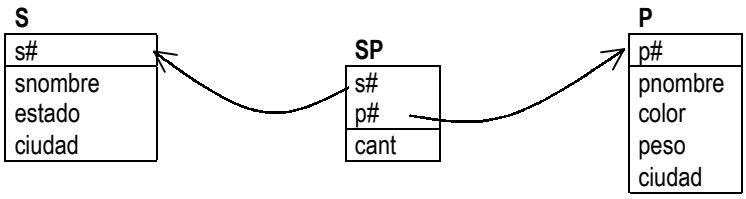
Curso 2001/2002

Ejercicios de Álgebra Relacional y Cálculo Relacional

APARTADO 1



Álgebra Relacional

1.1) Obtener el nombre de los proveedores que suministran la pieza con el código P2.

```
((SP WHERE p#='P2') JOIN S)[s#,snombre]
```

Podemos escribir una sentencia equivalente ordenando las operaciones como hace la sentencia SELECT de SQL:

```
((SP JOIN S) WHERE p#='P2')[s#,snombre]
```

Lo que da lugar a la siguiente sentencia SQL:

```
SELECT DISTINCT S.s#, S.snombre
FROM   SP, S
WHERE  SP.s#=S.s# AND p#='P2';
```

1.2) Obtener el nombre de los proveedores que suministran por lo menos una pieza roja.

```
((SP JOIN P WHERE color='rojo')[s#] JOIN S)[s#,snombre]
```

Podemos escribir una sentencia equivalente ordenando las operaciones como hace la sentencia SELECT de SQL:

```
((SP JOIN P JOIN (S[s#,snombre])) WHERE color='rojo')[s#,snombre]
```

Lo que da lugar a la siguiente sentencia SQL:

```
SELECT DISTINCT S.s#, S.snombre
FROM   SP, P, S
WHERE  SP.p#=P.p# AND SP.s#=S.s#
AND    P.color='rojo';
```

1.3) Obtener el nombre de las piezas de color rojo suministradas por los proveedores de la ciudad de Londres.

```
T1 := (S WHERE ciudad='Londres')[s#]
RDO := (T1 JOIN SP JOIN (P WHERE color='rojo'))[p#,pnombre]
```

La sentencia SQL que responde a esta consulta es la siguiente:

```
SELECT DISTINCT P.p#, P.pnombre
FROM   SP, S, P
WHERE  SP.s#=S.s# AND SP.p#=P.p#
AND    S.ciudad='Londres' AND P.color='rojo';
```

1.4) Obtener el código de los proveedores que suministran alguna de las piezas que suministra el proveedor con el código S2.

```
((SP WHERE s#='S2')[p#] JOIN SP) [s#]
```

La sentencia SQL que responde a esta consulta es la siguiente:

```
SELECT DISTINCT SPY.s#
FROM   SP SPX, SP SPY
WHERE  SPX.s#='S2' AND SPX.p#=SPY.p#;
```

1.5) Obtener los datos de los envíos de más de 100 unidades, mostrando también el nombre del proveedor y el de la pieza.

```
T1 := ((SP WHERE cant>100) JOIN P)[s#,p#,cant,pnombre]
RDO := (T1 JOIN S)[ s#,snombre,p#,pnombre,cant]
```

La sentencia SQL que responde a esta consulta es la siguiente:

```
SELECT SP.s#,S.snombre,SP.p#,P.pnombre,SP.cant
FROM   SP,S,P
WHERE  SP.s#=S.s# AND SP.p#=P.p#
AND    SP.cant>100;
```

1.6) Obtener el nombre de los proveedores que suministran todas las piezas.

```
((SP[s#,p#] DIVIDEBY P[p#]) JOIN S)[s#,snombre]
```

Que es equivalente a la siguiente expresión, porque la división no es una operación primitiva:

```
T1 := SP[s#] MINUS ((SP[s#] TIMES P[p#]) MINUS SP[s#,p#])[s#]
RDO := (T1 JOIN S)[s#,snombre]
```

