

UNIVERSIDAD NACIONAL DE INGENIERÍA

Bases de Datos

Álgebra relacional y SQL básico

Modelo relacional y SQL

Gerson Garrido

Semana 2

Hoy

1. Lenguajes de consulta: procedimentales y declarativos
2. Álgebra relacional: σ, Π, $\times$, $\bowtie$, $\cup$, $-$, $\cap$, ρ, $\leftarrow$
3. Consultas equivalentes: la idea detrás del optimizador
4. SQL: definición de datos con `CREATE TABLE`
5. `SELECT-FROM-WHERE`, conjuntos y multiconjuntos
6. Del álgebra a SQL

Lectura

Silberschatz, secs. 2.5–2.6 y 3.1–3.4. Berkeley CS186, *Note 6: Relational Algebra* (hasta *Rename*) y *Note 1: SQL I* (hasta *Boolean Operators*, más *ORDER BY* y *LIMIT*). PostgreSQL, *Table Basics* y *Select Lists*.

SECCIÓN 01

Lenguajes de consulta

01



Dos maneras de pedir datos

Procedimental

Se indica la secuencia de operaciones: “toma `agency`, quédate con las de Puno, y de esas, solo el nombre”.

Ejemplo: el **álgebra relacional**.

Declarativo

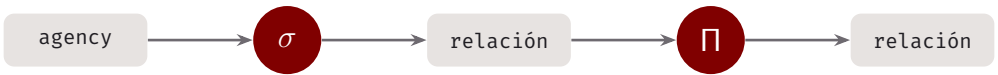
Se describe qué se quiere; el sistema decide el cómo: “los nombres de las agencias de Puno”.

Ejemplos: el cálculo relacional, **SQL**.

¿Por qué estudiar el álgebra?

Es el idioma interno del motor: PostgreSQL traduce el SQL a operaciones del álgebra y las reordena para encontrar un plan barato (semanas 11 y 12).

Clausura: de relaciones a relaciones



Cada operación recibe una o dos relaciones y devuelve una relación.

Por eso se pueden componer: $\Pi_{name}(\sigma_{region='\texttt{Puno}'}(\mathrm{agency}))$.

La notación

Operación	Símbolo	Recibe	Hace
Selección *(selection)*	σ_P	1 relación	filtra tuplas
Proyección *(projection)*	$\Pi_{A,B}$	1 relación	se queda con atributos
Renombramiento *(rename)*	ρ_X	1 relación	le pone nombre
Producto cartesiano	$\times$	2 relaciones	todas las parejas
Unión, diferencia	$\cup$, $-$	2 compatibles	de conjuntos
Intersección	$\cap$	2 compatibles	derivada
Join θ, natural join	$\bowtie_\theta$, $\bowtie$	2 relaciones	derivadas
Asignación	$\leftarrow$	—	nombra un paso

SECCIÓN 02

Selección y proyección

02

Selección σ

Selección

$\sigma_P(r) = \{\, t \mid t \in r \wedge P(t) \,\}$

- Mismo esquema que r; $|\sigma_P(r)| \leq |r|$.
- P: comparaciones ($=, \neq, <, \leq, >, \geq$) unidas con $\wedge$, $\vee$, $\neg$.
- En SQL: `WHERE`.

$\sigma_{kind='bus' \wedge capacity_kg>5000}(\text{vehicle})$

algebra.sql, A1

```
SELECT *
FROM vehicle
WHERE kind = 'bus' AND capacity_kg > 5000
ORDER BY vehicle_id;
 vehicle_id |  plate  | kind | capacity_kg
------------+---------+------+-------------
          4 | R5D-706 | bus  |        5500
          7 | L8P-227 | bus  |        5100
          8 | F4K-555 | bus  |        5900
(3 rows)
```

Dos propiedades de σ

- **Cascada**: una conjunción se parte en dos selecciones.

$$\sigma_{P_1 \wedge P_2}(r) = \sigma_{P_1}(\sigma_{P_2}(r))$$

- **Conmutatividad**: el orden de dos selecciones no importa.

$$\sigma_{P_1}(\sigma_{P_2}(r)) = \sigma_{P_2}(\sigma_{P_1}(r))$$

Comprobado

Una prueba de `algebra.test.mjs` corre las dos formas, $\sigma_{kind=\texttt{'bus'} \wedge capacity_kg > 5000}$ y la cascada, sobre `vehicle` y compara los resultados.

Proyección Π

Proyección

$\Pi_{A_1,\ldots,A_k}(r) = \{\, t[A_1,\ldots,A_k] \mid t \in r \,\}$

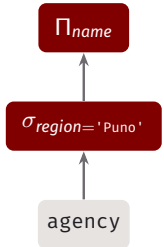
- El resultado es un conjunto: los duplicados desaparecen.
- $|\Pi_A(r)| \leq |r|$, con igualdad si A contiene una superclave.
- Puede llevar expresiones: $\Pi_{price \cdot 1.18}(\text{shipment})$.

$\Pi_{city}(\text{customer})$

	tuplas
`customer`	1800
$\Pi_{city}(\text{customer})$	26

En SQL hay que pedirlo: `SELECT DISTINCT city`.

