional

Dependencia funcional *(functional dependency)*

$\alpha \to \beta$ se cumple en la instancia r si para todo par de tuplas $t_1, t_2 \in r$:

$$t_1[\alpha] = t_2[\alpha] \;\Rightarrow\; t_1[\beta] = t_2[\beta].$$

Vale en el esquema R si se cumple en toda instancia válida.

Dependencia funcional

Dependencia funcional *(functional dependency)*

$\alpha \to \beta$ se cumple en la instancia r si para todo par de tuplas $t_1, t_2 \in r$:

$$t_1[\alpha] = t_2[\alpha] \;\Rightarrow\; t_1[\beta] = t_2[\beta].$$

Vale en el esquema R si se cumple en toda instancia válida.

- "Conocer α basta para conocer β". α es el determinante.
- Valen: `plate` $\to$ `kind`, `capacity_kg`; en `customer`, `dni` $\to$ `first_name`, `last_name`, `city`.
- No valen: `origin_city` $\to$ `origin_agency` (Lima tiene tres agencias), `capacity_kg` $\to$ `plate`.

Las reglas de la hoja de viajes: F

Dependencia	Por qué
`trip_id` $\to$ `departure_at`, `route_id`, `plate`	una salida, una ruta, un vehículo
`route_id` $\to$ `origin_agency`, `dest_agency`, `distance_km`, `base_fare`	de una agencia a otra
`origin_agency`, `dest_agency` $\to$ `route_id`	una ruta por par (`UNIQUE`)
`origin_agency` $\to$ `origin_city`	cada agencia, en una ciudad
`dest_agency` $\to$ `dest_city`	lo mismo en el destino
`plate` $\to$ `kind`, `capacity_kg`	cada placa, un vehículo

De dónde salen

De los requisitos del negocio, no de los datos.

Triviales y superclaves

Trivial

$\alpha \to \beta$ es trivial si $\beta \subseteq \alpha$. Se cumple en cualquier relación.

- $\texttt{trip_id}, \texttt{plate} \to \texttt{plate}$ es trivial; $\texttt{trip_id}, \texttt{plate} \to \texttt{plate}, \texttt{kind}$ no.

Triviales y superclaves

Trivial

$\alpha \to \beta$ es trivial si $\beta \subseteq \alpha$. Se cumple en cualquier relación.

- $\texttt{trip_id}, \texttt{plate} \to \texttt{plate}$ es trivial; $\texttt{trip_id}, \texttt{plate} \to \texttt{plate}, \texttt{kind}$ no.

Superclave y clave candidata

K es superclave de *R* si $K \to R$. Es clave candidata si además ningún subconjunto propio de *K* es superclave.

- Si dos tuplas coinciden en *K*, coinciden en todo: son la misma.
- Una dependencia generaliza la clave: `plate` identifica el vehículo, no la fila.

Los datos refutan, no demuestran

examples.sql

```sql
SELECT distance_km
FROM trip_sheet
GROUP BY distance_km
HAVING count(DISTINCT base_fare) > 1;
```

psql

```
 distance_km
-------------
(0 rows)
```

Los datos refutan, no demuestran

examples.sql

```sql
SELECT distance_km
FROM trip_sheet
GROUP BY distance_km
HAVING count(DISTINCT base_fare) > 1;
```

psql

```
 distance_km
-------------
(0 rows)
```

- Ningún contraejemplo, pero `distance_km` $\rightarrow$ `base_fare` no es una regla: hoy la tarifa se calcula con la distancia; mañana hay una promoción.
- En `customer` ningún teléfono se repite; una familia puede compartir uno.

Error común

Deducir dependencias mirando unas filas. Con diez clientes, quizá `last_name` $\rightarrow$ `dni` “se cumple”.

Refutar una dependencia con SQL

$\alpha \to \beta$ falla si un valor de α tiene más de un β:

examples.sql

```sql
SELECT origin_city,
       count(DISTINCT origin_agency)
         AS agencias
FROM trip_sheet
GROUP BY origin_city
HAVING count(DISTINCT origin_agency) > 1
ORDER BY origin_city;
```

psql

```
 origin_city | agencias
-------------+----------
 Arequipa    |        2
 Lima        |        3
(2 rows)
```

- Cada fila es un contraejemplo.
- Con dos columnas a la derecha: `count(DISTINCT (kind, capacity_kg))`.
- Cero filas: los datos de hoy no la contradicen. Nada más.

Una regla que creíamos cierta

¿Un vehículo puede salir en dos viajes a la misma hora?
`plate`, `departure_at` $\rightarrow$ `trip_id`:

examples.sql

```sql
SELECT plate, departure_at,
       array_agg(trip_id ORDER BY trip_id)
         AS viajes
FROM trip_sheet
GROUP BY plate, departure_at
HAVING count(DISTINCT trip_id) > 1
ORDER BY departure_at, plate
LIMIT 3;
```

psql

```
  plate  |      departure_at      | viajes
---------+------------------------+---------
 P5Z-632 | 2026-01-05 06:00:00-05 | {1,2}
 R7D-433 | 2026-01-05 08:00:00-05 | {24,26}
 T8D-416 | 2026-01-05 10:00:00-05 | {37,39}
(3 rows)
```

- 336 pares (placa, hora) con más de un viaje.
- O la regla no existe, o los datos están mal. La base de datos no puede decidirlo; lo que no se hace es ponerla en *F* y seguir.

Ojo con NULL

examples.sql

```sql
-- devuelve 0 filas
SELECT plate
FROM (VALUES ('A1A-111', 'bus'),
             ('A1A-111', NULL))
       AS t (plate, kind)
GROUP BY plate
HAVING count(DISTINCT kind) > 1;

-- NULL como un valor más: A1A-111
HAVING count(DISTINCT kind)
       + CASE WHEN count(kind) < count(*)
              THEN 1 ELSE 0 END > 1;
```

En PostgreSQL

`count(DISTINCT y)` no cuenta los `NULL`: el grupo {`'bus'`, `NULL`} pasa la prueba.

- `GROUP BY` sí junta los `NULL` del lado izquierdo.
- La teoría del libro supone que no hay `NULL`.

SECCIÓN 03

Descomposición sin pérdida

03

Descomposición sin pérdida

Sin pérdida ***(lossless)***

R_1, R_2 es sin pérdida si para toda instancia válida r: $\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) = r$. Si no, es con pérdida *(lossy)*.

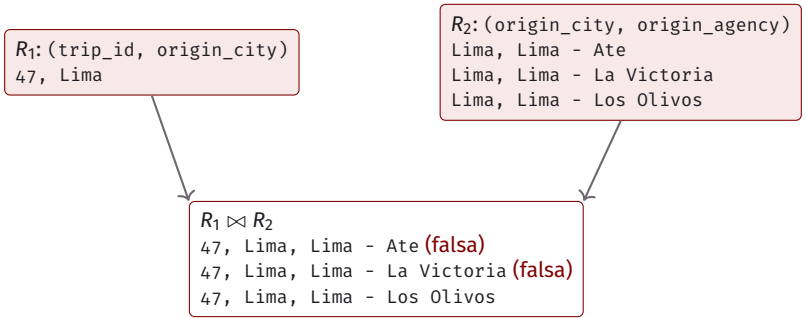
Descomposición sin pérdida

Sin pérdida *(lossless)*

R_1, R_2 es sin pérdida si para toda instancia válida r: $\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) = r$. Si no, es con pérdida *(lossy)*.