Otro modo de resolver esta consulta es contando piezas. Ya que SP.P# es clave ajena a P, por la regla de integridad referencial, todos los valores que aparecen en SP.P# son valores que aparecen en P.P#. Por lo tanto, si un proveedor suministra tantas piezas como hay en P, podemos asegurar que las suministra todas.

```
T1 := SUMMARIZE P GROUPBY() ADD COUNT(*) AS num_piezas
T2 := SUMMARIZE SP GROUPBY(s#) ADD COUNT(*) AS num_piezas_s#
T3 := ((T1 TIMES T2) WHERE num_piezas = num_piezas_s#)[s#]
RDO := (T3 JOIN S)[s#,snombre]
```

Que en SQL corresponde a la siguiente sentencia:

```
SELECT SP.s#,S.snombre
FROM   SP,S
WHERE  SP.s#=S.s#
GROUP BY SP.s#,S.snombre
HAVING COUNT(*) = (SELECT COUNT(*) FROM P);
```

1.7) Obtener el código de los proveedores que suministran, al menos, todas las piezas suministradas por el proveedor con código S2.

```
((SP[s#,p#] DIVIDEBY (SP WHERE s#='S2')[p#]) JOIN S)[s#,snombre]
```

De nuevo, esta consulta se puede resolver contando. Para cada proveedor obtenemos los envíos que realiza de las piezas que envía S2 y contamos. Si realiza tantos de estos envíos como S2 es porque realiza los mismos.

```
T1 := (SP WHERE S#='S2')[p#]
T2 := SUMMARIZE T1 GROUPBY() ADD COUNT(*) AS num_piezas_s2
T3 := SP JOIN T1
T4 := SUMMARIZE T3 GROUPBY(s#) ADD COUNT(*) AS num_piezas_s2_s#
T5 := ((T2 TIMES T4) WHERE num_piezas_s2 = num_piezas_s2_s#)[s#]
RDO := (T5 JOIN S)[s#,snombre]
```

Que en SQL corresponde a la siguiente sentencia:

```
SELECT SP1.s#,S.snombre
FROM   SP SP1,S, SP SP2
WHERE  SP1.s#=S.s# AND SP2.s#='S2' AND SP1.p#=SP2.p#
GROUP BY SP1.s#,S.snombre
HAVING COUNT(*) = (SELECT COUNT(*)
                   FROM SP WHERE s#='S2');
```

1.8) Obtener el nombre de los proveedores que no suministran la pieza con el código P2.

```
((S[s#] MINUS (SP WHERE p#='P2')[s#]) JOIN S)[s#,snombre]
```

Que es equivalente a la siguiente expresión:

```
(S[s#,snombre] MINUS ((SP JOIN S) WHERE p#='P2')[s#,snombre])
```

Que en SQL corresponde a la siguiente sentencia:

```
SELECT s#,snombre
FROM   S
MINUS
SELECT S.s#,S.snombre
FROM   SP,S
WHERE  SP.s#=S.s# AND SP.p#='P2';
```

Ya que una diferencia obtiene los elementos que se encuentran en el primer conjunto y no en el segundo conjunto, podemos escribir una sentencia SQL equivalente que utiliza el operador NOT IN:

```
SELECT s#,snombre
FROM   S
WHERE  s# NOT IN (SELECT s#
                  FROM   SP
                  WHERE  SP.p#='P2');
```

Nótese que el operador NOT IN es equivalente a <> ALL.

1.9) Obtener los datos de los proveedores que sólo suministran piezas de color rojo.

```
T1 := (SP JOIN (P WHERE color='rojo'))[s#]
T2 := (SP JOIN (P WHERE color<>'rojo'))[s#]
RDO := (T1 MINUS T2) JOIN S
```

Que en SQL corresponde a la siguiente sentencia:

```
SELECT S.s#,S.snombre,S.estado,S.ciudad
FROM   SP,P,S
WHERE  SP.p#=P.p# AND SP.s#=S.s# AND P.color='rojo'
MINUS
SELECT S.s#,S.snombre,S.estado,S.ciudad
FROM   SP,P,S
WHERE  SP.p#=P.p# AND SP.s#=S.s# AND P.color<>'rojo';
```

