Práctico 2 | Ema

☑ Archive ☐

1. Resolver:
   a) Ejecute el siguiente código paso por paso en papel utilizando la definición: $map(2*)[3,0,4]$ (notar que $(2*) :: Int \longrightarrow Int$).
   b) Demuestre ejecutando paso a paso que $1 \in [3,1,3]$.

```
(1) (a)

Tenemos:
        map (2*) [3, 0, 4]
        = {def recursiva map}
        ((2*) 3) : map (2*) [0, 4]
        = {def recursiva map}
        ((2*) 3) : ((2*) 0) : map (2*) [4]
        = {def recursiva map}
        ((2*) 3) : ((2*) 0) : ((2*) 4) : map (2*) []
        = {def base map}
        ((2*) 3) : ((2*) 0) : ((2*) 4) : []
        = {def 2*}
        6 : 0 : 8 : []
        = {def :}
        [6, 0, 8]
Por ello, map (2*) [3, 0, 4] = [6, 0, 8]
```

```
(1) (b)

Tenemos:
        1 ∈ [3, 1, 3]
        = {def recursiva ∈}
        1 == 3 || 1 ∈ [1, 3]
        = {def recursiva ∈}
        1 == 3 || 1 == 1 || 1 ∈ [3]
        = {def recursiva ∈}
        1 == 3 || 1 == 1 || 1 == 3 || 1 ∈ []
        = {def base ∈}
        1 == 3 || 1 == 1 || 1 == 3 || False
        = {def ==}
        False || True || False || False
        = {def ||}
        True
por lo que se demuestra que 1 ∈ [3, 1, 3]
```

2. Considerar la siguiente definición de reverse por medio de ecuaciones:
   $reverse :: [a] \longrightarrow [a]$
   $reverse[] = []$
   $reverse(x : xs) = reverse\ xs + +[x]$
   se pide:
   $a)$ probar: $reverse\ [x] = [x]$
   $b)$ probar: $reverse\ (xs + +ys) = reverse\ ys + +reverse\ xs$
   $c)$ probar: reverse (reverse xs) = xs

```
(2) (a)

Tenemos que:
        reverse [x]
        = {def recursiva reverse}
        reverse [] ++ [x]
        = {def base reverse}
        [] ++ [x]
        = {def base ++}
        [x]
por lo que se demuestra que reverse [x] = [x]
```

```
(2) (b)

Lo vamos a demostrar con inducción sobre xs:
- CASO BASE: reverse([] ++ ys) = reverse ys ++ reverse []
        IZQ:    reverse([] ++ ys)
                = {def base ++}
                reverse ys

        DER:    reverse ys ++ reverse []
                = {def base reverse}
                reverse ys ++ []
                = {def base ++}
                reverse ys

        Luego, se cumple el caso base

- HI: reverse (xs ++ ys) = reverse ys ++ reverse xs

- PASO INDUCTIVO: reverse ((x:xs) ++ ys) = reverse ys ++ reverse (x:xs)
        IZQ:    reverse ((x:xs) ++ ys)
                = {def recursiva ++}
                reverse (x:(xs++ys))
                = {def recursiva reverse}
                reverse (xs ++ ys) ++ [x]

        DER:    reverse ys ++ reverse (x:xs)
                = {def recursiva reverse}
                reverse ys ++ reverse xs ++ [x]
                = {HI}
                reverse (xs ++ ys) ++ [x]

        Luego, se cumple el paso inductivo

Por ello, entonces, se demuestra que reverse (xs ++ ys) = reverse ys ++ reverse xs
```

```
(2) (c)

Lo vamos a demostrar con inducción sobre xs:
- CASO BASE: reverse (reverse []) = []
        IZQ:    reverse (reverse [])
                = {def base reverse}
                reverse []
                = {def base reverse}
                []

        Luego, se cumple el caso base

- HI: reverse (reverse xs) = xs

- PASO INDUCTIVO: reverse (reverse (x:xs)) = x:xs
        IZQ:    reverse (reverse (x:xs))
                = {def recursiva reverse}
                reverse (reverse xs ++ [x])
                = {prop (2)(b)}
                reverse [x] ++ reverse (reverse xs)
                = {HI}
                reverse [x] ++ xs
                = {prop (2)(a)}
                [x] ++ xs
                = {def ++}
                x:xs

        Luego, se cumple el paso inductivo.

Por ello, entonces, se demuestra que reverse (reverse xs) = xs
```