- Siempre vale $r \subseteq \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$: cada $t \in r$ deja $t[R_1]$ y $t[R_2]$, que el join vuelve a juntar.
- Una descomposición con pérdida no pierde filas: gana filas. Pierde *información*: ya no se sabe cuáles son las verdaderas.
- Las filas de más son tuplas falsas *(spurious tuples)*.

Partir por la ciudad



El viaje 47 salió de Lima - Los Olivos; en las partes ya no hay forma de saberlo.

En los datos: 5110 filas falsas

examples.sql

```sql
WITH r1 AS (
  SELECT DISTINCT trip_id, origin_city
  FROM trip_sheet
), r2 AS (
  SELECT DISTINCT origin_city, origin_agency
  FROM trip_sheet
), joined AS (
  SELECT trip_id, origin_agency, origin_city
  FROM r1 JOIN r2 USING (origin_city)
), original AS (...)  -- las tres columnas
SELECT ...,           -- los dos conteos
  (SELECT count(*) FROM (TABLE joined
     EXCEPT TABLE original) AS x) AS de_mas,
  (SELECT count(*) FROM (TABLE original
     EXCEPT TABLE joined) AS x) AS faltan;
```

psql

```
 original | reconstruida | de_mas | faltan
----------+--------------+--------+--------
     5914 |        11024 |   5110 |      0
(1 row)
```

- Lima: 2330 viajes, 3 agencias. Arequipa: 450 viajes, 2 agencias.
- $2330 \times 2 + 450 \times 1 = 5110$.
- `DISTINCT`: la proyección del álgebra es un conjunto.
- Partiendo por `origin_agency`: 5914, 0, 0.

El criterio con dependencias

Teorema (sec. 7.2.3)

R_1, R_2 es sin pérdida si $R_1 \cap R_2 \to R_1$ o $R_1 \cap R_2 \to R_2$ está en F^+: los atributos comunes son superclave de una de las partes.

El criterio con dependencias

Teorema (sec. 7.2.3)

R_1, R_2 es sin pérdida si $R_1 \cap R_2 \to R_1$ o $R_1 \cap R_2 \to R_2$ está en F^+: los atributos comunes son superclave de una de las partes.

- Por ciudad: $\{\texttt{origin_city}\}^+ = \{\texttt{origin_city}\}$. No determina ninguna parte: con pérdida.
- Por agencia: $\texttt{origin_agency} \to \texttt{origin_agency}, \texttt{origin_city} = R_2$: **sin pérdida**.

El criterio con dependencias

Teorema (sec. 7.2.3)

R_1, R_2 es sin pérdida si $R_1 \cap R_2 \to R_1$ o $R_1 \cap R_2 \to R_2$ está en F^+: los atributos comunes son superclave de una de las partes.

- Por ciudad: $\{\texttt{origin_city}\}^+ = \{\texttt{origin_city}\}$. No determina ninguna parte: con pérdida.
- Por agencia: $\texttt{origin_agency} \to \texttt{origin_agency}, \texttt{origin_city} = R_2$: **sin pérdida**.
- Es suficiente; si las únicas restricciones son dependencias funcionales, también necesaria.
- Vale para instancias válidas: si una agencia aparece con dos ciudades por un error, partir por agencia también inventa filas (lo verán en el laboratorio).

Por qué funciona

Supongamos $R_1 \cap R_2 \to R_2$. Falta ver que el join está contenido en r.

Por qué funciona

Supongamos $R_1 \cap R_2 \to R_2$. Falta ver que el join está contenido en r.

1. Sea t del join: viene de $t_1 \in \Pi_{R_1}(r)$ y $t_2 \in \Pi_{R_2}(r)$ que coinciden en $R_1 \cap R_2$.

Por qué funciona

Supongamos $R_1 \cap R_2 \to R_2$. Falta ver que el join está contenido en r.

1. Sea t del join: viene de $t_1 \in \Pi_{R_1}(r)$ y $t_2 \in \Pi_{R_2}(r)$ que coinciden en $R_1 \cap R_2$.
2. $t_1 = u_1[R_1]$ y $t_2 = u_2[R_2]$ para $u_1, u_2 \in r$.

Por qué funciona

Supongamos $R_1 \cap R_2 \to R_2$. Falta ver que el join está contenido en r.

1. Sea t del join: viene de $t_1 \in \Pi_{R_1}(r)$ y $t_2 \in \Pi_{R_2}(r)$ que coinciden en $R_1 \cap R_2$.
2. $t_1 = u_1[R_1]$ y $t_2 = u_2[R_2]$ para $u_1, u_2 \in r$.
3. u_1 y u_2 coinciden en $R_1 \cap R_2$; por la dependencia, coinciden en R_2: $u_1[R_2] = t_2$.

Por qué funciona

Supongamos $R_1 \cap R_2 \to R_2$. Falta ver que el join está contenido en r.

1. Sea t del join: viene de $t_1 \in \Pi_{R_1}(r)$ y $t_2 \in \Pi_{R_2}(r)$ que coinciden en $R_1 \cap R_2$.
2. $t_1 = u_1[R_1]$ y $t_2 = u_2[R_2]$ para $u_1, u_2 \in r$.
3. u_1 y u_2 coinciden en $R_1 \cap R_2$; por la dependencia, coinciden en R_2: $u_1[R_2] = t_2$.
4. Entonces u_1 tiene los valores de t_1 en R_1 y los de t_2 en R_2: $u_1 = t \in r$. ■

SECCIÓN 04

BCNF

04



Forma normal de Boyce-Codd

BCNF

R está en BCNF si para toda $\alpha \to \beta$ de F^+ con $\alpha, \beta \subseteq R$:

- $\alpha \to \beta$ es trivial, o
- α es superclave de *R*.

Forma normal de Boyce-Codd

BCNF

R está en BCNF si para toda $\alpha \to \beta$ de F^+ con $\alpha, \beta \subseteq R$:

- $\alpha \to \beta$ es trivial, o
- α es superclave de *R*.

Intuición

Toda flecha no trivial sale de una superclave. Si algo determina otra cosa sin ser clave, ese hecho se repite en cada fila: va en su propia tabla.

Forma normal de Boyce-Codd

BCNF

R está en BCNF si para toda $\alpha \to \beta$ de F^+ con $\alpha, \beta \subseteq R$:

- $\alpha \to \beta$ es trivial, o
- α es superclave de *R*.

Intuición

Toda flecha no trivial sale de una superclave. Si algo determina otra cosa sin ser clave, ese hecho se repite en cada fila: va en su propia tabla.

- `trip_sheet`: la única clave es `trip_id`. `route_id`, el par de agencias, cada agencia y `plate` determinan cosas sin ser superclave. No está en BCNF.

Probar BCNF: el atajo y la trampa

- **En el esquema original** R: basta revisar las dependencias de F.
- **En una parte** R_i de una descomposición: no basta. Hay que mirar F^+: para cada $\alpha \subseteq R_i$, $\alpha^+ \cap (R_i - \alpha)$ es vacío o es todo $R_i - \alpha$.

Probar BCNF: el atajo y la trampa

- **En el esquema original** R: basta revisar las dependencias de F.
- **En una parte** R_i de una descomposición: no basta. Hay que mirar F^+: para cada $\alpha \subseteq R_i$, $\alpha^+ \cap (R_i - \alpha)$ es vacío o es todo $R_i - \alpha$.

Ejemplo del libro (sec. 7.5.1)

$R(A, B, C, D, E)$, $F = \{A \to B,\ BC \to D\}$, partida en $R_1(A, B)$ y $R_2(A, C, D, E)$.

Probar BCNF: el atajo y la trampa

- **En el esquema original** R: basta revisar las dependencias de F.
- **En una parte** R_i de una descomposición: no basta. Hay que mirar F^+: para cada $\alpha \subseteq R_i$, $\alpha^+ \cap (R_i - \alpha)$ es vacío o es todo $R_i - \alpha$.