La sentencia SQL equivalente que utiliza el operador NOT IN es la siguiente:

```
SELECT S.*
FROM   S
WHERE  s# IN (SELECT SP.s# FROM SP, P
              WHERE  SP.p#=P.p# AND P.color='rojo')
AND    s# NOT IN (SELECT SP.s# FROM SP, P
                  WHERE  SP.p#=P.p# AND P.color<>'rojo')
```

1.10) Obtener el nombre de los proveedores que suministran, al menos, todas las piezas que se almacenan en la ciudad de Paris.

```
T1 := SP[s#,p#]
T2 := (P WHERE ciudad='París')[p#]
RDO := ((T1 DIVIDEBY T2) JOIN S)[s#,snombre]
```

Otra solución se obtiene contando (es más larga, pero se puede escribir en SQL):

```
T1 := SUMMARIZE (P WHERE ciudad='París') GROUPBY()
                 ADD COUNT(*) AS piezas_París
T2 := (SP JOIN (P WHERE ciudad='París'))[s#,p#]
T3 := SUMMARIZE T2 GROUPBY(s#) ADD COUNT(*) AS piezas_París_s#
T4 := ((T1 TIMES T2) WHERE piezas_París = piezas_París_s#)[s#]
RDO := (T4 JOIN S)[s#,snombre]
```

Que en SQL corresponde a la siguiente sentencia:

```
SELECT S.s#,S.snombre
FROM   SP,P,S
WHERE  SP.p#=P.p# AND SP.s#=S.s# AND P.ciudad='París'
GROUP BY S.s#,S.snombre
HAVING COUNT(*)=(SELECT COUNT(*) FROM P WHERE ciudad='París');
```

1.11) Obtener los datos del envío de más piezas.

```
T1 := SUMMARIZE SP GROUPBY() ADD MAX(cant) AS cant_max
RDO := ((SP TIMES T1) WHERE cant=cant_max)[s#,p#,cant]
```

Que es equivalente a:

```
T1 := SUMMARIZE SP GROUPBY() ADD MAX(cant) AS cant
RDO := SP JOIN T1
```

La sentencia SQL que responde a esta consulta es la siguiente:

```
SELECT *
FROM   SP
WHERE  cant = (SELECT MAX(cant) FROM SP);
```

1.12) Para cada proveedor, mostrar la cantidad total de piezas que envía al mes, la cantidad media y el número de envíos.

```
SUMMARIZE SP GROUPBY(s#) ADD SUM(cant) AS cant_total,
                         ADD AVG(cant) AS cant_media,
                         ADD COUNT(*)  AS num_envíos
```

La sentencia SQL que responde a esta consulta es la siguiente:

```
SELECT s#, SUM(cant) cant_total, AVG(cant) cant_media,
           COUNT(*) num_envíos
FROM   SP
GROUP BY s#;
```

1.13) Obtener el código de los proveedores que realizan envíos en cantidades superiores a la cantidad media por envío.

```
T1 := SUMMARIZE SP GROUPBY() ADD AVG(cant) AS cant_media
RDO := ((SP TIMES T1) WHERE cant>cant_media)[s#]
```

La sentencia SQL que responde a esta consulta es la siguiente:

```
SELECT DISTINCT s#
FROM   SP
WHERE  cant > (SELECT AVG(cant) FROM SP);
```

1.14) Para cada ciudad en la que se almacenan piezas, obtener el número de piezas que almacena de cada color distinto.

```
SUMMARIZE P GROUPBY(ciudad,color) ADD COUNT(*) AS num_piezas
```

La sentencia SQL que responde a esta consulta es la siguiente:

```
SELECT ciudad,color,COUNT(*) num_piezas
FROM   P
GROUP BY ciudad,color;
```

Cálculo Relacional

1.1) Obtener el nombre de los proveedores que suministran la pieza con el código P2.

```
SX.s#,SX.snombre WHERE ∃SPX (SPX.s#=SX.s# AND SPX.p#='P2')
```

La sentencia SQL que responde a esta expresión es la siguiente:

```
SELECT s#,snombre
FROM   S
WHERE  EXISTS (SELECT *
               FROM   SP
               WHERE  SP.s#=S.s# AND SP.p#='P2');
```

Nótese la diferencia entre la estructura de esta sentencia y la sentencia basada en la expresión del álgebra relacional. Una de ellas será más eficiente que la otra en cuanto al tiempo de ejecución. Es por ello muy importante saber encontrar sentencias SQL equivalentes que resuelvan una misma consulta de modo que siempre se pueda escoger aquella que es más eficiente.

