Pauta Ayudantía 4

IIC2413, Bases de Datos
Benjamín Saavedra

Notación (recordatorio)

- Para una relación R, $\mathsf{att}(R)$ denota su conjunto de atributos.
- **Operadores base del álgebra relacional** (para relaciones R, S, atributos A, B y constante $a \in \mathbf{Data}$):
  - $R \cup S,\ R \cap S,\ R \setminus S$ (definidos si $\mathsf{att}(R) = \mathsf{att}(S)$)
  - $\sigma_{A=a}(R),\ \sigma_{A=B}(R)$, y su generalización $\sigma_\theta(R)$ con θ fórmula booleana de igualdades $(\wedge, \vee, \neg)$, además de $\sigma_{A\sim a}, \sigma_{A\sim B}$ con $\sim \in \{<, >, \leq, \geq\}$.
  - $\rho_{A\to B}(R)$ (definido si $A \in \mathsf{att}(R)$, $B \notin \mathsf{att}(R)$)
  - $\pi_{B_1,\ldots,B_k}(R)$
  - $R \bowtie S$ (join natural), $R \times S$ (caso particular de $\bowtie$ cuando $\mathsf{att}(R) \cap \mathsf{att}(S) = \varnothing$).
- **Consultas de primer orden**: $Q(x_1, \ldots, x_n) \leftarrow \varphi$, donde φ es una fórmula con $\mathsf{FV}(\varphi) \subseteq \{x_1, \ldots, x_n\}$ construida a partir de $x = a$, $x = y$, $R(t_1, \ldots, t_k)$ mediante $\wedge, \vee, \neg, \exists x, \forall x$. Recuerde que $\forall x.\, \varphi \equiv \neg\exists x.\, \neg\varphi$, clave para traducir consultas del tipo *"para todos"/"ninguno"*.

Recordatorio: SQL, agregación y nulos

- Plantilla general: `SELECT ... FROM ... WHERE ... GROUP BY ... HAVING ...`. Las funciones de agregación (`COUNT`, `SUM`, `AVG`, `MIN`, `MAX`) **no existen** en álgebra relacional ni en consultas de primer orden: cualquier consulta que las use excede el poder expresivo de ambos formalismos.
- `EXISTS`/`NOT EXISTS` (correlacionados) e `IN`/`NOT IN` (con subconsulta) son las herramientas de SQL para expresar $\exists/\neg\exists$, y combinando dos de ellos se obtiene $\forall$ (vía $\forall x.\varphi \equiv \neg\exists x.\neg\varphi$).
- Lógica de 3 valores: toda comparación (`=`,`<`,...) con `NULL` vale `unknown`. Sólo `IS NULL`/ `IS NOT NULL` valen siempre `true`/`false`. Una fila satisface el `WHERE` si y sólo si la condición vale `true` (`unknown` se descarta igual que `false`).

θ	`NOT` θ
true	false
false	true
unknown	unknown

θ_1	θ_2	`OR`	`AND`
true	unknown	true	unknown
false	unknown	unknown	false
unknown	unknown	unknown	unknown

- **Cuidado con** `NOT IN`: si la subconsulta puede devolver algún `NULL`, entonces `x NOT IN (...)` vale `unknown` **para toda fila** (incluso las que uno esperaría que calificaran), y el resultado puede quedar vacío. La alternativa segura es `NOT EXISTS` correlacionado.

Consejos generales

- Identifique primero *qué relaciones debe combinar* y con qué atributo se conectan.
- Para *"que tengan dos o más..."*: dos testigos distintos (dos $\exists$ con $\neg(\cdot = \cdot)$, o dos copias renombradas con ρ).
- Para *"que NO tengan ningún..."*: negación de un existencial en lógica, o $\setminus$ en álgebra.
- Para *"para TODOS los..."*: use $\forall x.\varphi \equiv \neg\exists x.\neg\varphi$. En álgebra, separe un **universo de candidatos** de una relación de **evidencia**, y reste los pares candidato-objetivo que falten. En esta ayudantía el "objetivo" del $\forall$ puede ser a su vez el resultado de un $\exists$ anidado: identifique primero ese conjunto derivado antes de aplicar el patrón.
- En álgebra, al comparar dos tuplas de la misma relación, **renombre una copia con** ρ antes de $\bowtie/\times$.

1 Parte A: Consultas en contexto

Para cada consulta, salvo que se indique lo contrario, entregue **(i)** una consulta de primer orden, **(ii)** una expresión de álgebra relacional, y **(iii)** la consulta SQL equivalente.