Composición: se lee de adentro hacia afuera



2. proyecta

1. filtra

algebra.sql, A3

```
SELECT name
FROM agency
WHERE region = 'Puno'
ORDER BY name;
       name
------------------
 Juliaca - Centro
 Puno - Centro
(2 rows)
```

Error común: proyectar demasiado pronto

$\sigma_{city=\texttt{'Cusco'}}(\Pi_{name}(\text{agency}))$ no tiene sentido: después de Π_{name} ya no existe `city`.

errors.sql, E1

```
SELECT *
FROM (SELECT name FROM agency) AS t
WHERE city = 'Cusco';
ERROR:  column "city" does not exist
LINE 3: WHERE city = 'Cusco';
              ^
```

Proyectar antes sí vale si se conserva lo que σ necesita

$$\Pi_{name}(\sigma_{region=\texttt{'Puno'}}(\Pi_{name,\,region}(\text{agency}))) = \Pi_{name}(\sigma_{region=\texttt{'Puno'}}(\text{agency}))$$

SECCIÓN 03

Producto cartesiano y join

03

Producto cartesiano $\times$

Producto cartesiano

$r \times s = \{ t\,q \mid t \in r \wedge q \in s \}$

- Esquema: los dos esquemas concatenados; los nombres repetidos se califican (`route.origin_agency_id`).
- $|r \times s| = |r| \cdot |s|$.

algebra.sql, A4

```
 agencias | vehiculos | producto
----------+-----------+----------
       29 |        24 |      696
(1 row)
```

Un producto pequeño: casi nunca es lo que se quiere

$\Pi_{name,\ plate}(\sigma_{region=\texttt{'Puno'}}(\text{agency}) \times \sigma_{kind=\texttt{'van'}}(\text{vehicle}))$

- 2 agencias $\times$ 6 vans $=$ 12 tuplas.
- Ninguna dice nada útil: la van `C4J-431` no tiene que ver con Juliaca.
- `shipment` $\times$ `customer` empareja cada envío con todos los clientes, no con su remitente.

algebra.sql, A4b

```
       name       |  plate
------------------+---------
 Juliaca - Centro | C4J-431
 Juliaca - Centro | F6J-635
 Juliaca - Centro | K2U-314
 Juliaca - Centro | K8J-523
 Juliaca - Centro | T5E-356
 Juliaca - Centro | W6X-229
 Puno - Centro    | C4J-431
 Puno - Centro    | F6J-635
 Puno - Centro    | K2U-314
 Puno - Centro    | K8J-523
 Puno - Centro    | T5E-356
 Puno - Centro    | W6X-229
(12 rows)
```

Join θ: producto seguido de selección

Join θ *(theta join)*

$$r \bowtie_\theta s = \sigma_\theta(r \times s)$$

Si θ es una conjunción de igualdades, es un **equi-join**.

Los envíos con su remitente:

$$\text{shipment} \bowtie_{\textit{sender_id}=\textit{customer_id}} \text{customer}$$

$\times$: **21 600 000** tuplas

$\bowtie_\theta$: **12 000** tuplas

El join θ en SQL

algebra.sql, A5

```
SELECT s.shipment_id, c.first_name, c.last_name, s.weight_kg
FROM shipment AS s, customer AS c
WHERE s.sender_id = c.customer_id
  AND s.weight_kg > 140
ORDER BY s.weight_kg DESC, s.shipment_id
LIMIT 5;
 shipment_id | first_name |     last_name      | weight_kg
-------------+------------+--------------------+-----------
       10499 | Luz        | Huamán Sánchez     |    148.30
        9365 | Ximena     | Chávez Gutiérrez   |    146.90
        3690 | Luz        | Yupanqui Sánchez   |    146.60
       10818 | Piero      | Villanueva Ramírez |    146.50
        5574 | Luz        | Cáceres Vásquez    |    146.40
(5 rows)
```

La condición de join y la selección van juntas en `WHERE`. La forma `JOIN ... ON` es de la semana 4.

¿Por qué exactamente 12 000?

En general, $0 \leq |r \bowtie_\theta s| \leq |r| \cdot |s|$. Aquí:

- `sender_id` es clave foránea y `NOT NULL`: cada envío encuentra al menos un cliente;
- `customer_id` es clave primaria: encuentra a lo más uno.

Sin NOT NULL el argumento se cae

`trip_id` es clave foránea pero admite `NULL`: los envíos sin viaje no encuentran pareja.

algebra.sql, A5c

```
 envios | con_viaje | sin_viaje
--------+-----------+-----------
  12000 |     11500 |       500
(1 row)
```

Natural join $\bowtie$

Natural join

Si $R \cap S = \{A_1, \ldots, A_n\}$: $\quad r \bowtie s = \Pi_{R \cup S}(\sigma_{r.A_1 = s.A_1 \wedge \cdots \wedge r.A_n = s.A_n}(r \times s))$

$\mathrm{trip} \bowtie \mathrm{vehicle}$ iguala `vehicle_id`, el único nombre común.