Ejemplo del libro (sec. 7.5.1)

$R(A, B, C, D, E)$, $F = \{A \to B,\ BC \to D\}$, partida en $R_1(A, B)$ y $R_2(A, C, D, E)$.

- Ninguna dependencia de F cabe en R_2. ¿Está en BCNF?
- No: $AC \to D$ está en F^+ (de $A \to B$, $AC \to BC$; con $BC \to D$), cabe en R_2 y AC no determina E.

El algoritmo de descomposición en BCNF

pseudocódigo

```
resultado := {R}
mientras haya un Ri en resultado que no esté en BCNF
  sea alfa -> beta una dependencia no trivial que se cumple en Ri,
    con alfa y beta disjuntos y alfa no superclave de Ri
  resultado := (resultado - {Ri}) U {alfa U beta, Ri - beta}
```

El algoritmo de descomposición en BCNF

pseudocódigo

```
resultado := {R}
mientras haya un Ri en resultado que no esté en BCNF
  sea alfa -> beta una dependencia no trivial que se cumple en Ri,
    con alfa y beta disjuntos y alfa no superclave de Ri
  resultado := (resultado - {Ri}) U {alfa U beta, Ri - beta}
```

- **Sin pérdida**: las partes se cortan en α, y $\alpha \to \alpha \cup \beta$.
- **Termina**: cada parte es más chica; dos atributos siempre están en BCNF.
- Conviene tomar $\beta = (\alpha^+ \cap R_i) - \alpha$: todo lo que α determina.

El algoritmo de descomposición en BCNF

pseudocódigo

```
resultado := {R}
mientras haya un Ri en resultado que no esté en BCNF
  sea alfa -> beta una dependencia no trivial que se cumple en Ri,
    con alfa y beta disjuntos y alfa no superclave de Ri
  resultado := (resultado - {Ri}) U {alfa U beta, Ri - beta}
```

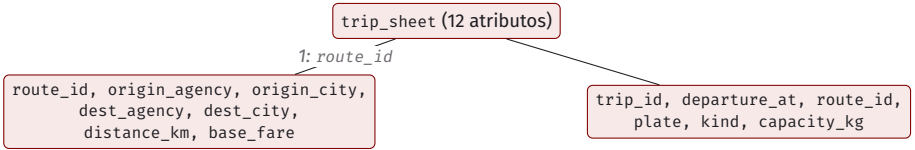
- **Sin pérdida**: las partes se cortan en α, y $\alpha \to \alpha \cup \beta$.
- **Termina**: cada parte es más chica; dos atributos siempre están en BCNF.
- Conviene tomar $\beta = (\alpha^+ \cap R_i) - \alpha$: todo lo que α determina.

Error común: creer que el resultado es único. Depende del orden de las violaciones.

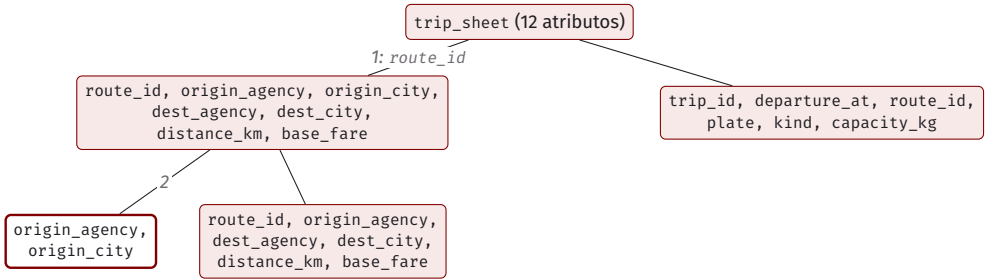
Ejemplo: la hoja de viajes

`trip_sheet` (12 atributos)

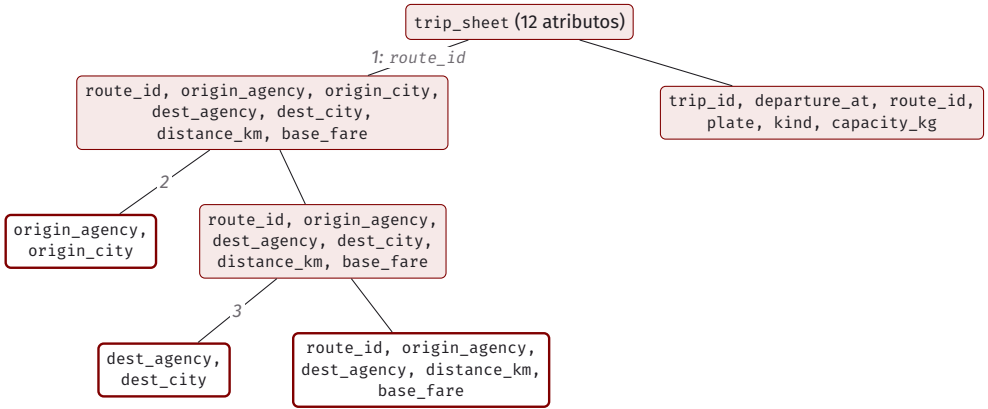
Ejemplo: la hoja de viajes



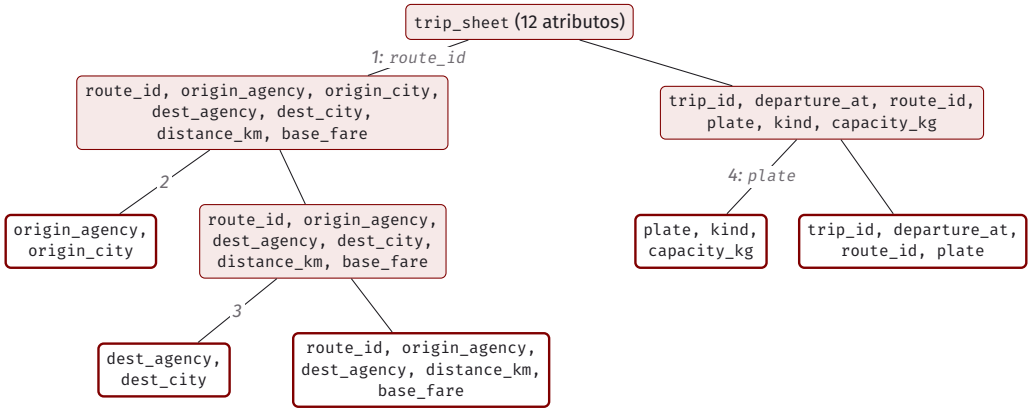
Ejemplo: la hoja de viajes



Ejemplo: la hoja de viajes



Ejemplo: la hoja de viajes



Cinco tablas en BCNF (borde grueso). Cada corte es sin pérdida.

El resultado y el esquema de Chasqui

Esquema	Clave(s)	En Chasqui
`(trip_id, departure_at, route_id, plate)`	`trip_id`	`trip`
`(route_id, origin_agency, dest_agency, distance_km, base_fare)`	`route_id`; el par	`route`
`(origin_agency, origin_city)`	`origin_agency`	`agency`
`(dest_agency, dest_city)`	`dest_agency`	`agency`
`(plate, kind, capacity_kg)`	`plate`	`vehicle`

- La tabla de rutas tiene **dos** claves candidatas y está en BCNF.
- Dos tablas de agencias: para el algoritmo son atributos distintos. Que son la misma entidad en dos papeles lo dice el modelo E-R, no las dependencias.
- `vehicle_id` y `agency_id` son claves sustitutas: otra decisión de diseño.