1.1 Contexto: Ferretería y Distribuidora "El Tornillo Feliz"

Una PYME chilena dedicada a la venta de artículos de ferretería registra su operación en el siguiente esquema:

$$\begin{aligned}
\textbf{Sucursal}(\mathsf{idSucursal}, \mathsf{comuna}, \mathsf{direccion}) &\quad \mathsf{ar} = 3\\
\textbf{Empleado}(\mathsf{rut}, \mathsf{nombre}, \mathsf{cargo}, \mathsf{idSucursal}) &\quad \mathsf{ar} = 4\\
\textbf{Cliente}(\mathsf{rutCliente}, \mathsf{nombre}, \mathsf{comuna}) &\quad \mathsf{ar} = 3\\
\textbf{Proveedor}(\mathsf{idProveedor}, \mathsf{nombre}, \mathsf{pais}) &\quad \mathsf{ar} = 3\\
\textbf{Producto}(\mathsf{sku}, \mathsf{nombre}, \mathsf{categoria}, \mathsf{precio}) &\quad \mathsf{ar} = 4\\
\textbf{Suministra}(\mathsf{idProveedor}, \mathsf{sku}) &\quad \mathsf{ar} = 2\\
\textbf{Pedido}(\mathsf{idPedido}, \mathsf{rutCliente}, \mathsf{rut}, \mathsf{idSucursal}, \mathsf{fecha}) &\quad \mathsf{ar} = 5\\
\textbf{DetallePedido}(\mathsf{idPedido}, \mathsf{sku}, \mathsf{cantidad}) &\quad \mathsf{ar} = 3
\end{aligned}$$

donde el atributo idSucursal de **Empleado** indica en qué sucursal trabaja el empleado *rut* (en SQL, este atributo puede ser `NULL` para un empleado recién contratado que aún no ha sido asignado a una sucursal). **Suministra** es multivaluada: un producto puede ser suministrado por varios proveedores, y un proveedor suministra varios productos. **Pedido**($idPedido, rutCliente, rut, idSucursal, fecha$) indica que el pedido *idPedido* fue realizado por el cliente *rutCliente*, atendido por el empleado *rut*, en la sucursal *idSucursal* (en SQL, *rutCliente* puede ser `NULL` para una venta de mostrador a un cliente no registrado). Y **DetallePedido**($idPedido, sku, cantidad$) indica que el pedido *idPedido* incluyó *cantidad* unidades del producto *sku*. El precio de un producto (en SQL) puede ser `NULL` si aún no ha sido fijado.

Instancia de ejemplo (opcional, para verificar sus respuestas: los valores `NULL` sólo son relevantes para las consultas en SQL):

Sucursal

idSucursal	comuna	direccion
1	Providencia	Av. Providencia 1234
2	Maipú	Camino Rinconada 500
3	Las Condes	Av. Apoquindo 4500

Empleado

rut	nombre	cargo	idSucursal
101	Juan Pérez	Vendedor Senior	1
102	María Soto	Vendedor	1
103	Pedro Gómez	Vendedor Senior	2
104	Camila Ruiz	Bodeguero	`NULL`

Cliente

rutCliente	nombre	comuna
201	Ana Torres	Providencia
202	Luis Vidal	Maipú
203	Carla Muñoz	Ñuñoa
204	Diego Rojas	Providencia

Proveedor

idProveedor	nombre	pais
301	FerroChile	Chile
302	ToolMax	Estados Unidos
303	Hierros del Sur	Chile
304	NordicTools	Suecia

Producto

sku	nombre	categoria	precio
1	Tornillo 1"	Tornillos	50
2	Tornillo 2"	Tornillos	70
3	Pintura Blanca 1L	Pinturas	8990
4	Taladro Eléctrico	Herramientas	NULL
5	Combo de Brocas	Herramientas	5990
6	Guantes de Seguridad	Seguridad	2990

Suministra

idProveedor	sku
301	1
301	3
302	1
302	4
303	2
303	5
304	6

Pedido

idPedido	rutCliente	rut	idSucursal	fecha
1001	201	101	1	2026-01-05
1002	202	103	2	2026-01-06
1003	201	103	2	2026-01-10
1004	203	101	1	2026-01-07
1005	NULL	102	1	2026-01-08
1006	204	101	1	2026-01-09

DetallePedido

idPedido	sku	cantidad
1001	1	10
1001	3	2
1002	2	5
1003	5	1
1004	1	3
1005	2	4
1006	3	1

1.2 Ejercicio A.1: clientes con pedido en todas las sucursales con vendedor senior

Enunciado. Obtenga el nombre de los clientes que han realizado al menos un pedido en **todas** las sucursales que cuentan con algún empleado de cargo "Vendedor Senior".

Construcción en CPO

Como el nombre no es la llave de **Cliente**, entramos primero a la tabla completa introduciendo variables para todos sus atributos, dejando existencialmente cuantificadas las que no van en la salida (el rut, para referirnos al cliente más adelante, y la comuna, sin otro uso). La palabra *"todas"* señala un cuantificador universal $\forall$: como CPO no tiene sintaxis para "$\forall s \in T$" restringido a un conjunto T, el $\forall$ cuantifica sobre todo el dominio, y la condición "que cuentan con algún empleado de cargo 'Vendedor Senior"' entra dentro de la fórmula como el antecedente de una implicación: "si s tiene algún vendedor senior, entonces…". Como CPO no tiene $\rightarrow$, usamos la equivalencia $A \rightarrow B \equiv \neg A \vee B$. Ese antecedente $A(s)$ es, a su vez, un existencial ("algún empleado"):

$$A(s) \;:=\; \exists re.\exists ne.\ \mathbf{Empleado}(re, ne, \text{VendedorSenior}, s)$$

y el consecuente, "al menos un pedido", es otro existencial sobre **Pedido**, correlacionado con el cliente r y la sucursal s:

$$B(r, s) \;:=\; \exists ip.\exists ree.\exists f.\ \mathbf{Pedido}(ip, r, ree, s, f)$$