Tiene $|\mathrm{trip}| = 5914$ tuplas, por la misma razón que antes.

algebra.sql, A6

```
 trip_id | vehicle_id |  plate  | kind
---------+------------+---------+-------
       1 |         11 | P5Z-632 | truck
       2 |         11 | P5Z-632 | truck
       3 |          3 | F3E-445 | bus
(3 rows)
```

El natural join, paso a paso

trip

trip_id	vehicle_id	…
1	11	…
2	11	…
3	3	…

vehicle

vehicle_id	plate	kind	…
3	F3E-445	bus	…
11	P5Z-632	truck	…

fragmento: tres viajes, dos vehículos; … son los demás atributos

trip ⋈ vehicle

trip_id	vehicle_id	…	plate	kind	…
1	11	…	P5Z-632	truck	…
2	11	…	P5Z-632	truck	…
3	3	…	F3E-445	bus	…

natural join: el atributo común vehicle_id queda una sola vez

Error común: confiar en los nombres

El natural join depende de los nombres de los atributos, no de su significado.

- $\text{customer} \bowtie \text{agency}$ iguala `city`: cada cliente con cada agencia de su ciudad, **3433** tuplas que nadie pidió.
- $\text{shipment} \bowtie \text{route}$ iguala `origin_agency_id` y `dest_agency_id`.
- $\text{route} \bowtie \text{agency}$: no hay nombres comunes, así que es el producto cartesiano.

algebra.sql, A6c

```
 natural_join | producto
--------------+----------
         3480 |     3480
(1 row)
```

Con `route` y `agency` escribimos siempre el join θ: $\text{route} \bowtie_{\textit{origin_agency_id}=\textit{agency_id}} \text{agency}$. Outer joins y `NATURAL JOIN` en SQL: semana 4.

SECCIÓN 04

Operaciones de conjuntos

04

Unión, diferencia e intersección

Relaciones compatibles

r y s tienen la misma aridad y, para cada i, el i-ésimo atributo de las dos tiene un dominio compatible.

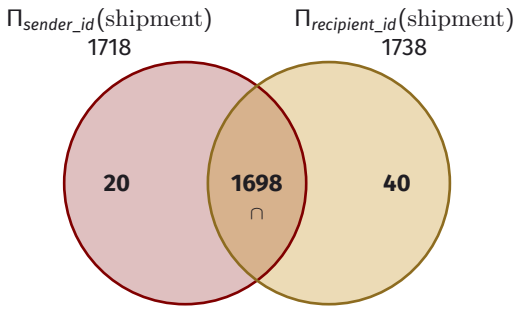
- $r \cup s = \{ t \mid t \in r \vee t \in s \}$
- $r - s = \{ t \mid t \in r \wedge t \notin s \}$
- $r \cap s = \{ t \mid t \in r \wedge t \in s \}$

El resultado toma los nombres de r. Casi siempre los argumentos son proyecciones.

Compatible no es lo mismo que razonable

$\Pi_{agency_id}(\text{agency}) \cup \Pi_{vehicle_id}(\text{vehicle})$ es una unión válida de enteros… y no significa nada.

Remitentes y destinatarios



$$|r \cup s| = |r| + |s| - |r \cap s| = 1718 + 1738 - 1698 = \mathbf{1758}$$

La diferencia no es conmutativa: $r - s$ tiene 20 tuplas, $s - r$ tiene 40.

Diferencia: “los que tienen esto pero no aquello”

Agencias desde las que no sale ninguna ruta:

$$\Pi_{agency_id}(\text{agency}) - \Pi_{origin_agency_id}(\text{route})$$

algebra.sql, A8

```
SELECT agency_id FROM agency
EXCEPT
SELECT origin_agency_id FROM route
ORDER BY agency_id;
 agency_id
-----------
        29
(1 row)
```

Iquitos: no se llega por carretera.

Origen de alguna ruta y nunca destino:

$$\Pi_{origin_agency_id}(\text{route}) - \Pi_{dest_agency_id}(\text{route})$$

algebra.sql, A8b

```
 origin_agency_id
------------------
(0 rows)
```

Vacío también es una respuesta

Cada ruta tiene su vuelta. La relación vacía es un resultado válido.

Relaciones no compatibles

$\Pi_{agency_id}(\text{agency}) \cup \Pi_{name}(\text{agency})$ mezcla enteros y textos:

errors.sql, E2

```
SELECT agency_id FROM agency
UNION
SELECT name FROM agency;
ERROR:  UNION types integer and text cannot be matched
LINE 3: SELECT name FROM agency;
               ^
```

Y una combinación: los clientes que no aparecen en ningún envío son **42**.

$$\Pi_{customer_id}(\text{customer}) - (\Pi_{sender_id}(\text{shipment}) \cup \Pi_{recipient_id}(\text{shipment}))$$

SECCIÓN 05

Renombramiento y asignación

05

Renombramiento ρ

Renombramiento

$\rho_x(E)$ le pone el nombre x al resultado de E; $\rho_{x(A_1,\ldots,A_n)}(E)$ renombra además sus atributos.