3. Probar las siguientes propiedades:
   a) $\sigma_P(\sigma_Q(r)) = \sigma_{P \wedge Q}(r)$
   b) $\Pi_{A1,\ldots,An}(\sigma_P(r)) = \sigma_P(\Pi_{A1,\ldots,An}(r))$ si P se refiere a lo mas a A1,...,An.

```
(3) (a)

Lo vamos a demostrar con inducción sobre r (xs):
- CASO BASE: σ_P (σ_Q ([])) = σ_{P∧Q} ([])
        IZQ:    σ_P (σ_Q ([]))
                = {def base σ}
                σ_P ([])
                = {def base σ}
                []

        DER:    σ_{P∧Q} ([])
                = {def base σ}
                []

        Luego, se cumple el caso base

- HI: σ_P (σ_Q (xs)) = σ_{P∧Q} (xs)

- PASO INDUCTIVO: σ_P (σ_Q (x:xs)) = σ_{P∧Q} (x:xs)
        IZQ:    σ_P (σ_Q (x:xs))
                = {def recursiva σ}
                σ_P (if Q(x) then x:σ_Q (xs) else σ_Q (xs))

                CASO DONDE Q(x) == True
                        σ_P (x:σ_Q (xs))
                        = {def recursiva σ}
                        if P(x) then x:σ_P (σ_Q (xs)) else σ_P (σ_Q (xs))

                        CASO DONDE P(x) == True
                                x:σ_P (σ_Q (xs))
                                = {HI}
                                x:σ_{P∧Q} (xs)
                        CASO DONDE P(x) == False
                                σ_P (σ_Q (xs))
                                = {HI}
                                σ_{P∧Q} (xs)
```

```
                CASO DONDE Q(x) = False
                        σ_P (σ_Q (xs))
                        = {HI}
                        σ_{P∧Q} (xs)

        DER:    σ_{P∧Q} (x:xs)
                = {def recursiva σ}
                if (P∧Q)(x) then x:σ_{P∧Q} (xs) else σ_{P∧Q} (xs)

                CASO DONDE Q(x) = True
                        CASO DONDE P(x) = True
                                x:σ_{P∧Q} (xs)
                        CASO DONDE P(x) = False
                                σ_{P∧Q} (xs)
                CASO DONDE Q(x) = False
                        σ_{P∧Q} (xs)

        Luego, el paso inductivo cumple para todas las posibles combinaciones de los
valores de P(x) y Q(x)

Por ello, entonces, se demuestra que σ_P (σ_Q (xs)) = σ_{P∧Q} (xs)
```

```
(3) (b)

Lo vamos a demostrar con inducción sobre r (xs):
- CASO BASE: ∏_{A1,..,An} (σ_P ([])) = σ_P (∏_{A1,..,An} ([]))
        IZQ:    ∏_{A1,..,An} (σ_P ([]))
                = {def base σ}
                ∏_{A1,..,An} []
                = {def base ∏}
                []

        DER:    σ_P (∏_{A1,..,An} ([]))
                = {def base ∏}
                σ_P ([])
                = {def base σ}
                []

        Luego, se cumple el caso base

- HI: ∏_{A1,..,An} (σ_P (xs)) = σ_P (∏_{A1,..,An} (xs))

- PASO INDUCTIVO: ∏_{A1,..,An} (σ_P (x:xs)) = σ_P (∏_{A1,..,An} (x:xs))
        IZQ:    ∏_{A1,..,An} (σ_P (x:xs))
                = {def recursiva σ}
                ∏_{A1,..,An} (if P(x) then x:σ_P (xs) else σ_P (xs))

                CASO DONDE P(x) == True
                        ∏_{A1,..,An} (x:σ_P (xs))
                        = {def recursiva de ∏}
                        (A1 x, .., An x) : ∏_{A1,..,An} (σ_P (xs))

                CASO DONDE P(x) == False
                        ∏_{A1,..,An} (σ_P (xs))

        DER:    σ_P (∏_{A1,..,An} (x:xs))
                = {def recursiva ∏}
                σ_P ((A1 x, .., An x) : ∏_{A1,..,An} (xs))
                = {def recursiva σ}
                if P((A1 x, .., An x)) then (A1 x, .., An x) : σ_P (∏_{A1,..,An} (xs))
else σ_P (∏_{A1,..,An} (xs))
```

```
                CASO DONDE P((A1 x, .., An x)) = True
                       (A1 x, .., An x) : σ_P (∏_{A1,..,An} (xs))
                       = {HI}
                       (A1 x, .., An x) : ∏_{A1,..,An} (σ_P (xs))

                CASO DONDE P((A1 x, .., An x)) = False
                       σ_P (∏_{A1,..,An} (xs))
                       = {HI}
                       ∏_{A1,..,An} (σ_P (xs))

        Luego, como por condición P se aplica a lo más a A1, .., An, entonces los
valores de P(x) y P(A1, .., An) tienen una relación One-to-One, por lo que el paso
inductivo se cumple.

Motivo de ello, entonces, se demuestra que ∏_{A1,..,An} (σ_P (xs)) = σ_P (∏_{A1,..,An}
(xs))
```