Juntando todo:

$$Q(y) \leftarrow \exists r.\exists c.\ \mathbf{Cliente}(r, y, c) \;\wedge\; \forall s.\big(\neg A(s) \;\vee\; B(r, s)\big)$$

Construcción en Álgebra

Antes de proyectar nombre necesitamos el universo de candidatos, porque el filtro "para todos" se construye restando de ese universo a quienes no califican:

$$C \;:=\; \pi_{\mathsf{rutCliente}}(\mathbf{Cliente})$$

El álgebra no tiene $\forall$. Lo simulamos construyendo primero el conjunto objetivo T (las sucursales que tienen algún vendedor senior) y la evidencia F (los pares cliente-sucursal donde sí hubo un pedido):

$$T \;:=\; \pi_{\mathsf{idSucursal}}\big(\sigma_{\mathsf{cargo}=\text{VendedorSenior}}(\mathbf{Empleado})\big), \qquad F \;:=\; \pi_{\mathsf{rutCliente},\mathsf{idSucursal}}(\mathbf{Pedido})$$

Con C, T y F ya construidos aplicamos el patrón candidatos/evidencia: generamos todos los pares que deberían existir en F si el $\forall$ se cumpliera para cualquier cliente, usando $\times$ (no $\bowtie$, pues C y T no comparten ningún atributo), restamos la evidencia real para encontrar los pares faltantes, y proyectamos por rutCliente para quedarnos con los clientes que fallan para alguna sucursal de T:

$$\mathit{Fallan} \;:=\; \pi_{\mathsf{rutCliente}}\big((C \times T) \setminus F\big)$$

Los clientes que sí satisfacen el $\forall$ son los candidatos que no están entre los que fallan. Recuperamos el nombre uniendo de vuelta con **Cliente**:

$$Q \;=\; \pi_{\mathsf{nombre}}\big(\mathbf{Cliente} \bowtie (C \setminus \mathit{Fallan})\big)$$

Construcción en SQL

Seleccionamos el nombre desde **Cliente**. "Todas" se traduce como "no existe una excepción": iteramos sobre **Sucursal** y buscamos que ninguna incumpla la condición. Dentro de ese filtro, "algún" empleado de cargo 'Vendedor Senior' es un `EXISTS` correlacionado con la sucursal s. Falta expresar que el cliente c no tiene pedido en esa sucursal, la ausencia de evidencia que buscamos dentro del `NOT EXISTS` externo:

```
SELECT c.nombre
FROM Cliente c
WHERE NOT EXISTS (
    SELECT s.idSucursal FROM Sucursal s
    WHERE EXISTS (
        SELECT 1 FROM Empleado e
        WHERE e.idSucursal = s.idSucursal
          AND e.cargo = 'Vendedor Senior'
    )
    AND NOT EXISTS (
        SELECT 1 FROM Pedido p
        WHERE p.rutCliente = c.rutCliente
          AND p.idSucursal = s.idSucursal
    )
);
```

1.3 Ejercicio A.2: proveedores extranjeros sin producto compartido con un chileno

Enunciado. Obtenga el nombre de los proveedores **extranjeros** (país distinto de "Chile") que **no** suministran ningún producto que también sea suministrado por algún proveedor chileno.

Construcción en CPO

Igual que en A.1, entramos a **Proveedor** completo y agregamos de una vez el filtro "extranjero" (depende sólo del propio proveedor). La palabra *"ningún"* señala $\neg\exists$ directo: a diferencia de A.1, aquí no hay ninguna implicación, así que abrimos la negación directamente sobre los productos (sk) que el proveedor suministra. La condición "que también sea suministrado por algún proveedor chileno" es lo que, junto con "id suministra sk", hace de sk un producto prohibido: un nuevo existencial anidado dentro de la negación:

$$\begin{aligned} Q(y) \leftarrow \exists id.\exists p.\ \mathbf{Proveedor}(id, y, p) \wedge \neg(p = \text{Chile}) \ \wedge \\ \neg\exists sk.\Big(\mathbf{Suministra}(id, sk) \ \wedge \\ \exists id_2.\exists n_2.\exists p_2.\ \mathbf{Proveedor}(id_2, n_2, p_2) \wedge p_2 = \text{Chile} \wedge \mathbf{Suministra}(id_2, sk)\Big) \end{aligned}$$

Construcción en Álgebra

Empezamos por la parte independiente y más simple: qué productos tienen respaldo chileno, primero los proveedores chilenos y después los productos que suministran. Por el otro lado, los proveedores extranjeros y qué productos suministra cada uno:

$$\begin{aligned} \mathit{Chilenos} &:= \pi_{\mathsf{idProveedor}}\big(\sigma_{\mathsf{pais}=\text{Chile}}(\mathbf{Proveedor})\big) \\ \mathit{ProdChilenos} &:= \pi_{\mathsf{sku}}(\mathbf{Suministra} \bowtie \mathit{Chilenos}) \\ \mathit{Extranjeros} &:= \pi_{\mathsf{idProveedor}}\big(\sigma_{\mathsf{pais}\neq\text{Chile}}(\mathbf{Proveedor})\big) \\ \mathit{SumExt} &:= \mathbf{Suministra} \bowtie \mathit{Extranjeros} \end{aligned}$$