1.2) Obtener el nombre de los proveedores que suministran por lo menos una pieza roja.

```
SX.s#,SX.snombre WHERE
     ∃SPX (SPX.s#=SX.s# AND ∃PX (SPX.p#=PX.p# AND PX.color='rojo'))
```

La sentencia SQL que responde a esta expresión es la siguiente:

```
SELECT s#,snombre
FROM   S
WHERE  EXISTS (SELECT *
               FROM   SP
               WHERE  SP.s#=S.s#
               AND EXISTS (SELECT *
                           FROM   P
                           WHERE  SP.p#=P.p#
                           AND    P.color='rojo'));
```

1.3) Obtener el nombre de las piezas de color rojo suministradas por los proveedores de la ciudad de Londres.

```
PX.p#,PX.pnombre WHERE (PX.color='rojo' AND
     ∃SPX (SPX.p#=PX.p# AND
           ∃SX (SPX.s#=SX.s# AND SX.ciudad='Londres'))
```

La sentencia SQL que responde a esta expresión es la siguiente:

```
SELECT p#,pnombre
FROM   P
WHERE  color='rojo'
AND    EXISTS (SELECT *
               FROM   SP
               WHERE  SP.p#=P.p#
               AND EXISTS (SELECT *
                           FROM   S
                           WHERE  SP.s#=S.s#
                           AND    S.ciudad='Londres'));
```

1.4) Obtener el código de los proveedores que suministran alguna de las piezas que suministra el proveedor con el código S2.

```
SPX.s# WHERE ∃SPY (SPY.s#='S2' AND SPX.p#=SPY.p#)
```

La sentencia SQL que responde a esta expresión es la siguiente:

```
SELECT DISTINCT SPX.s#
FROM   SP SPX
WHERE  EXISTS (SELECT *
               FROM   SP SPY
               WHERE  SPY.s#='S2'
               AND    SPX.p#=SPY.p#);
```

1.5) Obtener los datos de los envíos de más de 100 unidades, mostrando también el nombre del proveedor y el de la pieza.

```
SPX.s#,SX.snombre,SPX.p#,PX.pnombre,SPX.cant WHERE
   (SPX.p#=PX.p# AND SPX.s#=SX.s# AND SPX.cant>100)
```

La sentencia SELECT equivalente es exactamente la misma que la sentencia que se basa en el álgebra relacional.

1.6) Obtener el nombre de los proveedores que suministran todas las piezas.

```
SX.s#,SX.snombre WHERE ∀PX ∃SPX (SPX.s#=SX.s# AND SPX.p#=PX.p#)
```

Que es equivalente a:

```
SX.s#,SX.snombre WHERE
   NOT ∃PX NOT ∃SPX (SPX.s#=SX.s# AND SPX.p#=PX.p#)
```

Esta expresión del cálculo relacional se puede escribir del siguiente modo en SQL:

```
SELECT s#,snombre
FROM   S
WHERE  NOT EXISTS (SELECT * FROM P
                   WHERE NOT EXISTS (SELECT * FROM SP
                                     WHERE  SP.s#=S.s#
                                     AND    SP.p#=P.p#));
```

1.7) Obtener el código de los proveedores que suministran, al menos, todas las piezas suministradas por el proveedor con código S2.

```
SX.s#,SX.snombre WHERE
   ∀SPX (IF SPX.s#='S2' THEN ∃SPY (SPY.s#=SX.s# AND SPX.p#=SPY.p#))
```

Que es equivalente a:

```
SX.s#,SX.snombre WHERE
   NOT ∃SPX (SPX.s#='S2' AND
             NOT ∃SPY (SPY.s#=SX.s# AND SPX.p#=SPY.p#))
```