4. Demostrar las siguientes ecuaciones:
   a) $anexar\ []\ x\ q = q$
   b) $anexar\ (y : s)\ x\ q = (x; y)\ :\ (anexar\ s\ x\ q)$
   c) $s \times [] = []$

```
(4) (intro)

En el apunte se define anexar así:
        anexar s x q = (map (\t → (x ; t)) s) ++ q
donde
        map f [] = []
        map f (x:xs) = (f x) : (map f xs)
```

```
(4) (a)
Tenemos que:
        anexar [] x q = q
        = {def anexar}
        (map (\t → (x; t)) []) ++ q
        = {def base map}
        [] ++ q
        = {def ++}
        q
por lo que se demuestra la propiedad
```

```
(4) (b)
Tenemos que:
        anexar (y:s) x q
        = {def anexar}
        (map (\t → (x;t)) (y:s)) ++ q
        = {def recursiva map}
        ((((\t → (x;t) y) : (map (\t → (x;t)) s)) ++ q
        = {aplicar func}
        ((x;y) : (map (\t → (x;t)) s)) ++ q
        = {def : y ++}
        (x;y) : (map (\t → (x;t)) s ++ q)
        = {def anexar}
        (x;y) : anexar s x q
por lo que se demuestra la propiedad
```

```
(4) (c)
Vamos a demostrarlo con inducción sobre s (xs):
- CASO BASE: [] × [] = []
       IZQ:    [] × []
               = {def base ×}
               []

       por lo que cumple el caso base

- HI: xs × [] = []

- PASO INDUCTIVO: (x:xs) × [] = []
       IZQ:    (x:xs) × []
               = {def recursiva ×}
               anexar [] x (xs × [])
               = {HI}
               anexar [] x []
               = {Prop A}
               []
        por lo que se cumple el paso inductivo.

Luego, entonces, se demuestra la propiedad de que s × [] = []
```

5. Sean $R = (A, B, C)$, $S = (B, C, D)$ y $T = (C, D)$ esquemas de relación; y sean $r(R)$, $s(S)$ y $t(T)$ definidas como:

r	A	B	C
	a	b	c
	d	b	c
	b	b	f
	c	a	d

s	B	C	D
	b	c	d
	b	c	e
	a	d	b

t	C	D
	c	d
	c	e

Calcular las siguientes expresiones del álgebra relacional:

a) $\pi_{B,C}(r) - \pi_{B,C}(s)$.
b) $\sigma_{C=c \vee B \neq b}(r)$.
c) $r \bowtie s$.
d) $r \times s$.

```
(5) (a)
Veamos cada parte por separado:
        ∏_{B, C} (r)
                ———     Nombre
              B   C     Esquema
              b   c     Comienzo de cuerpo
              b   c
              b   f
              a   d

        ∏_{B, C} (s)
                ———     Nombre
              B   C     Esquema
              b   c     Comienzo de cuerpo
              b   c
              a   d

        ∏_{B, C} (r) - ∏_{B, C} (s)
                ———     Nombre
              B   C     Esquema
              b   f     Comienzo de cuerpo
```

(5) (b)

$\sigma_{C=c \vee B \neq b}(r)$

A	B	C	Nombre
a	b	c	Esquema
d	b	c	Comienzo de cuerpo
c	a	d	

(5) (c)

$r \bowtie s$

A	B	C	D	Nombre
a	b	c	d	Esquema
a	b	c	e	Comienzo de cuerpo
d	b	c	d	
d	b	c	e	
c	a	d	b	

(5) (d)

$r \times s$

A	Br	Cr	Bs	Cs	D	Nombre
a	b	c	b	c	d	Esquema
a	b	c	b	c	e	Comienzo de cuerpo
a	b	c	a	d	b	
d	b	c	b	c	d	
d	b	c	b	c	e	
d	b	c	a	d	b	
b	b	f	b	c	d	
b	b	f	b	c	e	
b	b	f	a	d	b	
c	a	d	b	c	d	
c	a	d	b	c	e	
c	a	d	a	d	b	