La negación se implementa como resta: primero encontramos a los que fallan (comparten un producto con un chileno), los pares de *SumExt* cuyo sk también está en *ProdChilenos*:

$$\mathit{Fallan} := \pi_{\mathsf{idProveedor}}(\mathit{SumExt} \bowtie \mathit{ProdChilenos})$$

Los proveedores que buscamos son los extranjeros que no están en *Fallan*. Recuperamos el nombre uniendo de vuelta con **Proveedor**:

$$Q = \pi_{\mathsf{nombre}}\big(\mathbf{Proveedor} \bowtie (\mathit{Extranjeros} \setminus \mathit{Fallan})\big)$$

Construcción en SQL

El filtro "extranjero" no requiere subconsulta, pues es un atributo de la propia tupla. "Ningún" se traduce como `NOT EXISTS`, iterando sobre los productos que `p1` suministra. "Algún" proveedor chileno es un segundo `EXISTS`, correlacionado con el `sku` del producto que estamos revisando, buscando un proveedor chileno que suministre ese mismo producto:

```
SELECT p1.nombre
FROM Proveedor p1
WHERE p1.pais <> 'Chile'
  AND NOT EXISTS (
      SELECT 1 FROM Suministra s1
      WHERE s1.idProveedor = p1.idProveedor
        AND EXISTS (
            SELECT 1 FROM Suministra s2
```

```
            JOIN Proveedor p2 ON s2.idProveedor = p2.idProveedor
            WHERE s2.sku = s1.sku AND p2.pais = 'Chile'
        )
    );
```

1.4 Ejercicio A.3: monto vendido por sucursal sobre el promedio (única con agregación, responda sólo en SQL)

Enunciado. Obtenga el identificador y la comuna de cada sucursal, junto con el monto total vendido (suma de cantidad×precio de todos los pedidos que atendió), pero sólo para las sucursales cuyo monto total supere el promedio de los montos totales de todas las sucursales.

Las palabras gatillo "suma" y "promedio" son agregación, que no existe en CPO ni en álgebra relacional, así que esta consulta se construye directamente en SQL. El precio está en **Producto**, la cantidad en **DetallePedido**, y la sucursal en **Pedido**. Armamos el `JOIN` y el `SUM`. Como el monto es por sucursal, agrupamos con `GROUP BY` (si no, SQL se quejaría de mezclar `s.comuna` con `SUM(...)`), y la condición "sólo para las sucursales cuyo monto supere el promedio" es sobre el grupo ya agregado, así que va en `HAVING` (no en `WHERE`, que filtra antes de agrupar). No podemos escribir `AVG(SUM(...))` directamente, pues eso agregaría dos veces sobre el mismo `GROUP BY`: la solución es calcular el monto por sucursal en una subconsulta separada y aplicar `AVG` sobre esa tabla ya agregada:

```
SELECT s.idSucursal, s.comuna, SUM(d.cantidad * pr.precio) AS montoTotal
FROM Sucursal s
JOIN Pedido p ON p.idSucursal = s.idSucursal
JOIN DetallePedido d ON d.idPedido = p.idPedido
JOIN Producto pr ON pr.sku = d.sku
GROUP BY s.idSucursal, s.comuna
HAVING SUM(d.cantidad * pr.precio) > (
    SELECT AVG(monto)
    FROM (
        SELECT SUM(d2.cantidad * pr2.precio) AS monto
        FROM Pedido p2
        JOIN DetallePedido d2 ON d2.idPedido = p2.idPedido
        JOIN Producto pr2 ON pr2.sku = d2.sku
        GROUP BY p2.idSucursal
    ) AS ventasPorSucursal
);
```

Con la instancia: la Sucursal 1 (Providencia) vende $18480+150+280+8990 = 27900$, y la Sucursal 2 (Maipú) vende $350 + 5990 = 6340$. El promedio de estos dos montos es 17120, así que sólo la Sucursal 1 supera el promedio: el resultado es $(1, \text{Providencia}, 27900)$. La Sucursal 3 (Las Condes) no tiene pedidos, por lo que el `JOIN` interno la descarta por completo y tampoco participa en el promedio.

2 Parte B: división relacional, poder expresivo y propiedades

Recuerde la definición de $\div$ dada en la Notación: para R, S con $\mathsf{att}(S) \subsetneq \mathsf{att}(R)$ y $A := \mathsf{att}(R) \setminus \mathsf{att}(S)$,

$$R \div S \;:=\; \big\{\bar{\tau} \in \pi_A(R) \;\big|\; \forall \bar{\sigma} \in S.\ (\bar{\tau} \bowtie \bar{\sigma}) \in R\big\}.$$

2.1 Ejercicio B.1: ¿÷ agrega poder expresivo?

Antes de demostrar propiedades de $\div$, verifiquemos que este operador **no** extiende el álgebra relacional a un lenguaje más expresivo: debemos poder simularlo usando sólo los operadores base $(\cup, \cap, \setminus, \sigma, \rho, \pi, \bowtie)$.