La descomposición en SQL

decompose.sql

```sql
CREATE TEMP TABLE s_vehicle AS
  SELECT DISTINCT plate, kind, capacity_kg
  FROM trip_sheet;
ALTER TABLE s_vehicle ADD PRIMARY KEY (plate);
-- ... lo mismo con las otras cuatro
WITH joined AS (
  SELECT ...  -- las doce columnas
  FROM s_trip
    JOIN s_route USING (route_id)
    JOIN s_origin USING (origin_agency)
    JOIN s_dest USING (dest_agency)
    JOIN s_vehicle USING (plate)
)
SELECT ... -- EXCEPT en los dos sentidos
```

psql

```
   tabla   | filas
-----------+-------
 s_origin  |    28
 s_dest    |    28
 s_route   |   120
 s_vehicle |    24
 s_trip    |  5914
(5 rows)

 reconstruida | de_mas | faltan
--------------+--------+--------
         5914 |      0 |      0
(1 row)
```

28, no 29: de Iquitos no sale ningún viaje.

Primera forma normal

1NF (sec. 7.8)

Un dominio es atómico si sus elementos son indivisibles. *R* está en 1NF si todos sus dominios son atómicos.

- No atómicos: `'Lima, Cusco'`, teléfonos separados por comas, un código `'LIM-2026-000123'` del que el programa saca la agencia. Sobre las partes no hay clave foránea ni `CHECK`.
- Atómico depende del uso: un DNI que nadie parte es atómico. La teoría de dependencias supone 1NF.

En PostgreSQL: `text[]` y `jsonb` no son atómicos. Si se va a buscar, unir o restringir por un elemento, va en una tabla aparte.

SECCIÓN 05

Axiomas de Armstrong

05

Clausura de F y axiomas de Armstrong

F^+

El conjunto de todas las dependencias que F implica lógicamente: las que se cumplen en toda instancia que cumple F.

Clausura de F y axiomas de Armstrong

F^+

El conjunto de todas las dependencias que F implica lógicamente: las que se cumplen en toda instancia que cumple F.

1. **Reflexividad**: si $\beta \subseteq \alpha$, entonces $\alpha \to \beta$.
2. **Aumento**: si $\alpha \to \beta$, entonces $\gamma\alpha \to \gamma\beta$.
3. **Transitividad**: si $\alpha \to \beta$ y $\beta \to \gamma$, entonces $\alpha \to \gamma$.

Clausura de F y axiomas de Armstrong

F^+

El conjunto de todas las dependencias que F implica lógicamente: las que se cumplen en toda instancia que cumple F.

1. **Reflexividad**: si $\beta \subseteq \alpha$, entonces $\alpha \to \beta$.
2. **Aumento**: si $\alpha \to \beta$, entonces $\gamma\alpha \to \gamma\beta$.
3. **Transitividad**: si $\alpha \to \beta$ y $\beta \to \gamma$, entonces $\alpha \to \gamma$.

- **Correctos**: no deducen nada falso. (Aumento: si coinciden en $\gamma\alpha$, coinciden en α, luego en β, y en γ.)
- **Completos**: deducen todo F^+.

Reglas derivadas

- **Unión**: $\alpha \to \beta$ y $\alpha \to \gamma \Rightarrow \alpha \to \beta\gamma$.
  Aumento: $\alpha \to \alpha\gamma$ y $\alpha\gamma \to \beta\gamma$. Transitividad.

Reglas derivadas

- **Unión**: $\alpha \to \beta$ y $\alpha \to \gamma \Rightarrow \alpha \to \beta\gamma$.
  Aumento: $\alpha \to \alpha\gamma$ y $\alpha\gamma \to \beta\gamma$. Transitividad.
- **Descomposición**: $\alpha \to \beta\gamma \Rightarrow \alpha \to \beta$ y $\alpha \to \gamma$.
  Reflexividad: $\beta\gamma \to \beta$. Transitividad.

Reglas derivadas

- **Unión**: $\alpha \to \beta$ y $\alpha \to \gamma \Rightarrow \alpha \to \beta\gamma$.
  Aumento: $\alpha \to \alpha\gamma$ y $\alpha\gamma \to \beta\gamma$. Transitividad.
- **Descomposición**: $\alpha \to \beta\gamma \Rightarrow \alpha \to \beta$ y $\alpha \to \gamma$.
  Reflexividad: $\beta\gamma \to \beta$. Transitividad.
- **Pseudotransitividad**: $\alpha \to \beta$ y $\gamma\beta \to \delta \Rightarrow \alpha\gamma \to \delta$.
  Aumento: $\alpha\gamma \to \gamma\beta$. Transitividad.

Reglas derivadas

- **Unión**: $\alpha \to \beta$ y $\alpha \to \gamma \Rightarrow \alpha \to \beta\gamma$.
  Aumento: $\alpha \to \alpha\gamma$ y $\alpha\gamma \to \beta\gamma$. Transitividad.
- **Descomposición**: $\alpha \to \beta\gamma \Rightarrow \alpha \to \beta$ y $\alpha \to \gamma$.
  Reflexividad: $\beta\gamma \to \beta$. Transitividad.
- **Pseudotransitividad**: $\alpha \to \beta$ y $\gamma\beta \to \delta \Rightarrow \alpha\gamma \to \delta$.
  Aumento: $\alpha\gamma \to \gamma\beta$. Transitividad.

Error común

El lado derecho se junta y se separa; el **izquierdo no**. De $AB \to C$ no sale $A \to C$: el par de agencias determina la ruta; una agencia sola, no.

SECCIÓN 06

Clausura de atributos

06

Clausura de un conjunto de atributos

α^+

Todos los atributos A con $\alpha \to A$ en F^+. Por la unión: $\alpha \to \beta \in F^+ \iff \beta \subseteq \alpha^+$.

pseudocódigo

```
resultado := alfa
repetir
  para cada beta -> gamma en F
    si beta está contenido en resultado:
      resultado := resultado U gamma
hasta que resultado no cambie
```

- Cada atributo que entra está justificado por los axiomas; termina porque solo crece.
- A lo más $|R|$ vueltas, cada una recorre F.

Ejemplo del libro (sec. 7.4.2)

$R = (A, B, C, G, H, I)$, $F = \{A \to B,\ A \to C,\ CG \to H,\ CG \to I,\ B \to H\}$

F

- A → B
- A → C
- CG → H
- CG → I
- B → H

RESULTADO

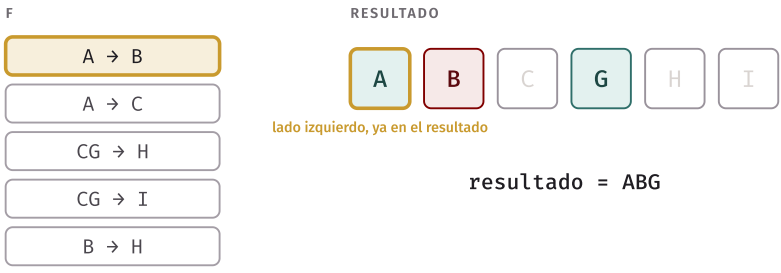
A	B	C	G	H	I

resultado = AG

inicio: resultado = α = AG

Ejemplo del libro (sec. 7.4.2)