6. Probar las siguientes propiedades:
   a) $\Pi_{f1,\ldots,fn}(r + +s) = \Pi_{f1,\ldots,fn}(r) + +\Pi_{f1,\ldots,fn}(s)$
   b) $\sigma_P(r + +s) = \sigma_P(r) + +\sigma_P(s)$

```
(6) (a)
Lo vamos a demostrar por inducción sobre r (xs):
- CASO BASE: ∏_{f1,..,fn} ([]++s) = ∏_{f1,..,fn} ([]) ++ ∏_{f1,..,fn} (s)
        IZQ:    ∏_{f1,..,fn} ([]++s)
                = {def base ++}
                ∏_{f1,..,fn} (s)

        DER:    ∏_{f1,..,fn} ([]) ++ ∏_{f1,..,fn} (s)
                = {def base ∏}
                [] ++ ∏_{f1,..,fn} (s)
                = {def base ++}
                ∏_{f1,..,fn} (s)

        por lo que se cumple con el caso base

- HI: ∏_{f1,..,fn} (xs++s) = ∏_{f1,..,fn} (xs) ++ ∏_{f1,..,fn} (s)

- PASO INDUCTIVO: ∏_{f1,..,fn} ((x:xs) ++ s) = ∏_{f1,..,fn} ((x:xs)) ++ ∏_{f1,..,fn} (s)
        IZQ:    ∏_{f1,..,fn} ((x:xs) ++ s)
                = {def recursiva ++}
                ∏_{f1,..,fn} (x : (xs ++ s))
                = {def recursiva ∏}
                (f1 x,.., fn x) : ∏_{f1,..,fn} (xs ++ s)

        DER:    ∏_{f1,..,fn} ((x:xs)) ++ ∏_{f1,..,fn} (s)
                = {def recursiva ∏}
                ((f1 x,.., fn x) : ∏_{f1,..,fn} (xs)) ++ ∏_{f1,..,fn} (s)
                = {def recustiva ++}
                (f1 x,.., fn x) : (∏_{f1,..,fn} (xs) ++ ∏_{f1,..,fn} (s))
                = {HI}
                (f1 x,.., fn x) : ∏_{f1,..,fn} (xs++s)

        por lo que se cumple el paso inductivo

Por ello, entonces, se demuestra que ∏_{f1,..,fn} (xs++s) = ∏_{f1,..,fn} (xs) ++ ∏_{f1,..,fn} (s)
```

```
(6) (b)
Lo vamos a demostrar por inducción sobre r (xs):
- CASO BASE: σ_P ([] ++ s) = σ_P ([]) ++ σ_P (s)
        IZQ:    σ_P ([] ++ s)
                = {def base ++}
                σ_P (s)

        DER:    σ_P ([]) ++ σ_P (s)
                = {def base σ}
                [] ++ σ_P (s)
                = {def base ++}
                σ_P (s)

        por lo que se cumple el caso base

- HI: σ_P (xs ++ s) = σ_P (xs) ++ σ_P (s)

- PASO INDUCTIVO: σ_P ((x:xs) ++ s) = σ_P ((x:xs)) ++ σ_P (s)
        IZQ:    σ_P ((x:xs) ++ s)
                = {def recursiva ++}
                σ_P (x:(xs ++ s))
                = {def recursiva σ}
                if P(x) then x : σ_P (xs ++ s) else σ_P (xs ++ s)

        DER:    σ_P ((x:xs)) ++ σ_P (s)
                = {def recursiva σ}
                (if P(x) then x : σ_P (xs) else σ_P (xs)) ++ σ_P (s)
                = {prop trivial}
                if P(x) then ((x : σ_P (xs)) ++ σ_P (s)) else (σ_P (xs) ++ σ_P (s))
                = {def recursiva ++}
                if P(x) then (x : (σ_P (xs) ++ σ_P (s))) else (σ_P (xs) ++ σ_P (s))
                = {HI}
                if P(x) then (x : σ_P (xs ++ s)) else σ_P (xs ++ s)

        por lo que se cumple el paso inductivo

Por ello, entonces, se demuestra que σ_P (xs ++ s) = σ_P (xs) ++ σ_P (s)
```

7. Considere la siguiente base de datos para una pizzería:

```
cliente(cid, cnombre, telefono, dirección, edad)
pizzas(zid, znombre, tamaño, precio)
pedido(cid, zid, phora, paño, pmes, pdía, cantidad)
```