(a) ¿Por qué la definición exige explícitamente que $\bar{\tau} \in \pi_A(R)$, en lugar de simplemente pedir que $\bar{\tau}$ sea cualquier tupla con atributos A?

(b) Demuestre que

$$R \div S \;=\; \pi_A(R) \;\setminus\; \pi_A\Big(\big(\pi_A(R) \times S\big) \setminus R\Big).$$

Parte (a)

Demostración. Si $S = \varnothing$, la condición "$\forall \bar{\sigma} \in S. \ldots$" se cumple **vacuamente** para cualquier tupla $\bar{\tau}$. Sin la exigencia $\bar{\tau} \in \pi_A(R)$, *cualquier* tupla con atributos A (tomando valores arbitrarios de **Data**, que es infinito) calificaría, y $R \div S$ no sería un conjunto finito, violando la definición de relación. Exigir $\bar{\tau} \in \pi_A(R)$ restringe los candidatos a los finitos valores que efectivamente aparecen en R, evitando este problema.

Parte (b)

Usamos el único hecho auxiliar que se necesita:

Lema. Sean $\bar{\tau}, \bar{\sigma}$ tuplas con $\mathsf{att}(\bar{\tau}) = A$, $\mathsf{att}(\bar{\sigma}) = B$, y $A \cap B = \varnothing$ (esto se cumple siempre que $\bar{\tau} \in \pi_A(R)$ y $\bar{\sigma} \in S$, ya que $A := \mathsf{att}(R) \setminus B$ con $B := \mathsf{att}(S)$). Entonces

$$\pi_A(\bar{\tau} \bowtie \bar{\sigma}) \;=\; \bar{\tau}.$$

Demostración. Como $\mathsf{att}(\bar{\tau}) \cap \mathsf{att}(\bar{\sigma}) = \varnothing$, la definición de $\bowtie$ (Clase 02) da $[\bar{\tau} \bowtie \bar{\sigma}](C) = \bar{\tau}(C)$ para $C \in A$ y $[\bar{\tau} \bowtie \bar{\sigma}](C) = \bar{\sigma}(C)$ para $C \in B$. Al proyectar sobre A sólo nos quedamos con el primer caso, así que para todo $C \in A$: $[\pi_A(\bar{\tau} \bowtie \bar{\sigma})](C) = \bar{\tau}(C)$. Como ambas tuplas tienen dominio A y coinciden en todo $C \in A$, son iguales. ■

Demostración. Partimos del lado **derecho** de la igualdad que queremos probar, y lo vamos simplificando hasta llegar al lado izquierdo.

Paso 1 (desarmar $\times$). Por la definición de $\times$ (válida porque $\mathsf{att}(\pi_A(R)) = A$, $\mathsf{att}(S) = B$, $A \cap B = \varnothing$):

$$\pi_A(R) \times S \;=\; \{\bar{\tau} \bowtie \bar{\sigma} \mid \bar{\tau} \in \pi_A(R) \;\wedge\; \bar{\sigma} \in S\}.$$

Paso 2 (desarmar $\setminus$, sustituyendo el Paso 1). Por la definición de $\setminus$ (quedarse con lo que **no** está en R):

$$\big(\pi_A(R) \times S\big) \setminus R \;=\; \{\bar{\tau} \bowtie \bar{\sigma} \mid \bar{\tau} \in \pi_A(R) \;\wedge\; \bar{\sigma} \in S \;\wedge\; \bar{\tau} \bowtie \bar{\sigma} \notin R\}.$$

Paso 3 (desarmar π_A: ¿de dónde sale el $\exists$?). El conjunto del Paso 2 se arma probando cada pareja $(\bar{\tau}, \bar{\sigma})$ posible y quedándose con las que fallan ($\bar{\tau} \bowtie \bar{\sigma} \notin R$). Si un mismo candidato $\bar{\tau}$ falla

con **más de un** $\bar{\sigma}$, el conjunto tiene **varias** tuplas distintas con ese mismo $\bar{\tau}$. Por ejemplo, si $\bar{\tau} = 3$ falla tanto con $\bar{\sigma}$=1 como con $\bar{\sigma}$=2, el conjunto del Paso 2 contiene *dos* tuplas: $(A{:}3, X{:}1)$ y $(A{:}3, X{:}2)$.

Ahora aplicamos π_A, es decir, le **borramos la parte de** $\bar{\sigma}$ a cada tupla (por el Lema): esas dos tuplas distintas se convierten en el **mismo** valor, 3. Como estamos hablando de un conjunto (no de una lista), da lo mismo que hayan "llegado" dos veces: 3 queda ahí una sola vez.

Al proyectar perdimos justo la información de **cuál** $\bar{\sigma}$ causó el problema (¿fue el 1? ¿el 2? ¿los dos?). Lo único que sobrevive es la versión más débil de la pregunta: *"¿hubo algún $\bar{\sigma}$ que hiciera fallar a este candidato?"*, y esa pregunta, en símbolos, es exactamente $\exists\bar{\sigma}$. Por eso:

$$\pi_A\Big(\big(\pi_A(R) \times S\big) \setminus R\Big) \;=\; \{\bar{\tau} \in \pi_A(R) \mid \exists\bar{\sigma} \in S.\; \bar{\tau} \bowtie \bar{\sigma} \notin R\}.$$

Llamemos Y a este último conjunto.