Sirve para comparar dos tuplas de la misma relación. Rutas con ruta de vuelta:

$$\rho_{r1}(\text{route}) \bowtie_{\substack{r1.origin_agency_id=r2.dest_agency_id \\ \wedge\ r1.dest_agency_id=r2.origin_agency_id}} \rho_{r2}(\text{route})$$

Sin ρ es otra pregunta

$\sigma_{origin_agency_id=dest_agency_id}(\text{route})$ compara dos atributos de la *misma* tupla. Es vacía: un CHECK lo prohíbe.

En SQL, ρ es el alias de tabla

algebra.sql, A10

```sql
SELECT count(*) AS pares_ordenados
FROM route AS r1, route AS r2
WHERE r1.origin_agency_id = r2.dest_agency_id
  AND r1.dest_agency_id = r2.origin_agency_id;
```

psql

```
 pares_ordenados
-----------------
             120
(1 row)
```

- `route AS r1` es $\rho_{r1}(\text{route})$.
- Salen las **120** rutas: todas tienen su vuelta.

Cada par una sola vez

Cada par sale dos veces: una con la ida como `r1`, otra con la vuelta. Con `r1.route_id < r2.route_id` quedan 60 pares.

algebra.sql, A10b

```
 ida | vuelta | origin_agency_id | dest_agency_id
-----+--------+------------------+----------------
   1 |      2 |                1 |              7
   3 |      4 |                1 |             11
   5 |      6 |                2 |             11
   7 |      8 |                1 |             12
   9 |     10 |                2 |             12
(5 rows)
```

Las cinco primeras (`LIMIT 5`).

Asignación $\leftarrow$: una consulta larga, en pasos

Rutas que unen dos agencias de la misma región:

$$
\begin{aligned}
o &\leftarrow \rho_{o}(\text{agency}) \\
d &\leftarrow \rho_{d}(\text{agency}) \\
ro &\leftarrow \text{route} \bowtie_{\text{route.origin_agency_id}=o.\text{agency_id}} o \\
\textit{resultado} &\leftarrow \Pi_{\textit{route_id},\, o.\text{name},\, d.\text{name}}(\sigma_{o.\text{region}=d.\text{region}}(ro \bowtie_{\text{route.dest_agency_id}=d.\text{agency_id}} d))
\end{aligned}
$$

- La asignación no modifica la base de datos: nombra un resultado intermedio.
- No agrega poder: cada variable se puede reemplazar por su expresión.
- En SQL: `WITH` (semana 3).

La misma consulta en SQL: dos alias de agency

algebra.sql, A11

```sql
SELECT r.route_id, o.name AS origen,
       d.name AS destino, r.distance_km
FROM route AS r, agency AS o, agency AS d
WHERE r.origin_agency_id = o.agency_id
  AND r.dest_agency_id = d.agency_id
  AND o.region = d.region
ORDER BY r.route_id;
```

- Cada ruta necesita dos agencias, y las dos salen de la misma relación: dos alias, `o` y `d`.
- Los pasos del álgebra se juntan en una sola consulta.
- Tres relaciones en `FROM`, dos condiciones de join y una selección.

12 rutas dentro de una región

algebra.sql, A11

```
 route_id |          origen          |          destino          | distance_km
----------+--------------------------+---------------------------+-------------
       65 | Chimbote - Centro        | Huaraz - Centro           |          82
       66 | Huaraz - Centro          | Chimbote - Centro         |          82
      103 | Juliaca - Centro         | Puno - Centro             |          60
      104 | Puno - Centro            | Juliaca - Centro          |          60
      113 | Lima - La Victoria       | Lima - Los Olivos         |          22
      114 | Lima - La Victoria       | Lima - Ate                |          22
      115 | Lima - Los Olivos        | Lima - La Victoria        |          22
      116 | Lima - Los Olivos        | Lima - Ate                |          31
      117 | Lima - Ate               | Lima - La Victoria        |          22
      118 | Lima - Ate               | Lima - Los Olivos         |          31
      119 | Arequipa - Centro        | Arequipa - Cerro Colorado |          18
      120 | Arequipa - Cerro Colorado | Arequipa - Centro         |          18
(12 rows)
```

Fundamentales y derivadas

Las seis fundamentales

σ, Π, ρ, $\cup$, $-$, $\times$. Una relación es una expresión; si E_1 y E_2 lo son, también $\sigma_P(E_1)$, $\Pi_S(E_1)$, $\rho_x(E_1)$, $E_1 \cup E_2$, $E_1 - E_2$, $E_1 \times E_2$.

Las demás se escriben con ellas:

- $r \cap s = r - (r - s)$
- $r \bowtie_\theta s = \sigma_\theta(r \times s)$
- $r \bowtie s$: con $\times$, σ y Π
- $\leftarrow$: se reemplaza cada variable

`INTERSECT` y dos `EXCEPT` dan los mismos **1698** clientes (A9 y A9b).