¿Cuáles deberían ser las claves primarias? Expresar las siguientes consultas en el álgebra relacional:

a) Obtener las pizzas (código y nombre) que se ofrecen.
b) Obtener los códigos de las pizzas que fueron pedidas.
c) Obtener las pizzas (código y nombre) que fueron pedidas.
d) Obtener los códigos de las pizzas que fueron pedidas por clientes de menos de 25 años.
e) Obtener las pizzas que fueron pedidas por clientes de menos de 25 años.
f) Obtener las pizzas pedidas durante el último mes de agosto.
g) Obtener los clientes (código, nombre y dirección) que pidieron alguna pizza que no sea de nombre "muzza".
h) Obtener los clientes (código, nombre y dirección) que pidieron al menos dos códigos de pizzas diferentes.
i) (requiere agregaciones) Obtener el número total de pizzas de cada código que fueron pedidas durante el mes de agosto.
j) (requiere agregaciones) Obtener la pizza más solicitada de agosto.

```
(7) Tenemos la BD

cliente(cid, cnombre, teléfono, dirección, edad)
        con PK cid

pizzas(zid, znombre, tamaño, precio)
        con PK zid

pedido(cid, zid, phora, paño, pmes, pdía, cantidad)
        FK cid REFERENCES cliente(cid)
        FK zid REFERENCES pizzas(zid)
```

```
(7) (a) ∏_{zid, znombre} (pizzas)
```

```
(7) (b) v(∏_{zid} (pedido))
```

```
(7) (c) v(∏_{zid, znombre} (pizzas ⋈ pedido))
```

```
(7) (d) let cliente_menos_25 = σ_{edad < 25} (cliente)
        v(∏_{zid} (cliente_menos_25 ⋈ pedido))
```

```
(7) (e) let cliente_menos_25 = σ_{edad < 25} (cliente)
        let info_completa = (cliente_menos_25 ⋈ pedido) ⋈ pizzas
        v(∏_{zid, znombre} (info_completa))
```

```
(7) (f) let pizzas_agosto = σ_{pmes = "Agosto"} (pizzas)
        let info_completa_pizzas_agosto = pizzas_agosto ⋈ pizzas
        v(∏_{zid, znombre} (info_completa_pizzas_agosto))
```

```
(7) (g) let pedidos_de_muzza = σ_{znombre = "muzza"} (pizzas ⋈ pedido)
        let clientes_que_pidieron_muzza = ∏_{cid, cnombre, dirección} (cliente ⋈ pedidos_de_muzza)
        ∏_{cid, cnombre, dirección} (cliente) / clientes_que_pidieron_muzza
```

```
(7) (h) let info_cliente_pizza = v(∏_{cid, zid} (pedido))
        let info_cliente_cnt_pizzas_distintas = ρ_(cid, cnt) ({cid}_γ_{count zid} (info_cliente_pizza))
        let clientes_que_quiero = ∏_{cid} (σ_{cnt > 1} (info_cliente_cnt_pizzas_distintas))
        let info_completa_cliente_que_quiero = cliente ⋈ cliente_que_quiero
        ∏_{cid, cnombre, dirección} (info_completa_cliente_que_quiero)
```

```
(7) (i) let pedidos_agosto = σ_{pmes = "Agosto"} (pedido)
        {zid}_γ{count cid} (pedidos_agosto)
```

```
(7) (j) let pedidos_agosto = σ_{pmes = "Agosto"} (pedido)
        let pizza_por_cnt = ρ_(zid, cnt) ({zid}_γ_{count cid} (pedidos_agosto))
        (O^{>} pizza_por_cnt)[0].zid
```