Paso 4 (desarmar $\setminus$ otra vez, ahora restando Y). Por construcción, $Y \subseteq \pi_A(R)$ (la condición "$\bar{\tau} \in \pi_A(R)$" ya es parte de la definición de Y). Restar un subconjunto de esa forma a $\pi_A(R)$ equivale simplemente a negar la condición extra que lo define:

$$\pi_A(R) \setminus Y \;=\; \{\bar{\tau} \in \pi_A(R) \mid \bar{\tau} \notin Y\} \;=\; \{\bar{\tau} \in \pi_A(R) \mid \neg\exists\bar{\sigma} \in S.\; \bar{\tau} \bowtie \bar{\sigma} \notin R\}.$$

(Justificación de esta última igualdad: para $\bar{\tau} \in \pi_A(R)$, decir $\bar{\tau} \notin Y$ es decir que **no** se cumple la condición que define a Y. Esa condición, aplicada a un $\bar{\tau}$ que ya sabemos que está en $\pi_A(R)$, se reduce a "$\exists\bar{\sigma} \in S.\bar{\tau} \bowtie \bar{\sigma} \notin R$".)

Paso 5 (aplicar $\neg\exists\neg \equiv \forall$). Con $\psi(\bar{\sigma}) := (\bar{\tau} \bowtie \bar{\sigma} \in R)$, la condición "$\bar{\tau} \bowtie \bar{\sigma} \notin R$" es exactamente $\neg\psi(\bar{\sigma})$, así que el paso anterior dice $\neg\exists\bar{\sigma} \in S.\neg\psi(\bar{\sigma})$. Por la equivalencia lógica estándar $\neg\exists x \in S.\neg\psi(x) \iff \forall x \in S.\psi(x)$:

$$\pi_A(R) \setminus Y \;=\; \{\bar{\tau} \in \pi_A(R) \mid \forall\bar{\sigma} \in S.\; \bar{\tau} \bowtie \bar{\sigma} \in R\}.$$

Paso 6 (reconocer la definición de $\div$). El conjunto de la derecha es **exactamente** la definición de $R \div S$. Por lo tanto:

$$\pi_A(R) \setminus \pi_A\Big(\big(\pi_A(R) \times S\big) \setminus R\Big) \;=\; R \div S. \qquad \blacksquare$$

2.2 Ejercicio B.2

Demuestre que

$$R \div (S \cup S') \;=\; (R \div S) \cap (R \div S').$$

Solución. Verdadera. Sea $B := \mathsf{att}(S) = \mathsf{att}(S')$ y $A := \mathsf{att}(R) \setminus B$. Nótese que $\mathsf{att}(R \div (S \cup S')) = A = \mathsf{att}((R \div S) \cap (R \div S'))$, así que basta comparar las tuplas. Para toda $\bar{\tau}$:

$$\begin{aligned}
\bar{\tau} \in R \div (S \cup S') &\iff \bar{\tau} \in \pi_A(R) \;\wedge\; \forall\bar{\sigma} \in (S \cup S').\; (\bar{\tau} \bowtie \bar{\sigma}) \in R \\
&\iff \bar{\tau} \in \pi_A(R) \;\wedge\; \Big(\forall\bar{\sigma} \in S.\; (\bar{\tau} \bowtie \bar{\sigma}) \in R\Big) \\
&\qquad \wedge\; \Big(\forall\bar{\sigma} \in S'.\; (\bar{\tau} \bowtie \bar{\sigma}) \in R\Big) \\
&\iff \Big(\bar{\tau} \in \pi_A(R) \wedge \forall\bar{\sigma} \in S.\; (\bar{\tau} \bowtie \bar{\sigma}) \in R\Big) \\
&\qquad \wedge\; \Big(\bar{\tau} \in \pi_A(R) \wedge \forall\bar{\sigma} \in S'.\; (\bar{\tau} \bowtie \bar{\sigma}) \in R\Big) \\
&\iff \bar{\tau} \in (R \div S) \;\wedge\; \bar{\tau} \in (R \div S') \\
&\iff \bar{\tau} \in (R \div S) \cap (R \div S').
\end{aligned}$$

La segunda equivalencia usa que "$\forall x$ en una unión" equivale a "$\forall x$ en cada conjunto por separado" (si algo vale para todo elemento de $S \cup S'$, en particular vale para todo elemento de S y para todo elemento de S', y viceversa), y la tercera simplemente duplica el conjunto $\bar{\tau} \in \pi_A(R)$ (válido porque $P \wedge P \equiv P$). ■

2.3 Ejercicio B.3

¿Es cierto, en general, que

$$R \div (S \cap S') \;=\; (R \div S) \cup (R \div S') \;?$$

Demuestre que la contención $\supseteq$ siempre se cumple, y decida si la igualdad es válida en general: si lo es, demuéstrela. Si no, entregue un contraejemplo concreto (relaciones R, S, S' con sus tuplas).

Solución.