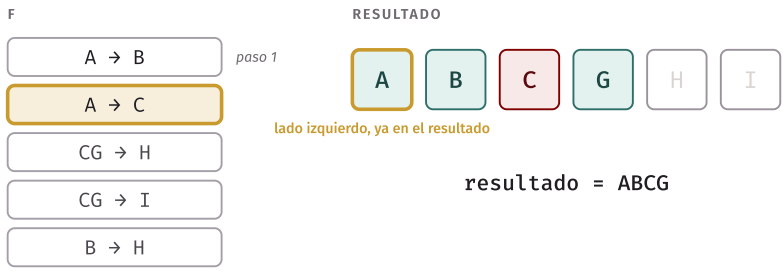
$R = (A, B, C, G, H, I)$, $F = \{A \to B,\ A \to C,\ CG \to H,\ CG \to I,\ B \to H\}$



paso 1: A está en el resultado, así que entra B

Ejemplo del libro (sec. 7.4.2)

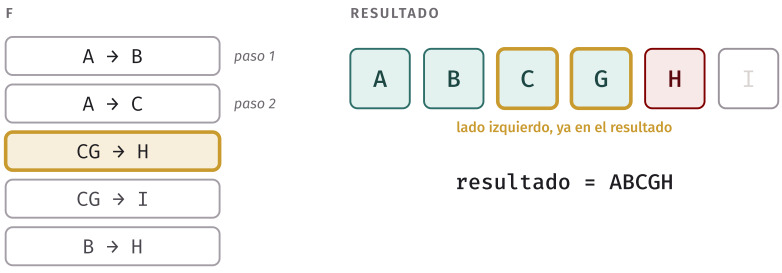
$R = (A, B, C, G, H, I)$, $F = \{A \to B,\ A \to C,\ CG \to H,\ CG \to I,\ B \to H\}$



paso 2: A está en el resultado, así que entra C

Ejemplo del libro (sec. 7.4.2)

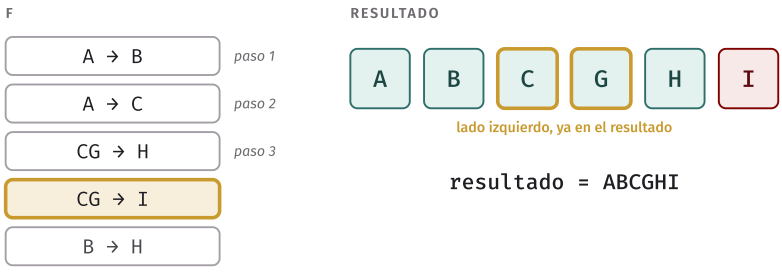
$R = (A, B, C, G, H, I)$, $F = \{A \to B,\ A \to C,\ CG \to H,\ CG \to I,\ B \to H\}$



paso 3: con C adentro, C y G están en el resultado, así que entra H

Ejemplo del libro (sec. 7.4.2)

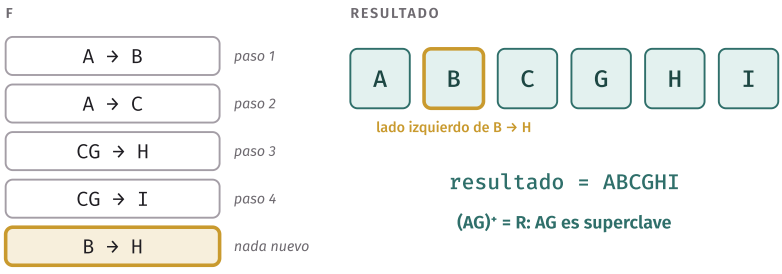
$R = (A, B, C, G, H, I)$, $F = \{A \to B,\ A \to C,\ CG \to H,\ CG \to I,\ B \to H\}$



paso 4: C y G están en el resultado, así que entra I

Ejemplo del libro (sec. 7.4.2)

$R = (A, B, C, G, H, I)$, $\quad F = \{A \to B,\ A \to C,\ CG \to H,\ CG \to I,\ B \to H\}$



- $(AG)^+ = R$: superclave. $A^+ = ABCH$, $G^+ = G$: ninguno basta.
- AG es clave candidata.

Para qué sirve la clausura

1. ¿α es superclave? $\alpha^+ \supseteq R$
2. ¿$\alpha \to \beta$ está en F^+? $\beta \subseteq \alpha^+$
3. Calcular F^+: para cada $\gamma \subseteq R$, $\gamma \to S$ para todo $S \subseteq \gamma^+$
4. Encontrar claves candidatas
5. Probar BCNF en una parte y probar “sin pérdida”

Para qué sirve la clausura

1. ¿α es superclave? $\alpha^+ \supseteq R$
2. ¿$\alpha \to \beta$ está en F^+? $\beta \subseteq \alpha^+$
3. Calcular F^+: para cada $\gamma \subseteq R$, $\gamma \to S$ para todo $S \subseteq \gamma^+$
4. Encontrar claves candidatas
5. Probar BCNF en una parte y probar “sin pérdida”

En la hoja de viajes:

- $\{\texttt{trip_id}\}^+$: las doce columnas. Clave candidata.
- $\{\texttt{plate}\}^+ = \{\texttt{plate}, \texttt{kind}, \texttt{capacity_kg}\}$.
- $\{\texttt{origin_agency}, \texttt{dest_agency}\}^+$: toda la ruta, con las ciudades; no el viaje.

Encontrar las claves candidatas

Probar subconjuntos de menor a mayor; dos atajos:

- Un atributo que **no aparece a la derecha** está en **toda** clave.
- Uno que aparece **solo a la derecha** no está en ninguna.

Encontrar las claves candidatas

Probar subconjuntos de menor a mayor; dos atajos:

- Un atributo que **no aparece a la derecha** está en **toda** clave.
- Uno que aparece **solo a la derecha** no está en ninguna.

`attention(customer_id, agency_id, executive_id)`

Un ejecutivo por cliente en cada agencia; cada ejecutivo, en una agencia.

$\texttt{customer_id}, \texttt{agency_id} \to \texttt{executive_id}, \quad \texttt{executive_id} \to \texttt{agency_id}$

Encontrar las claves candidatas

Probar subconjuntos de menor a mayor; dos atajos:

- Un atributo que **no aparece a la derecha** está en **toda** clave.
- Uno que aparece **solo a la derecha** no está en ninguna.

`attention(customer_id, agency_id, executive_id)`

Un ejecutivo por cliente en cada agencia; cada ejecutivo, en una agencia.

$\texttt{customer_id}, \texttt{agency_id} \to \texttt{executive_id}, \quad \texttt{executive_id} \to \texttt{agency_id}$

- `customer_id` no aparece a la derecha, pero solo no basta.
- $\{\texttt{customer_id}, \texttt{agency_id}\}^+$ y $\{\texttt{customer_id}, \texttt{executive_id}\}^+$: todo.
- Dos claves candidatas.

SECCIÓN 07

Cubierta canónica

07

Atributos extraños

Si alguien agrega a F: $\texttt{trip_id} \to \texttt{origin_city}$ y $\texttt{trip_id}, \texttt{plate} \to \texttt{capacity_kg}$, F^+ no cambia. Sobran.

Atributos extraños

Si alguien agrega a F: `trip_id` $\to$ `origin_city` y `trip_id`, `plate` $\to$ `capacity_kg`, F^+ no cambia. Sobran.

Atributo extraño *(extraneous)* en $\alpha \to \beta$

- $A \in \alpha$ es extraño si $\beta \subseteq (\alpha - A)^+$, calculada con F.
- $A \in \beta$ es extraño si $A \in \alpha^+$, calculada con $F' = (F - \{\alpha \to \beta\}) \cup \{\alpha \to (\beta - A)\}$.

Atributos extraños

Si alguien agrega a F: `trip_id` $\to$ `origin_city` y `trip_id, plate` $\to$ `capacity_kg`, F^+ no cambia. Sobran.