La sentencia equivalente en SQL es la siguiente:

```
SELECT s#,snombre
FROM   S
WHERE  NOT EXISTS (SELECT * FROM SP SPX
                   WHERE  SPX.s#='S2'
                   AND    NOT EXISTS (SELECT * FROM SP SPY
                                      WHERE  SPY.s#=S.s#
                                      AND    SPX.p#=SPY.p#));
```

1.8) Obtener el nombre de los proveedores que no suministran la pieza con el código P2.

```
SX.s#,SX.snombre WHERE
   ∀SPX (IF SPX.s#=SX.s# THEN SPX.p#<>'P2')
```

Que es equivalente a:

```
SX.s#,SX.snombre WHERE
   NOT ∃SPX (SPX.s#=SX.s# AND SPX.p#='P2')
```

La sentencia equivalente en SQL es la siguiente:

```
SELECT s#,snombre
FROM   S
WHERE  NOT EXISTS (SELECT * FROM SP
                   WHERE  SP.s#=S.s#
                   AND    SP.p#='P2');
```

1.9) Obtener los datos de los proveedores que sólo suministran piezas de color rojo.

```
SX WHERE
   ∀SPX (IF SPX.s#=SX.s#
         THEN ∃PX (SPX.p#=PX.p# AND PX.color='rojo'))
```

Nótese que en la expresión anterior, si un proveedor no realiza ningún envío, sus datos salen en el resultado. Para solucionar este problema nos debemos asegurar que el proveedor realiza algún envío.

```
SX WHERE ∃SPY (SPY.s#=SX.s#) AND
   ∀SPX (IF SPX.s#=SX.s#
         THEN ∃PX (SPX.p#=PX.p# AND PX.color='rojo'))
```

Esta expresión es equivalente a:

```
SX WHERE ∃SPY (SPY.s#=SX.s#) AND
   NOT ∃SPX (SPX.s#=SX.s#
         AND NOT ∃PX (SPX.p#=PX.p# AND PX.color='rojo'))
```

La sentencia equivalente en SQL es la siguiente:

```
SELECT *
FROM   S
WHERE  EXISTS (SELECT * FROM SP
               WHERE  SP.s#=S.s#)
AND    NOT EXISTS (SELECT * FROM SP
                   WHERE  SP.s#=S.s#
                   AND    NOT EXISTS (SELECT * FROM P
                                      WHERE  P.p#=SP.p#
                                      AND    P.color='rojo'));
```

1.10) Obtener el nombre de los proveedores que suministran, al menos, todas las piezas que se almacenan en la ciudad de Paris.

```
SX.s#, SX.snombre WHERE ∃SPX (SPX.s#=SX.s#)
   ∀PX (IF PX.ciudad='París'
        THEN ∃SPY (SPY.p#=PX.p# AND SPY.s#=SX.s#))
```

Esta expresión es equivalente a:

```
SX.s#, SX.snombre WHERE ∃SPX (SPX.s#=SX.s#)
   NOT ∃PX (PX.ciudad='París' AND
        NOT ∃SPY (SPY.p#=PX.p# AND SPY.s#=SX.s#))
```

La sentencia equivalente en SQL es la siguiente:

```
SELECT s#,snombre
FROM   S
WHERE  EXISTS (SELECT * FROM SP
               WHERE  SP.s#=S.s#)
AND    NOT EXISTS (SELECT * FROM P
                   WHERE  P.ciudad='París'
                   AND    NOT EXISTS (SELECT * FROM SP
                                      WHERE  SP.p#=P.p#
                                      AND    SP.s#=S.s#));
```

1.11) Obtener los datos del envío de más piezas.

```
SPX WHERE ∀SPY (SPX.cant>=SPY.cant)
```

Que en SQL se puede escribir del siguiente modo:

```
SELECT *
FROM   SP
WHERE  cant >= ALL(SELECT cant FROM SP);
```

1.12) Para cada proveedor, mostrar la cantidad total de piezas que envía al mes, la cantidad media y el número de envíos.

No se puede resolver en el cálculo relacional.