8. Demostrar:
   - $a)\ r \setminus [] = r$
   - $b)\ [] \setminus r = []$
   - $c)\ (x : r) \setminus s = if x \notin s\ then\ x : (r \setminus s)\ else\ r \setminus s$
   - $d)\ x \in (r \setminus s) = (x \in r)\ \&\&\ (x \notin s)$
   - $e)\ r \cap [] = []$
   - $f)\ [] \cap r = []$
   - $g)\ (x : r) \cap s = \ if\ x \in s\ then\ x : (r \cap s)\ else\ r \cap s$

```
(8) (a)
La definición de resta considerada es:
        r \ s = σ_{\t → t ∉ s} (r)

Por ello, entonces, podemos notar que tenemos que
        r \ []
        = {def \}
        σ_{\t → t ∉ []} (r)
        = {def base ∉}
        σ_{\t → True} (r)
        = {dado que en la función no interviene t}
        σ_True (r)
        = {prop σ}
        r
por lo que se demuestra la propiedad.
```

```
(8) (b)
En este caso tenemos
        [] \ r
        = {def \}
        σ_{\t → t ∉ r} ([])
        = {def base σ}
        []
por lo que se demuestra la propiedad.
```

```
(8) (c)
Tenemos que:
        (x : r) \ s
        = {def \}
        σ_{\t → t ∉ s} (x : r)
        = {def recursiva σ}
        if x ∉ s then x : σ_{\t → t ∉ s} r else σ_{\t → t ∉ s} r
        = {def \}
        if x ∉ s then x : (r \ s) else (r \ s)
por lo que se demuestra la propiedad
```

```
(8) (d)
Lo vamos a demostrar por inducción en r (ys):
- CASO BASE: x ∈ ([] \ s) = (x ∈ []) ∧ (x ∉ s)
        IZQ:    x ∈ ([] \ s)
                = {prop A}
                x ∈ []
                = {def base ∈}
                False

        DER:    (x ∈ []) ∧ (x ∉ s)
                = {def base ∈}
                False ∧ (x ∉ s)
                = {prop ∧}
                False

        por lo que se cumple con el caso base.

- HI: x ∈ (ys \ s) = (x ∈ ys) ∧ (x ∉ s)

- PASO INDUCTIVO: x ∈ ((y:ys) \ s) = (x ∈ (y:ys)) ∧ (x ∉ s)
        IZQ:    x ∈ ((y:ys) \ s)
                = {prop C}
                x ∈ (if y ∉ s then y : (ys \ s) else (ys \ s))

                CASO CON y ∉ s = True
                        x ∈ (y : (ys \ s))
                        = {def recursiva ∈}
                        x == y ∨ x ∈ (ys \ s)

                CASO CON y ∉ s = False
                        x ∈ (ys \ s)
```

```
DER:    (x ∈ (y:ys)) ∧ (x ∉ s)
        = {def recursiva ∈}
        (x == y ∨ x ∈ ys) ∧ (x ∉ s)
        = {distributiva ∧ sobre ∨}
        (x == y ∧ x ∉ s) ∨ (x ∈ ys ∧ x ∉ s)
        = {HI}
        (x == y ∧ x ∉ s) ∨ x ∈ (ys \ s)

        CASO CON y ∉ s = True
                = {si y ∉ s, entonces x = y ⟹ x ∉ s. Luego, se puede sacar la "redundancia"}
                x == y ∨ x ∈ (ys \ s)

        CASO CON y ∉ s = False
                = {si y ∈ s, entonces x = y ⟹ x ∈ s. Luego, se llegaría a una contradicción en las condiciones del AND. Luego, se puede considerar False}
                False ∨ x ∈ (ys \ s)
                = {prop ∨}
                x ∈ (ys \ s)

    Luego, entonces, se cumple con el paso inductivo para los dos casos de la condición y ∉ s

Por ello, entonces, se demuestra que x ∈ (ys \ s) = (x ∈ ys) ∧ (x ∉ s)
```

```
(8) (e)
Por definición, sabemos que la intersección se considera como:
        r ∩ s = σ_{\t → t ∈ s} (r)

Dado esto, tenemos que:
        r ∩ []
        = {def ∩}
        σ_{\t → t ∈ []} (r)
        = {def base ∈}
        σ_{\t → False} (r)
        = {dado que en la func. no interviene t}
        σ_False (r)
        = {prop σ}
        []
por lo que se demuestra la propiedad
```

```
(8) (f)
Tenemos que:
        [] ∩ r
        = {def ∩}
        σ_{\t → t ∈ r} ([])
        = {def base σ}
        []
por lo que se demuestra la propiedad
```

```
(8) (g)
Tenemos que
        (x : r) ∩ s
        = {def ∩}
        σ_{\t → t ∈ s} (x : r)
        = {def recursiva σ}
        if x ∈ s then x : (σ_{\t → t ∈ s} r) else (σ_{\t → t ∈ s} r)
        = {def ∩}
        if x ∈ s then x : (r ∩ s) else (r ∩ s)
por lo que se demuestra la propiedad
```

9. Dado el siguiente modelo relacional:

   `cliente(cId,cNombre,calle,ciudad)`
   `sucursal(sId,sNombre,ciudad,fondos)`
   `depósito(cId,sId,nCuenta,dMonto,dfecha,dhora)`
   `préstamo(cId,sId,nPrestamo,pMonto,pfecha,phora)`