Atributo extraño *(extraneous)* en $\alpha \to \beta$

- $A \in \alpha$ es extraño si $\beta \subseteq (\alpha - A)^+$, calculada con F.
- $A \in \beta$ es extraño si $A \in \alpha^+$, calculada con $F' = (F - \{\alpha \to \beta\}) \cup \{\alpha \to (\beta - A)\}$.

Ojo

A la derecha se calcula **sin** el atributo que se prueba. Con F, todo parecería extraño.

Cubierta canónica

> *F_c (canonical cover)*
> Equivalente a F ($F_c^+ = F^+$), sin atributos extraños y sin dos dependencias con el mismo lado izquierdo.

pseudocódigo

```
Fc := F
repetir
  regla de unión: alfa -> b1 y alfa -> b2 se juntan en alfa -> b1 b2
  buscar una dependencia con un atributo extraño y quitárselo
hasta que Fc no cambie
```

- La unión se repite: quitar un atributo a la izquierda puede repetir un lado izquierdo.
- En la hoja de viajes, la cubierta canónica de F con las dos de más vuelve a las seis.

Ejemplo del libro (sec. 7.4.3)

$F = \{A \to BC,\ B \to C,\ A \to B,\ AB \to C\}$

Ejemplo del libro (sec. 7.4.3)

$F = \{A \to BC,\ B \to C,\ A \to B,\ AB \to C\}$

1. Unión: $\{A \to BC,\ B \to C,\ AB \to C\}$.

Ejemplo del libro (sec. 7.4.3)

$F = \{A \to BC,\ B \to C,\ A \to B,\ AB \to C\}$

1. Unión: $\{A \to BC,\ B \to C,\ AB \to C\}$.
2. ¿C extraño en $A \to BC$? Con $F' = \{A \to B, B \to C, AB \to C\}$, $A^+ = ABC \ni C$. Sí.

Ejemplo del libro (sec. 7.4.3)

$F = \{A \to BC,\ B \to C,\ A \to B,\ AB \to C\}$

1. Unión: $\{A \to BC,\ B \to C,\ AB \to C\}$.
2. ¿C extraño en $A \to BC$? Con $F' = \{A \to B, B \to C, AB \to C\}$, $A^+ = ABC \ni C$. Sí.
3. ¿A extraño en $AB \to C$? $B^+ = BC \supseteq C$. Sí; queda $B \to C$, que ya estaba.

Ejemplo del libro (sec. 7.4.3)

$F = \{A \to BC,\ B \to C,\ A \to B,\ AB \to C\}$

1. Unión: $\{A \to BC,\ B \to C,\ AB \to C\}$.
2. ¿C extraño en $A \to BC$? Con $F' = \{A \to B, B \to C, AB \to C\}$, $A^+ = ABC \ni C$. Sí.
3. ¿A extraño en $AB \to C$? $B^+ = BC \supseteq C$. Sí; queda $B \to C$, que ya estaba.
4. $F_c = \{A \to B,\ B \to C\}$.

Ejemplo del libro (sec. 7.4.3)

$F = \{A \to BC,\ B \to C,\ A \to B,\ AB \to C\}$

1. Unión: $\{A \to BC,\ B \to C,\ AB \to C\}$.
2. ¿C extraño en $A \to BC$? Con $F' = \{A \to B, B \to C, AB \to C\}$, $A^+ = ABC \ni C$. Sí.
3. ¿A extraño en $AB \to C$? $B^+ = BC \supseteq C$. Sí; queda $B \to C$, que ya estaba.
4. $F_c = \{A \to B,\ B \to C\}$.

No es única

$\{A \to BC, B \to AC, C \to AB\}$ tiene, entre otras, $\{A \to B, B \to C, C \to A\}$, $\{A \to C, C \to B, B \to A\}$ y $\{A \to C, B \to C, C \to AB\}$, según el orden.

SECCIÓN 08

Preservación de dependencias

08

Preservación de dependencias

Que preserva dependencias ***(dependency-preserving)***

F_i: las dependencias de F^+ con atributos solo en R_i. La descomposición preserva dependencias si $(F_1 \cup \cdots \cup F_n)^+ = F^+$.

Preservación de dependencias

Que preserva dependencias *(dependency-preserving)*

F_i: las dependencias de F^+ con atributos solo en R_i. La descomposición preserva dependencias si $(F_1 \cup \cdots \cup F_n)^+ = F^+$.

- Por qué importa: cada dependencia que cabe en una tabla se verifica con una clave; las otras, con joins en cada modificación.
- **Basta** que cada dependencia de F quepa en una tabla. No es necesario:

Preservación de dependencias

Que preserva dependencias *(dependency-preserving)*

F_i: las dependencias de F^+ con atributos solo en R_i. La descomposición preserva dependencias si $(F_1 \cup \cdots \cup F_n)^+ = F^+$.

- Por qué importa: cada dependencia que cabe en una tabla se verifica con una clave; las otras, con joins en cada modificación.
- **Basta** que cada dependencia de F quepa en una tabla. No es necesario:

Ejemplo

$F = \{A \to B, B \to C, A \to C\}$ en (A, B) y (B, C): $A \to C$ no cabe en ninguna, pero se deduce de las otras dos. Se preserva.

Probarla sin calcular F^+ (sec. 7.4.4)

Para cada $\alpha \to \beta$ de F:

pseudocódigo

```
resultado := alfa
repetir
  para cada Ri:  resultado := resultado U ((resultado inter Ri)+ inter Ri)
hasta que resultado no cambie
alfa -> beta se preserva si beta está contenido en resultado
```

- En cada tabla se deduce lo que se puede deducir mirando solo esa tabla.
- $A \to C$ en $(A, B), (B, C)$: $A \to AB \to ABC$. Se preserva.
- En $(A, B), (A, C)$ se pierde $B \to C$: desde B ninguna tabla agrega nada.
- La descomposición en BCNF de la hoja preserva las seis.

BCNF no siempre preserva dependencias

`attention(customer_id, agency_id, executive_id)`:

$\texttt{customer_id}, \texttt{agency_id} \to \texttt{executive_id}, \quad \texttt{executive_id} \to \texttt{agency_id}$

BCNF no siempre preserva dependencias

`attention(customer_id, agency_id, executive_id)`:
$\texttt{customer_id}, \texttt{agency_id} \to \texttt{executive_id}$, $\texttt{executive_id} \to \texttt{agency_id}$

- `executive_id` no es superclave: no está en BCNF.
- El algoritmo parte en `executive_agency(executive_id, agency_id)` y `customer_executive(customer_id, executive_id)`.
- Sin pérdida (`executive_id` es clave de la primera), las dos en BCNF.

BCNF no siempre preserva dependencias

`attention(customer_id, agency_id, executive_id)`:
$\texttt{customer_id}, \texttt{agency_id} \to \texttt{executive_id}$, $\texttt{executive_id} \to \texttt{agency_id}$

- `executive_id` no es superclave: no está en BCNF.
- El algoritmo parte en `executive_agency(executive_id, agency_id)` y `customer_executive(customer_id, executive_id)`.
- Sin pérdida (`executive_id` es clave de la primera), las dos en BCNF.

Pero

$\texttt{customer_id}, \texttt{agency_id} \to \texttt{executive_id}$ no cabe en ninguna y no se deduce de lo que queda. Ninguna descomposición en BCNF de `attention` la preserva.