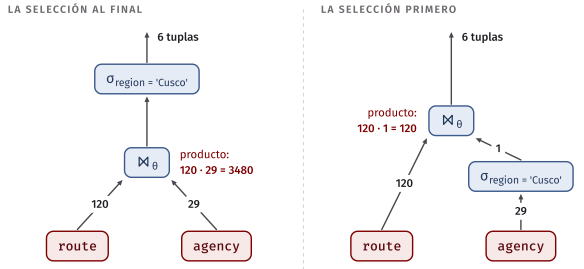
SECCIÓN 06

Consultas equivalentes

06

Dos expresiones, el mismo resultado

Rutas que salen de la región Cusco, con θ: *origin_agency_id* $=$ *agency_id*



Las dos devuelven las mismas **6** rutas. Pero la de la izquierda arma $120 \cdot 29 = 3480$ tuplas antes de filtrar; la de la derecha, $120 \cdot 1 = 120$.

Reglas de equivalencia

Regla	Equivalencia
Cascada de σ	$\sigma_{P_1 \wedge P_2}(E) = \sigma_{P_1}(\sigma_{P_2}(E))$
Cascada de Π	$\Pi_{L_1}(\Pi_{L_2}(E)) = \Pi_{L_1}(E)$, si $L_1 \subseteq L_2$
σ y join θ	$\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$
Empujar σ	$\sigma_P(E_1 \bowtie_\theta E_2) = \sigma_P(E_1) \bowtie_\theta E_2$, si P usa solo atributos de E_1
Conmutatividad	$E_1 \cup E_2 = E_2 \cup E_1$, $E_1 \cap E_2 = E_2 \cap E_1$, $E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$

Equivalentes = iguales en *toda* instancia

$\Pi_{origin_agency_id}(\mathrm{route})$ y $\Pi_{dest_agency_id}(\mathrm{route})$ coinciden hoy, pero basta una ruta nueva sin vuelta para que dejen de coincidir.

La base del optimizador

- El usuario escribe **una** consulta.
- El optimizador *(query optimizer)* genera expresiones equivalentes con reglas como estas.
- Estima el costo de cada una y elige la más barata.

En PostgreSQL

Uno escribe el filtro donde quiera dentro del `WHERE`: el planificador lo empuja por su cuenta. Cómo lo hace, y cómo verlo con `EXPLAIN`: semanas 11 y 12.

SECCIÓN 07

SQL: definición de datos

07



SQL: un estándar, muchos dialectos

- Nace en IBM (System R, *Sequel*), a comienzos de los setenta.
- Hoy: estándar ISO. Cada motor implementa un **dialecto**.
- En el curso: PostgreSQL 18. Lo que no es estándar, lo decimos.

Las partes de SQL

DDL: definir esquemas y restricciones.
DML: consultar, insertar, borrar, modificar.
Además: vistas, transacciones, autorización, SQL desde un programa.

Tipos básicos

Tipo	Qué guarda	En Chasqui
`integer`, `bigint`	enteros de 4 y 8 bytes	`agency_id`, `capacity_kg`
`numeric(p, d)`	decimal exacto: *p* dígitos, *d* decimales	`price numeric(8, 2)`
`char(n)`, `varchar(n)`	texto de largo fijo o con máximo	`dni char(8)`
`text`	texto sin límite (de PostgreSQL)	`name`, `city`
`date`, `timestamptz`	fecha; instante con zona horaria	`created_at`
`boolean`	verdadero o falso	

Error común: dinero en `double precision`

- `real` y `double precision` son binarios y aproximados.
- 0,1 no tiene representación exacta en binario.
- Para `price`, `base_fare` y todo monto: `numeric`.

basics.sql, B2 y B2b

```
 con_double | con_numeric
------------+-------------
 f          | t
(1 row)

     suma_double      | suma_numeric
----------------------+--------------
 0.30000000000000004 |          0.3
(1 row)
```

`CREATE TABLE`: tipos y restricciones

ddl.sql

```sql
CREATE TABLE vehicle (
  vehicle_id  integer,
  plate       varchar(7) NOT NULL,
  kind        text NOT NULL,
  capacity_kg integer NOT NULL,
  CONSTRAINT vehicle_pk PRIMARY KEY (vehicle_id)
);
CREATE TABLE trip (
  trip_id      integer,
  vehicle_id   integer NOT NULL,
  departure_at timestamptz NOT NULL,
  arrival_at   timestamptz,
  CONSTRAINT trip_pk PRIMARY KEY (trip_id),
  CONSTRAINT trip_vehicle_fk FOREIGN KEY (vehicle_id)
    REFERENCES vehicle (vehicle_id)
);
```

- `NOT NULL`: prohíbe el nulo.
- `PRIMARY KEY`: sin duplicados y, de paso, `NOT NULL`.
- `FOREIGN KEY ... REFERENCES`: el valor tiene que existir en la otra tabla.
- Con nombre (`CONSTRAINT`): es el que sale en el error.
- En un esquema aparte, `taller`, para no tocar Chasqui.