   ¿Que información contiene cada una de las relaciones? ¿Cuáles deberían ser las claves primarias? Expresar las siguientes consultas en el álgebra de tablas:
   a) Nombre de clientes que hayan depositado y tomado préstamos en la misma sucursal.
   b) Nombre de clientes que depositaron alguna vez en su propia ciudad.
   c) Nombre de clientes que sólo depositan en ciudades donde no viven (según la relación `cliente`).

```
(9) Las relaciones que tenemos en nuestra BD Relacional son:

cliente(cId, cNombre, calle, ciudad)
        con PK cId

sucursal(sId, sNombre, ciudad, fondos)
        con PK sId

depósito(cId, sId, nCuenta, dMonto, dfecha, dhora)
        FK cId REFERENCES cliente(cId)
        FK sId REFERENCES sucursal(sId)

préstamo(cId, sId, nPrestamo, pMonto, pfecha, phora)
        FK cId REFERENCES cliente(cId)
        FK sId REFERENCES sucursal(sId)
```

```
(9) (a) let info_deposito = v(∏_{cId, sId} (depósito))
        let info_préstamo = v(∏_{cId, sId} (préstamo))
        let clientes_buscados = v(∏_{cId} (info_deposito ∩ info_préstamo))
        ∏_{cNombre} (cliente ⋈ clientes_buscados)
```

```
(9) (b) let info_deposito = v(∏_{cId, sId} (depósito))
        let info_deposito_completa = (info_deposito ⋈ cliente) ⋈ sucursal
        let clientes_buscados = σ_{cNombre == sNombre} (info_deposito_completa)
        ∏_{cNombre} (clientes_buscados)
```

```
(9) (c) let info_deposito = v(∏_{cId, sId} (depósito))
        let info_deposito_completa = (info_deposito ⋈ cliente) ⋈ sucursal
        let clientes_buscados = σ_{cNombre == sNombre} (info_deposito_completa)
        ∏_{cNombre} (cliente) \ (∏_{cNombre} (clientes_buscados))
```

10. Dado el siguiente modelo relacional:

    `proveedor(pID,pNombre,ciudad)`
    `trabajo(tID,tNombre,ciudad)`
    `máquina(mID,mNombre,color,peso)`
    `ptm(pID,tID,mID)`

    La idea de la relación `ptm` es que una tupla (p, t, m) de ella nos dice que el proveedor p provee una máquina m para hacer un trabajo t. Encontrar las claves primarias. Expresar las siguientes consultas en el álgebra de tablas:

    a) El color de las máquinas suplidas por el proveedor p_1.
    b) Los valores `pID` de proveedores que suplen al menos una máquina roja al trabajo t_1.
    c) Los valores `tID` de trabajos suplidos por al menos un proveedor de otra ciudad.
    d) Los valores `tID` de trabajos suplidos enteramente por el proveedor p_1.

```
(10) Tenemos el siguiente modelo relacional:

proveedor(pID, pNombre, ciudad)
        con PK pID

trabajo(tID, tNombre, ciudad)
        con PK tID

máquina(mID, mNombre, color, peso)
        con PK mID

ptm(pID, tID, mID)
        FK pID REFERENCES proveedor(pID)
        FK tID REFERENCES trabajo(tID)
        FK mID REFERENCES máquina(mID)
```

```
(10) (a)        let relacion_p1_maq = v(∏_{pID, mID} (σ_{pID = p1} (ptm)))
                let info_completa_maq_buscadas = relacion_p1_maq ⋈ máquina
                v(∏_{color} (info_completa_maq_buscadas))
```

```
(10) (b)        let relacion_con_info_maq = ptm ⋈ máquina
                let maquina_roja_trabajo_t1 = σ_{color = "rojo" ∧ tID = t1} (relacion_con_info_maq)
                v(∏_{pID} (maquina_roja_trabajo_t1))
```

```
(10) (c)        let trabajo2 = ρ(ciudad ← tCiudad)
                let info_completa = (ptm ⋈ proveedor) ⋈ trabajo
                let info_completa_distintas_ciudades = σ_{ciudad ≠ tCiudad} (info_completa)
                v(∏_{pID} (info_completa_distintas_ciudades))
```

```
(10) (d)        let trabajos_de_p1 = v(∏_{tID} (σ_{pID = p1} (ptm)))
                let trabajos_no_de_p1 = v(∏_{tID} (σ_{pID ≠ p1} (ptm)))
                trabajos_de_p1 \ trabajos_no_de_p1
```

11. Dada la base de datos universitaria:

```
aula(edificio, aulaNro, capacidad)
facultad(nombreFacultad, edificio, presupuesto)
curso(idCurso, título, nombreFacultad, créditos)
profe(ID, nombre, nombreFacultad, salario)
actividad(idCurso, idAct, semestre, año, edificio, aulaNro, idDurClase)
enseña(ID, idCurso, idAct, semestre, año)
estudiante(ID, nombre, nombreFacultad, total de créditos)
toma(ID, idCurso, idAct, semestre, año, nota)
supervisore(IDe, IDp)
horarios(idDurClase, día, horaInicio, horaFin)
correlativa(idCurso, idPre-requisito)
```

escribir expresiones de consulta en el álgebra de tablas que permitan:

a) Encontrar los ID de todos los estudiantes a los que les enseñó un profesor llamado Einstein.
b) Encontrar el salario más alto de todos los profesores.
c) Encontrar todos los profesores que ganan el salario más alto (puede haber más de uno con el mismo salario).
d) Encontrar la matrícula de cada actividad que fue ofrecida en otoño de 2009.
e) Encontrar la matrícula máxima, a lo largo de todas las actividades, en otoño de 2009.
f) Encontrar las actividades que tuvieron la máxima matrícula en otoño de 2009.
g) Encontrar los nombres de todos los estudiantes que tomaron al menos un curso de Ciencias de la Computación.
h) Encontrar los ID y los nombres de todos los estudiantes que no tomaron ningún curso ofrecido antes de la primavera de 2009.
i) Para cada facultad encontrar el salario máximo de los profesores en esa facultad. Puedes asumir que cada facultad tiene al menos un profesor.
j) Encontrar el menor, entre todas las facultades, de los salarios máximos por facultad computado por la consulta anterior.

```
(11) (a)        let id_profes_Einstein = ρ_(ID ← pID) (∏_{ID} (σ_{nombre == "Einstein"} (profe)))
                let id_cursos_de_Einstein = v(∏_{idCurso} (enseña ⋈ id_profes_Einstein))
                v(∏_{ID} (estudiante ⋈ id_cursos_de_Einstein))
```

```
(11) (b)        γ_{max salario} (profe)
```

```
(11) (c)        let maxi_salario = ρ_(salario) (γ_{max salario} (profe))
                let profes_con_maxi_salario = profe ⋈ maxi_salario
                v(∏_{ID} (profes_con_maxi_salario))
```

```
(11) (d)    let id_actividades_otoño_2009 = ∏_{idAct} (σ_{año == 2009 ∧ semestre = 2} (actividad))
            let id_estudiantes_de_actividad = ∏_{idAct, ID} (id_actividades_otoño_2009 ⋈ toma)
            {IdAct}_γ_{count ID} (id_estudiantes_de_actividad)
```

```
(11) (e)    let id_actividades_otoño_2009 = ∏_{idAct} (σ_{año == 2009 ∧ semestre = 1} (actividad))
            let id_estudiantes_de_actividad = ∏_{idAct, ID} (id_actividades_otoño_2009 ⋈ toma)
            let actividad_con_cnt = ρ_(idAct, cnt) ({idAct}_γ_{count ID} (id_estudiantes_de_actividad))
            γ_{max cnt} (actividad_con_cnt)
```

```
(11) (f)    let id_actividades_otoño_2009 = ∏_{idAct} (σ_{año == 2009 ∧ semestre = 2} (actividad))
            let id_estudiantes_de_actividad = ∏_{idAct, ID} (id_actividades_otoño_2009 ⋈ toma)
            let actividad_con_cnt = ρ_(idAct, cnt) ({idAct}_γ_{count ID} (id_estudiantes_de_actividad))
            let maxi_cnt_de_actividad = γ_{max cnt} (actividad_con_cnt)
            ∏_{idAct} (σ_{cnt == maxi_cnt_de_actividad[0]} (actividad_con_cnt))
```

```
(11) (g)    let id_cursos_compu = ∏_{idCurso} (σ_{título == "Ciencias de la Computación"} (curso))
            let id_estudiantes_de_compu = ν(∏_{ID} (toma ⋈ id_cursos_compu))
            ∏_{nombre} (estudiante ⋈ id_estudiantes_de_compu)
```

12. Considerar la siguiente base de datos relacional:

    `empleado(pId,nombre-empleado,calle,ciudad)`
    `trabaja(pId,sId,sueldo)`
    `sucursal(sId,nombre-empresa,ciudad)`
    `jefe(pIdempleado,pIdjefe)`

    Una misma empresa puede tener varias sucursales (incluso en una misma ciudad), cada una tiene un código `sId` diferente, pero comparten el nombre de la empresa. Dar una expresión del álgebra de tablas para cada una de las consultas siguientes:

    a) Averiguar el nombre, la calle y la ciudad de residencia de todos los empleados que trabajan para la sucursal identificada por `sId` = 12345 y ganan más de 20000 pesos anuales.
    b) Averiguar el nombre de todos los empleados de esta base de datos que viven en la misma ciudad que la sucursal para la que trabajan.
    c) Averiguar el nombre de todos los empleados que viven en la misma ciudad y en la misma calle que sus jefes.
    d) Averiguar el nombre de todos los empleados de esta base de datos que no trabajan para la sucursal identificada por `sId` = 12345.
    e) Averiguar el nombre de todos los empleados que ganan más que cualquier empleado de la empresa Banco Hipotecario.
    f) Averiguar el nombre de la compañía con mayor número de empleados.
    g) Averiguar el nombre de la compañía con la nómina más reducida (suma de sueldos de sus empleados).
    h) Averiguar los nombres de las compañías cuyos empleados ganen en promedio un sueldo más elevado que el sueldo medio del Banco Hipotecario.