La regla perdida, en SQL

attention.sql

```sql
-- 101 y 102 en la agencia 6; 103 en la 1
INSERT INTO executive_agency
VALUES (101, 6), (102, 6), (103, 1);
INSERT INTO customer_executive
VALUES (1, 101), (1, 102), (1, 103);

SELECT ce.customer_id, ea.agency_id,
       array_agg(ce.executive_id
                 ORDER BY ce.executive_id)
         AS ejecutivos
FROM customer_executive ce
  JOIN executive_agency ea
    USING (executive_id)
GROUP BY ce.customer_id, ea.agency_id
HAVING count(DISTINCT ce.executive_id) > 1;
```

psql

```
INSERT 0 3
INSERT 0 3
 customer_id | agency_id | ejecutivos
-------------+-----------+------------
           1 |         6 | {101,102}
(1 row)
```

- Cada tabla acepta sus filas.
- La violación solo se ve con un join.
- Para impedirla: un trigger en las dos tablas.

SECCIÓN 09

Tercera forma normal

09

Tercera forma normal

3NF

R está en 3NF si para toda $\alpha \to \beta$ de F^+ con $\alpha, \beta \subseteq R$:

- $\alpha \to \beta$ es trivial, o
- α es superclave de *R*, o
- cada atributo de $\beta - \alpha$ está en alguna clave candidata de *R*.

Tercera forma normal

3NF

R está en 3NF si para toda $\alpha \to \beta$ de F^+ con $\alpha, \beta \subseteq R$:

- $\alpha \to \beta$ es trivial, o
- α es superclave de *R*, o
- cada atributo de $\beta - \alpha$ está en alguna clave candidata de *R*.

- Las claves pueden ser distintas para cada atributo. BCNF $\Rightarrow$ 3NF.
- `attention` **está en 3NF**: `agency_id` es parte de la clave $\{$`customer_id`, `agency_id`$\}$.
- `trip_sheet` **no**: en `route_id` $\to$ `base_fare`, `base_fare` no está en ninguna clave. Es `trip_id` $\to$ `route_id` $\to$ `base_fare`.

`attention` en 3NF, con las dos reglas

attention.sql

```sql
CREATE TABLE executive (
  executive_id integer PRIMARY KEY,
  agency_id    integer NOT NULL
               REFERENCES agency,
  UNIQUE (executive_id, agency_id)
);
```

Una clave foránea solo apunta a una `PRIMARY KEY` o un `UNIQUE`.

attention.sql

```sql
CREATE TABLE attention (
  customer_id  integer REFERENCES customer,
  agency_id    integer,
  executive_id integer NOT NULL,
  CONSTRAINT attention_pk
    PRIMARY KEY (customer_id, agency_id),
  CONSTRAINT attention_executive_fk
    FOREIGN KEY (executive_id, agency_id)
    REFERENCES executive
      (executive_id, agency_id)
);
```

Las dos violaciones fallan

Con `(1, 6, 101)` y `(1, 1, 103)` ya guardados en `attention`:

psql

```
INSERT INTO attention VALUES (1, 6, 102);
ERROR:  duplicate key value violates unique constraint "attention_pk"
INSERT INTO attention VALUES (2, 1, 101);
ERROR:  insert or update on table "attention" violates foreign key constraint "attention_executive_fk"
```

- Otro ejecutivo para el cliente 1 en la agencia 6: lo impide la clave.
- El ejecutivo 101 atendiendo en la agencia 1: lo impide la clave foránea compuesta.
- No es BCNF: la agencia de cada ejecutivo se repite en cada fila de `attention`.

El algoritmo de síntesis en 3NF

pseudocódigo

```
Fc := una cubierta canónica de F
para cada alfa -> beta en Fc:  agregar el esquema alfa U beta
si ningún esquema contiene una clave candidata de R:
  agregar un esquema con una clave candidata
quitar los esquemas contenidos en otro
```

El algoritmo de síntesis en 3NF

pseudocódigo

```
Fc := una cubierta canónica de F
para cada alfa -> beta en Fc:  agregar el esquema alfa U beta
si ningún esquema contiene una clave candidata de R:
  agregar un esquema con una clave candidata
quitar los esquemas contenidos en otro
```

- **Preserva dependencias**: cada dependencia de F_c cabe en su esquema.
- **Sin pérdida**: gracias al esquema con la clave (sec. 7.5.3). No es opcional.
- **Cada esquema en 3NF**: usa que F_c no tiene atributos extraños.

El algoritmo de síntesis en 3NF

pseudocódigo

```
Fc := una cubierta canónica de F
para cada alfa -> beta en Fc:  agregar el esquema alfa U beta
si ningún esquema contiene una clave candidata de R:
  agregar un esquema con una clave candidata
quitar los esquemas contenidos en otro
```

- **Preserva dependencias**: cada dependencia de F_c cabe en su esquema.
- **Sin pérdida**: gracias al esquema con la clave (sec. 7.5.3). No es opcional.
- **Cada esquema en 3NF**: usa que F_c no tiene atributos extraños.

La clave: con $R(A, B, C)$ y $F = \{A \to B\}$ queda solo (A, B), sin la clave AC; se agrega (A, C). Sin él, C ni aparece.

Ejemplo del libro: `cust_banker_branch` (sec. 7.5.2)

$\texttt{customer_id}, \texttt{employee_id} \rightarrow \texttt{branch_name}, \texttt{type}, \quad \texttt{employee_id} \rightarrow \texttt{branch_name},$
$\texttt{customer_id}, \texttt{branch_name} \rightarrow \texttt{employee_id}$

Ejemplo del libro: `cust_banker_branch` (sec. 7.5.2)

$\texttt{customer_id}, \texttt{employee_id} \to \texttt{branch_name}, \texttt{type}$, $\texttt{employee_id} \to \texttt{branch_name}$, $\texttt{customer_id}, \texttt{branch_name} \to \texttt{employee_id}$

1. F_c: `branch_name` es extraño a la derecha de la primera (`employee_id` ya lo determina).

Ejemplo del libro: `cust_banker_branch` (sec. 7.5.2)

$\texttt{customer_id}, \texttt{employee_id} \to \texttt{branch_name}, \texttt{type}, \quad \texttt{employee_id} \to \texttt{branch_name},$
$\texttt{customer_id}, \texttt{branch_name} \to \texttt{employee_id}$

1. F_c: `branch_name` es extraño a la derecha de la primera (`employee_id` ya lo determina).
2. Esquemas: `(customer_id, employee_id, type)`, `(employee_id, branch_name)`, `(customer_id, branch_name, employee_id)`.

Ejemplo del libro: `cust_banker_branch` (sec. 7.5.2)

$\texttt{customer_id}, \texttt{employee_id} \to \texttt{branch_name}, \texttt{type}$, $\texttt{employee_id} \to \texttt{branch_name}$, $\texttt{customer_id}, \texttt{branch_name} \to \texttt{employee_id}$

1. F_c: `branch_name` es extraño a la derecha de la primera (`employee_id` ya lo determina).
2. Esquemas: `(customer_id, employee_id, type)`, `(employee_id, branch_name)`, `(customer_id, branch_name, employee_id)`.
3. La primera contiene la clave $\{\texttt{customer_id}, \texttt{employee_id}\}$.

Ejemplo del libro: `cust_banker_branch` (sec. 7.5.2)

$\texttt{customer_id}, \texttt{employee_id} \to \texttt{branch_name}, \texttt{type}$, $\texttt{employee_id} \to \texttt{branch_name}$, $\texttt{customer_id}, \texttt{branch_name} \to \texttt{employee_id}$

1. F_c: `branch_name` es extraño a la derecha de la primera (`employee_id` ya lo determina).
2. Esquemas: `(customer_id, employee_id, type)`, `(employee_id, branch_name)`, `(customer_id, branch_name, employee_id)`.
3. La primera contiene la clave $\{\texttt{customer_id}, \texttt{employee_id}\}$.
4. `(employee_id, branch_name)` está dentro de la tercera: se quita.