\d: lo que quedó definido

psql

```
chasqui_w02=# \d taller.trip
                          Table "taller.trip"
    Column    |           Type           | Collation | Nullable | Default
--------------+--------------------------+-----------+----------+---------
 trip_id      | integer                  |           | not null |
 vehicle_id   | integer                  |           | not null |
 departure_at | timestamp with time zone |           | not null |
 arrival_at   | timestamp with time zone |           |          |
Indexes:
    "trip_pk" PRIMARY KEY, btree (trip_id)
Foreign-key constraints:
    "trip_vehicle_fk" FOREIGN KEY (vehicle_id) REFERENCES taller.vehicle(vehicle_id)
```

- `trip_id` es `not null` sin haberlo escrito: lo implica la clave primaria.
- La clave primaria viene con un índice (semana 10).

Lo que el motor rechaza

ddl.sql, D2 a D5

```
INSERT INTO trip VALUES (2, 99, '2026-03-16 09:00', NULL);
ERROR:  insert or update on table "trip" violates foreign key constraint "trip_vehicle_fk"
DETAIL:  Key (vehicle_id)=(99) is not present in table "vehicle".

INSERT INTO vehicle VALUES (2, NULL, 'van', 900);
ERROR:  null value in column "plate" of relation "vehicle" violates not-null constraint
DETAIL:  Failing row contains (2, null, van, 900).

INSERT INTO vehicle VALUES (1, 'T8D-416', 'bus', 4800);
ERROR:  duplicate key value violates unique constraint "vehicle_pk"
DETAIL:  Key (vehicle_id)=(1) already exists.

INSERT INTO vehicle VALUES (3, 'ABC-12345', 'truck', 12000);
ERROR:  value too long for type character varying(7)
```

Las pruebas comparan el código **SQLSTATE**, no el texto: `23503`, `23502`, `23505`, `22001`.

DELETE, DROP y ALTER

- `DELETE FROM r`: borra las tuplas; la tabla queda, vacía.
- `DROP TABLE r`: borra la relación entera, esquema y tuplas.
- `ALTER TABLE r ADD COLUMN a t`: las filas que ya están quedan con `NULL`.

No se puede borrar una tabla de la que otra depende:

ddl.sql, D6

```
DROP TABLE vehicle;
ERROR:  cannot drop table vehicle because other objects depend on it
DETAIL:  constraint trip_vehicle_fk on table trip depends on table vehicle
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
```

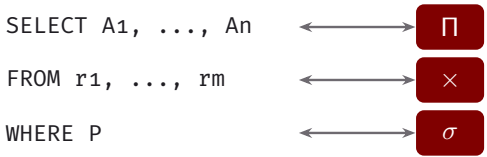
`CASCADE` borra la restricción de `trip`, no la tabla `trip`.

SECCIÓN 08

SELECT-FROM-WHERE

08

La estructura básica



Casi $\Pi_{A_1,\ldots,A_n}(\sigma_P(r_1 \times \cdots \times r_m))$...

...pero con multiconjuntos

`SELECT` **no elimina duplicados**. Y es la definición del significado, no el algoritmo: el motor casi nunca calcula el producto.

Conjuntos y multiconjuntos

basics.sql, B1

```
SELECT city FROM customer
ORDER BY customer_id
LIMIT 5;
   city
----------
 Ayacucho
 Puno
 Lima
 Tacna
 Lima
(5 rows)
```

Un **multiconjunto** *(multiset, bag)* admite elementos repetidos.

basics.sql, B1b

```
 con_duplicados | sin_duplicados
----------------+----------------
           1800 |             26
(1 row)
```

`SELECT DISTINCT city` es la traducción fiel de $\Pi_{city}(\text{customer})$.

¿Por qué SQL no elimina duplicados siempre?

- **Cuesta**: hay que ordenar o armar una tabla hash.
- **Importan**: dos envíos de 2,5 kg no son el mismo envío.

Semántica de multiconjuntos: si t_1 aparece c_1 veces en r y t_2 aparece c_2 veces en s, la tupla $t_1 t_2$ aparece $c_1 \cdot c_2$ veces en $r \times s$.

Error común: `DISTINCT` "por si acaso"

Si el `SELECT` incluye una clave, no hay duplicados y `DISTINCT` solo agrega trabajo. Si los hay, a veces son información.

Varias tablas en `FROM`

Tres relaciones, dos condiciones de join. `customer` también tiene `city`: por eso `a.city`.

basics.sql, B4

```
SELECT s.shipment_id, c.first_name || ' ' || c.last_name AS remitente,
       a.city AS ciudad_origen
FROM shipment AS s, customer AS c, agency AS a
WHERE s.sender_id = c.customer_id
  AND s.origin_agency_id = a.agency_id
ORDER BY s.shipment_id
LIMIT 5;
 shipment_id |         remitente         | ciudad_origen
-------------+---------------------------+---------------
           1 | Luis Mamani Chávez        | Piura
           2 | Kevin Quispe Sánchez      | Cerro de Pasco
           3 | Gonzalo Rodríguez Sánchez | Cusco
           4 | José Yupanqui Ramírez     | Tarapoto
           5 | Lucía García Rojas        | Chiclayo
(5 rows)
```

Error común: olvidar la condición de join

basics.sql, B3b

```
SELECT count(*) AS filas
FROM shipment, customer;
  filas
----------
 21600000
(1 row)
```

12 000 · 1800 filas, sin ningún error.