($\supseteq$ siempre se cumple) Sea $\bar{\tau} \in (R \div S) \cup (R \div S')$. Sin pérdida de generalidad, $\bar{\tau} \in R \div S$ (el otro caso es simétrico). Entonces $\bar{\tau} \in \pi_A(R)$ y $\forall \bar{\sigma} \in S.\ (\bar{\tau} \bowtie \bar{\sigma}) \in R$. Como $S \cap S' \subseteq S$, un "para todo" que vale sobre el conjunto más grande S vale en particular sobre el subconjunto $S \cap S'$:

$$\forall \bar{\sigma} \in (S \cap S').\ (\bar{\tau} \bowtie \bar{\sigma}) \in R.$$

Luego $\bar{\tau} \in R \div (S \cap S')$. ■

(La igualdad es falsa en general) Contraejemplo: sea $\mathsf{att}(R) = \{A, B\}$, $\mathsf{att}(S) = \mathsf{att}(S') = \{B\}$ (luego $\mathsf{att}(R \div \cdot) = \{A\}$),

$$R = \{(A{:}1, B{:}1), (A{:}2, B{:}1), (A{:}1, B{:}2), (A{:}3, B{:}3)\}, \qquad S = \{(B{:}1)\}, \qquad S' = \{(B{:}2)\}.$$

Como $B{:}1 \neq B{:}2$, se tiene $S \cap S' = \varnothing$. Calculemos cada lado:

- $\pi_A(R) = \{1, 2, 3\}$.
- $R \div S$: para cada $a \in \pi_A(R)$, ¿$(A{:}a, B{:}1) \in R$? Para $a = 1$: sí. Para $a = 2$: sí. Para $a = 3$: no (R sólo tiene $(3, 3)$). Luego $R \div S = \{1, 2\}$.
- $R \div S'$: ¿$(A{:}a, B{:}2) \in R$? Para $a = 1$: sí. Para $a = 2$: no. Para $a = 3$: no. Luego $R \div S' = \{1\}$.
- $(R \div S) \cup (R \div S') = \{1, 2\}$.
- $R \div (S \cap S') = R \div \varnothing$: como $S \cap S' = \varnothing$, la condición "$\forall \bar{\sigma} \in \varnothing \ldots$" se cumple **vacuamente** para todo $a \in \pi_A(R)$. Luego $R \div \varnothing = \pi_A(R) = \{1, 2, 3\}$.

Como $\{1, 2, 3\} \neq \{1, 2\}$, la igualdad es falsa (de hecho aquí la contención es estricta: $\{1, 2\} \subsetneq \{1, 2, 3\}$). El valor $a = 3$ pertenece a $R \div (S \cap S')$ sólo por la vacuidad del $\forall$ sobre el conjunto vacío, pero no cumple la condición para $b{=}1$ (no está en $R \div S$) ni para $b{=}2$ (no está en $R \div S'$): al intersectar S y S' estamos exigiendo *menos* condiciones, por lo que más candidatos pueden calificar. ■

3 Parte C: nulos y equivalencia de consultas

Use la tabla **Producto** y el resto del esquema de la Parte A (incluyendo los valores `NULL` de la instancia de ejemplo).

3.1 Ejercicio C.1

Sea $\theta :=$ (precio > 100), evaluado con lógica de 3 valores sobre **Producto** (que puede tener precio $=$ `NULL`). ¿Es cierto que

$$\sigma_\theta(\textbf{Producto}) \cup \sigma_{\neg\theta}(\textbf{Producto}) \;=\; \textbf{Producto}\ ?$$

Justifique con la instancia de ejemplo (piense en el producto sin precio fijado).
Recordatorio (extensión de σ a lógica de 3 valores). Para una relación R y una condición θ evaluada con lógica de 3 valores:

$$\sigma_\theta(R) \;:=\; \{\bar{\tau} \in R \mid \theta(\bar{\tau}) = \texttt{true}\}.$$

Es decir, σ_θ (igual que `WHERE` en SQL) conserva únicamente las tuplas donde θ vale exactamente `true`: una tupla con $\theta(\bar{\tau}) =$ `unknown` se descarta, exactamente igual que si $\theta(\bar{\tau}) =$ `false`.

Solución. Falsa en general (cuando hay nulos en el atributo de θ). El producto con sku $= 4$ (Taladro Eléctrico) tiene precio $=$ `NULL`. Evaluando con lógica de 3 valores:

$$\theta(\text{sku} = 4) = (\texttt{NULL} > 100) = \texttt{unknown}, \qquad \neg\theta(\text{sku} = 4) = \neg(\texttt{unknown}) = \texttt{unknown}.$$

Como σ (igual que `WHERE`) sólo conserva las filas donde la condición vale `true` (`unknown` se descarta igual que `false`), el sku 4 **no** pertenece a $\sigma_\theta(\textbf{Producto})$ ni a $\sigma_{\neg\theta}(\textbf{Producto})$: queda fuera de ambos lados. Concretamente, con la instancia:

$$\sigma_\theta(\textbf{Producto}) = \{3, 5, 6\} \qquad (\text{precios } 8990, 5990, 2990 > 100),$$

$$\sigma_{\neg\theta}(\textbf{Producto}) = \{1, 2\} \qquad (\text{precios } 50, 70 \not> 100),$$

$$\sigma_\theta(\textbf{Producto}) \cup \sigma_{\neg\theta}(\textbf{Producto}) = \{1, 2, 3, 5, 6\} \;\neq\; \{1, 2, 3, 4, 5, 6\} = \textbf{Producto}.$$

Falta exactamente el sku 4, el único con precio `NULL`. (Si **Producto** no tuviera ningún nulo en precio, la propiedad sí sería verdadera: con lógica de 2 valores, $\theta \vee \neg\theta$ es una tautología.)