Ejemplo del libro: `cust_banker_branch` (sec. 7.5.2)

$\texttt{customer_id}, \texttt{employee_id} \to \texttt{branch_name}, \texttt{type}$, $\texttt{employee_id} \to \texttt{branch_name}$, $\texttt{customer_id}, \texttt{branch_name} \to \texttt{employee_id}$

1. F_c: `branch_name` es extraño a la derecha de la primera (`employee_id` ya lo determina).
2. Esquemas: `(customer_id, employee_id, type)`, `(employee_id, branch_name)`, `(customer_id, branch_name, employee_id)`.
3. La primera contiene la clave $\{\texttt{customer_id}, \texttt{employee_id}\}$.
4. `(employee_id, branch_name)` está dentro de la tercera: se quita.

Resultado

`(customer_id, employee_id, type)` y `(customer_id, branch_name, employee_id)`. La segunda está en 3NF pero no en BCNF: es `attention`.

Ejemplo del libro: `cust_banker_branch` (sec. 7.5.2)

$\texttt{customer_id}, \texttt{employee_id} \to \texttt{branch_name}, \texttt{type}$, $\texttt{employee_id} \to \texttt{branch_name}$, $\texttt{customer_id}, \texttt{branch_name} \to \texttt{employee_id}$

1. F_c: `branch_name` es extraño a la derecha de la primera (`employee_id` ya lo determina).
2. Esquemas: `(customer_id, employee_id, type)`, `(employee_id, branch_name)`, `(customer_id, branch_name, employee_id)`.
3. La primera contiene la clave $\{\texttt{customer_id}, \texttt{employee_id}\}$.
4. `(employee_id, branch_name)` está dentro de la tercera: se quita.

Resultado

`(customer_id, employee_id, type)` y `(customer_id, branch_name, employee_id)`. La segunda está en 3NF pero no en BCNF: es `attention`.

La hoja de viajes en 3NF da las mismas cinco tablas que BCNF.

BCNF o 3NF

	BCNF	3NF
Sin pérdida	sí	sí
Preserva dependencias	a veces	sí
Sin redundancia por d. f.	sí	a veces

- **Primero BCNF**; casi siempre preserva todo.
- **3NF** si BCNF pierde una dependencia importante.
- O BCNF y la perdida con un **trigger** que haga el join.

En la práctica

SQL hace cumplir barato las claves y las claves foráneas, nada más. `CREATE ASSERTION` está en el estándar; PostgreSQL no la implementa. 3NF deja cada dependencia en una tabla; si su lado izquierdo no es superclave, igual hace falta un trigger o, como en `attention`, otra tabla y una clave foránea.

SECCIÓN 10

4NF (opcional)

10



Una redundancia que BCNF no ve

examples.sql

```sql
WITH agency_phone (agency_id, phone) AS (
  VALUES (6, '984111222'), (6, '984333444')
), agency_service (agency_id, service) AS (
  VALUES (6, 'embalaje'), (6, 'recojo'),
         (6, 'seguro')
)
SELECT agency_id, phone, service
FROM agency_phone
  JOIN agency_service USING (agency_id)
ORDER BY phone, service;
```

psql

```
 agency_id |   phone   | service
-----------+-----------+----------
         6 | 984111222 | embalaje
         6 | 984111222 | recojo
         6 | 984111222 | seguro
         6 | 984333444 | embalaje
         6 | 984333444 | recojo
         6 | 984333444 | seguro
(6 rows)
```

`agency_info(agency_id, phone, service)`: teléfonos y servicios independientes. Sin dependencias funcionales no triviales, en BCNF, y cada teléfono se repite con cada servicio.

Dependencias multivaluadas y 4NF

$\alpha \twoheadrightarrow \beta$ (sec. 7.6)

Si $t_1[\alpha] = t_2[\alpha]$, existen t_3, t_4 con el mismo α tales que t_3 tiene el β de t_1 y el resto de t_2, y t_4 el β de t_2 y el resto de t_1.

- Para cada α, los β y el resto se combinan todos con todos: `agency_id` $\twoheadrightarrow$ `phone`.
- Toda dependencia funcional es multivaluada.

Dependencias multivaluadas y 4NF

$\alpha \twoheadrightarrow \beta$ (sec. 7.6)

Si $t_1[\alpha] = t_2[\alpha]$, existen t_3, t_4 con el mismo α tales que t_3 tiene el β de t_1 y el resto de t_2, y t_4 el β de t_2 y el resto de t_1.

- Para cada α, los β y el resto se combinan todos con todos: `agency_id` $\twoheadrightarrow$ `phone`.
- Toda dependencia funcional es multivaluada.

4NF

Para toda $\alpha \twoheadrightarrow \beta$ de D^+: es trivial ($\beta \subseteq \alpha$ o $\alpha \cup \beta = R$) o α es superclave.

- Se separa `(agency_id, phone)` y `(agency_id, service)`: dos atributos multivaluados del modelo E-R. 4NF $\Rightarrow$ BCNF.

SECCIÓN 11

El código de la semana

11



`fd.mjs` y `make test`

fd.mjs

```
1  import { fds, closure, candidateKeys,
2           bcnfDecompose } from './fd.mjs';
3
4  const F = fds('A -> B; A -> C; CG -> H;'
5              + 'CG -> I; B -> H');
6  closure('AG', F);
7  // [ 'A', 'B', 'C', 'G', 'H', 'I' ]
8  candidateKeys('ABCGHI', F);
9  // [ [ 'A', 'G' ] ]
10 bcnfDecompose('ABCGHI', F).schemas;
```

terminal

```
$ make test
...
✓ partir por origin_agency es sin pérdida
...
i tests 35
i pass 35
i fail 0
```

Cada ejemplo de los apuntes es una prueba.

Ojo: la parte difícil es escribir *F* desde los requisitos. Eso no lo hace ninguna función.

En resumen

- Redundancia $\Rightarrow$ anomalías de actualización, inserción y borrado.
- $\alpha \to \beta$: coinciden en $\alpha \Rightarrow$ coinciden en β, en toda instancia válida. Los datos solo la refutan (`GROUP BY ... HAVING count(DISTINCT ...) > 1`).
- Sin pérdida si $R_1 \cap R_2$ es superclave de una parte. Con pérdida = tuplas falsas de más.
- BCNF: todo determinante no trivial es superclave. El algoritmo parte en $\alpha\beta$ y $R_i - \beta$.
- Armstrong: reflexividad, aumento, transitividad. α^+ responde casi todo.
- F_c: sin atributos extraños; no es única.
- Preservar: $(F_1 \cup \cdots \cup F_n)^+ = F^+$. BCNF no siempre lo logra.
- 3NF: también vale si $\beta - \alpha$ está en claves candidatas. La síntesis siempre es sin pérdida y preserva.
- Se prefiere BCNF; 3NF si BCNF pierde una dependencia importante.

Para la próxima clase

- Laboratorio: normalizar la planilla de envíos, con `GROUP BY ... HAVING` y `EXCEPT`.
- P2 sigue en curso: el esquema normalizado y justificado con dependencias se entrega la semana 9.
- Leer Silberschatz, sec. 8.1 (la parte de JSON) y repasar para el examen parcial.
- Correr `make test` en `code/week-07` y leer los apuntes de la semana.

Pregunta

`attention` no está en BCNF. ¿Por qué ninguna descomposición en BCNF puede preservar $\texttt{customer_id}, \texttt{agency_id} \to \texttt{executive_id}$?