Regla práctica

Con m relaciones en `FROM` hacen falta al menos $m - 1$ condiciones de join.

Nombres ambiguos

Si dos tablas del `FROM` tienen una columna con el mismo nombre, hay que calificarla:

errors.sql, E3

```
SELECT shipment_id, origin_agency_id
FROM shipment, route
WHERE shipment.trip_id = 1;
ERROR:  column reference "origin_agency_id" is ambiguous
LINE 1: SELECT shipment_id, origin_agency_id
                            ^
```

Se escribe `shipment.origin_agency_id` o, con alias, `s.origin_agency_id`. Aunque no haya ambigüedad, calificar todo deja claro de qué tabla sale cada columna.

SECCIÓN 09

Operaciones adicionales

09

Renombrar con AS

- **Columnas**: `SELECT price * 1.18 AS total`. Sin `AS`, PostgreSQL la llama `?column?`.
- **Tablas**: `FROM shipment AS s`. Es el ρ del álgebra.
- El alias de tabla es obligatorio si la tabla aparece dos veces (`route AS r1, route AS r2`).
- El estándar permite omitir `AS` en el alias de tabla; Oracle no lo acepta ahí. En el curso lo escribimos siempre.

Patrones de texto: `LIKE`

- `%`: cualquier cadena, incluida la vacía.
- `_`: exactamente un carácter.

`'Q%'`	empieza con Q: 51 clientes
`'% Quispe'`	termina en “ Quispe”
`'_3_-%'`	un 3 en la segunda posición

`||` concatena: `first_name || ' ' || last_name`.

basics.sql, B6

```
SELECT plate FROM vehicle
WHERE plate LIKE '_3_-%'
ORDER BY plate;
  plate
---------
 F3E-445
 G3S-419
 H3H-976
 J3D-585
 L3A-234
 S3G-624
(6 rows)
```

Mayúsculas y extremos

`LIKE` distingue mayúsculas; `ILIKE` no.

basics.sql, B6b

```
 con_like | con_ilike
----------+-----------
        0 |        51
(1 row)
```

`ILIKE` es de PostgreSQL. En otros motores: `lower(last_name) LIKE 'q%'`.

`BETWEEN a AND b` es `>= a AND <= b`: incluye los extremos.

basics.sql, B7

```
SELECT route_id, distance_km
FROM route
WHERE distance_km BETWEEN 100 AND 121
ORDER BY distance_km, route_id;
 route_id | distance_km
----------+-------------
       85 |         121
       86 |         121
(2 rows)
```

`ORDER BY` y `LIMIT`

Los diez envíos más pesados del 16 de marzo de 2026, del más pesado al más liviano:

basics.sql, B8

```sql
SELECT shipment_id, weight_kg, created_at
FROM shipment
WHERE created_at >= '2026-03-16' AND created_at < '2026-03-17'
ORDER BY weight_kg DESC, shipment_id
LIMIT 10;
```

- Día semiabierto: `>=` el inicio, `<` el día siguiente (hora de Lima, `-05`).
- `shipment_id` desempata: sin él, el décimo puesto podría cambiar de una corrida a otra.
- `LIMIT` no es estándar; el estándar es `FETCH FIRST 10 ROWS ONLY` (mismo resultado, B8b).

El resultado

psql

```
 shipment_id | weight_kg |       created_at
-------------+-----------+------------------------
        3645 |    144.60 | 2026-03-16 12:26:00-05
        5074 |    123.10 | 2026-03-16 01:29:00-05
        1797 |    113.80 | 2026-03-16 05:55:00-05
        6182 |    111.00 | 2026-03-16 19:42:00-05
        1306 |    109.60 | 2026-03-16 17:34:00-05
        4720 |    102.70 | 2026-03-16 23:20:00-05
       10435 |    101.00 | 2026-03-16 17:45:00-05
        4251 |     91.60 | 2026-03-16 22:35:00-05
        8083 |     89.70 | 2026-03-16 22:47:00-05
        4669 |     82.90 | 2026-03-16 06:32:00-05
(10 rows)
```

- Ese día hay **68** envíos; `LIMIT` se queda con los diez primeros después de ordenar.
- `created_at` es `timestamptz`: se muestra en la zona horaria de la sesión.

Error común: suponer un orden sin `ORDER BY`

Una tabla no tiene orden

Si `SELECT * FROM agency` sale ordenado por `agency_id`, es casualidad: las filas se insertaron en ese orden y la tabla es pequeña.

- Después de unas actualizaciones, o con otro plan, puede salir en otro orden.
- Si el orden importa, se pide con `ORDER BY`.
- `LIMIT` sin `ORDER BY` devuelve “algunas” filas, no “las primeras”.