3.2 Ejercicio C.2

Considere las siguientes dos consultas SQL, que buscan obtener el nombre de los clientes que **no** han comprado ningún producto de categoría "Tornillos":

```
-- Consulta A
SELECT c.nombre
FROM Cliente c
WHERE c.rutCliente NOT IN (
    SELECT p.rutCliente
    FROM Pedido p
    JOIN DetallePedido d ON d.idPedido = p.idPedido
    JOIN Producto pr ON pr.sku = d.sku
    WHERE pr.categoria = 'Tornillos'
);
```

```
-- Consulta B
SELECT c.nombre
FROM Cliente c
WHERE NOT EXISTS (
    SELECT 1
    FROM Pedido p
    JOIN DetallePedido d ON d.idPedido = p.idPedido
    JOIN Producto pr ON pr.sku = d.sku
    WHERE pr.categoria = 'Tornillos'
      AND p.rutCliente = c.rutCliente
);
```

¿Son equivalentes? Evalúe ambas sobre la instancia de ejemplo (note que el pedido 1005 es una venta de mostrador, con rutCliente = NULL) y justifique su respuesta usando lógica de 3 valores.

Solución. No son equivalentes.

Los productos de categoría "Tornillos" son los skus 1 y 2. Las filas de **DetallePedido** que los involucran son: $(1001, 1, 10)$, $(1002, 2, 5)$, $(1004, 1, 3)$ y $(1005, 2, 4)$, correspondientes a los pedidos 1001 (cliente 201), 1002 (cliente 202), 1004 (cliente 203) y 1005 (venta de mostrador, rutCliente = NULL).

(Consulta A) La subconsulta del `NOT IN` retorna $\{201, 202, 203, \texttt{NULL}\}$: incluye un `NULL` por el pedido de mostrador 1005. Para Diego Rojas (204), el único cliente que en verdad no ha comprado tornillos:

$$\begin{aligned} 204 \texttt{ NOT IN } \{201, 202, 203, \texttt{NULL}\} &= \neg(204{=}201 \vee 204{=}202 \vee 204{=}203 \vee 204{=}\texttt{NULL}) \\ &= \neg(\texttt{false} \vee \texttt{false} \vee \texttt{false} \vee \texttt{unknown}) \\ &= \neg(\texttt{unknown}) = \texttt{unknown}. \end{aligned}$$

Como `unknown` no satisface el `WHERE`, Diego queda **excluido**, a pesar de que en la instancia él es exactamente el cliente que no ha comprado ningún tornillo (la respuesta que la consulta debería devolver). Para 201, 202 y 203 el `IN` da `true` (hay coincidencia exacta), así que `NOT IN` da `false` y también quedan excluidos. La Consulta A devuelve el conjunto **vacío**.

(Consulta B) A diferencia de la Consulta A (que arma **un solo** conjunto $\{201, 202, 203, \texttt{NULL}\}$ compartido por todos los candidatos), aquí la subconsulta está correlacionada: para cada cliente c se recalcula un conjunto distinto Q_c (las filas de Tornillos con p.rutCliente $= c$). Para $c = 204$ (Diego), revisamos las 4 filas de Tornillos: en las de los pedidos 1001, 1002 y 1004 la comparación da 201=204, 202=204, 203=204, las tres `false`. En la de mostrador (1005), `NULL`=204 = `unknown`, tampoco `true`. Ninguna fila pasa el filtro, así que $Q_{204} = \varnothing$.

c	filas que pasan el filtro	Q_c	$\lvert Q_c \rvert$	`EXISTS`(Q_c)	resultado
201	fila de 1001	$\{1001\}$	1	`true`	excluido
202	fila de 1002	$\{1002\}$	1	`true`	excluido
203	fila de 1004	$\{1004\}$	1	`true`	excluido
204	ninguna	$\varnothing$	0	`false`	**incluido**

Como rutCliente $= c$ se evalúa **dentro** del `WHERE` de la subconsulta (no en una lista externa), la fila de mostrador nunca llega a formar parte de ningún Q_c: se filtra en el origen, en vez de sobrevivir para *"contaminar"* una comparación posterior (es decir, producir un `unknown` que arruine el resultado de esa comparación). Por lo tanto `EXISTS`(Q_{204}) = `false`, `NOT EXISTS`(Q_{204}) = `true`,

y Diego queda **incluido**. Los demás clientes sí tienen su propia fila en Q_c y quedan excluidos. La Consulta B devuelve {Diego Rojas}.

Como el resultado de A ($\varnothing$) difiere del de B ({Diego Rojas}), las consultas **no son equivalentes**: la presencia de un `NULL` en la subconsulta del `NOT IN` contamina la comparación para **todas** las filas candidatas, mientras que `NOT EXISTS` correlacionado no se ve afectado por ese nulo.