El orden lógico de evaluación



aquí nace `total`

errors.sql, E4

```
SELECT price * 1.18 AS total
FROM shipment
WHERE total > 100;
ERROR:  column "total" does not exist
LINE 3: WHERE total > 100;
              ^
```

basics.sql, B5

```
SELECT shipment_id, price, price * 1.18 AS total
FROM shipment
WHERE price * 1.18 > 230
ORDER BY total DESC, shipment_id;
 shipment_id | price  |  total
-------------+--------+---------
        3645 | 201.98 | 238.3364
        1618 | 196.86 | 232.2948
(2 rows)
```

SECCIÓN 10

Del álgebra a SQL

10



La correspondencia

Álgebra	SQL	Ojo
$\sigma_P(r)$	`SELECT * FROM r WHERE P`	
$\Pi_{A,B}(r)$	`SELECT DISTINCT A, B FROM r`	sin `DISTINCT`, duplicados
$r \times s$	`SELECT * FROM r, s`	
$r \bowtie_\theta s$	`SELECT * FROM r, s WHERE` θ	
$r \bowtie s$	una igualdad por atributo común	`NATURAL JOIN`: semana 4
$\rho_x(r)$	`FROM r AS x`	
$r \cup s$, $r - s$, $r \cap s$	`UNION`, `EXCEPT`, `INTERSECT`	sí eliminan duplicados
$x \leftarrow E$	`WITH x AS (E) ...`	semana 3

Receta

Las relaciones (con su ρ) al `FROM`; todas las condiciones, con `AND`, al `WHERE`; la Π exterior al `SELECT`, con `DISTINCT` si puede haber duplicados.

Las operaciones de conjuntos sí eliminan duplicados

basics.sql, B10

```
 union_all | union_conjunto
-----------+----------------
      1829 |             26
(1 row)
```

- `UNION ALL`: las 29 ciudades de `agency` (Lima tres veces) más las 1800 de `customer`.
- `UNION`: las 26 ciudades distintas.
- `ALL` y multiconjuntos: semana 3.

Idea clave

Una consulta sin `DISTINCT` devuelve las mismas tuplas que su expresión del álgebra, pero algunas varias veces. Para compararlas, se comparan como conjuntos.

Un ejemplo completo

Las agencias de origen de los envíos de más de 145 kg:

$$\Pi_{name}(\sigma_{weight_kg>145}(\text{shipment}) \bowtie_{origin_agency_id=agency_id} \text{agency})$$

basics.sql, B11

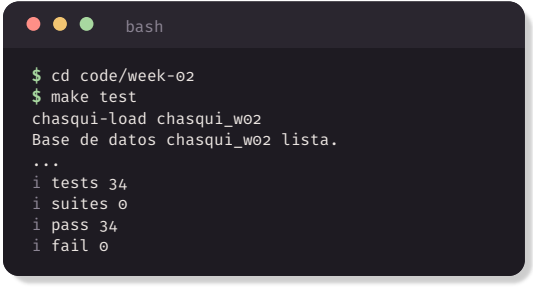
```sql
SELECT DISTINCT a.name
FROM shipment AS s, agency AS a
WHERE s.origin_agency_id = a.agency_id
  AND s.weight_kg > 145
ORDER BY a.name;
```

- La σ quedó en el mismo `WHERE` que la condición de join: en SQL no se dice dónde va cada filtro.
- Sin `DISTINCT`, una agencia con dos envíos pesados sale dos veces.

El código de la semana

`code/week-02`:

- `algebra.sql`: A1–A12, cada expresión con su SQL.
- `basics.sql`: B1–B11.
- `ddl.sql`, `errors.sql`: lo que el motor rechaza.
- `*.test.mjs`: formulaciones equivalentes dan el mismo multiconjunto, y cada número de los apuntes sale de los datos.



Resumen

- Álgebra: procedimental, cerrada. Fundamentales: σ, Π, ρ, $\cup$, $-$, $\times$.
- $r \bowtie_\theta s = \sigma_\theta(r \times s)$; $|r \times s| = |r| \cdot |s|$.
- Natural join: iguala *todos* los nombres comunes. Cuidado.
- Equivalentes: iguales en toda instancia. Empujar σ abarata.

- `CREATE TABLE` con `NOT NULL`, `PRIMARY KEY`, `FOREIGN KEY`. Dinero en `numeric`.
- `SELECT-FROM-WHERE` $\approx$ Π-$\times$-σ, con multiconjuntos.
- `AS`, `LIKE`, `BETWEEN`, `ORDER BY` con desempate, `LIMIT`.
- Orden lógico: `FROM`, `WHERE`, `SELECT`, `ORDER BY`, `LIMIT`.

Para la próxima clase

- Laboratorio: primeras consultas con `psql` (`lab02.sql`).
- Hacer los ejercicios de los apuntes y correr `make test` en `code/week-02`.
- Leer Silberschatz, secs. 3.5–3.7 (conjuntos, `NULL`, agregación); Berkeley CS186, *Note 1: SQL I*, secciones *Filtering Null Values*, *Grouping and Aggregation* y *A Word of Caution*; y PostgreSQL, *Aggregate Functions* y *Combining Queries*.
- P0 (Calentamiento) se entrega en la semana 3.

Pregunta

¿Por qué $\Pi_{city}(\text{customer})$ y `SELECT city FROM customer` devuelven un número distinto de filas? ¿Qué hay que agregar al SQL para que coincidan?