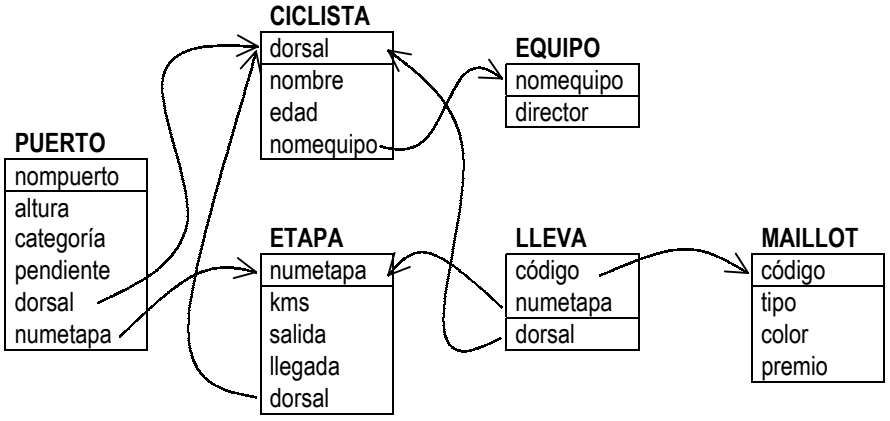
1.13) Obtener el código de los proveedores que realizan envíos en cantidades superiores a la cantidad media por envío.

No se puede resolver en el cálculo relacional.

1.14) Para cada ciudad en la que se almacenan piezas, obtener el número de piezas que almacena de cada color distinto.

No se puede resolver en el cálculo relacional.

APARTADO 2



2.1) Obtener los datos de las etapas que pasan por algún puerto de montaña y que tienen salida y llegada en la misma población.

```
(ETAPA JOIN PUERTO[numetapa]) WHERE salida=llegada
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT DISTINCT E.numetapa,E.kms,E.salida,E.llegada,E.dorsal
FROM   ETAPA E, PUERTO P
WHERE  E.numetapa = P.numetapa
AND    E.salida = E.llegada;
```

La expresión del cálculo relacional que responde a esta consulta es:

```
ETAPAX WHERE (ETAPAX.salida=ETAPAX.llegada AND
              ∃PUERTOX (PUERTOX.numetapa=ETAPAX.numetapa))
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT *
FROM   ETAPA E
WHERE  E.llegada=E.salida
AND    EXISTS (SELECT * FROM PUERTO P
               WHERE  E.numetapa = P.numetapa);
```

Se puede escribir una sentencia equivalente en la que se utilice el operador IN en lugar de EXISTS y no se necesite la referencia externa:

```
SELECT *
FROM   ETAPA
WHERE  llegada=salida
AND    numetapa IN (SELECT numetapa FROM PUERTO P);
```

2.2) Obtener las poblaciones que tienen la meta de alguna etapa, pero desde las que no se realiza ninguna salida.

```
ETAPA[llegada] MINUS ETAPA[salida]
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT llegada FROM ETAPA
MINUS
SELECT salida FROM ETAPA;
```

En la siguiente sentencia se utiliza el operador NOT IN en lugar de la operación MINUS:

```
SELECT DISTINCT llegada
FROM   ETAPA
WHERE  llegada NOT IN (SELECT salida FROM ETAPA);
```

La expresión del cálculo relacional que responde a esta consulta es:

```
ETAPAX.llegada WHERE ∀ETAPAY (ETAPAY.salida<>ETAPAX.llegada)
```

Esta espresión es equivalente a:

```
ETAPAX.llegada WHERE NOT ∃ETAPAY (ETAPAY.salida=ETAPAX.llegada)
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT llegada
FROM   ETAPA ETAPAX
WHERE  NOT EXISTS (SELECT * FROM ETAPA ETAPAY
                   WHERE  ETAPAY.salida=ETAPAX.llegada);
```

Una sentencia equivalente que no utiliza referencia externa es la siguiente:

```
SELECT llegada
FROM ETAPA
WHERE llegada <> ALL (SELECT salida FROM ETAPA);
```

2.3) Obtener el nombre y el equipo de los ciclistas que han ganado alguna etapa llevando el maillot amarillo, mostrando también el número de etapa.

```
T1 := (ETAPA JOIN LLEVA JOIN
              (MAILLOT WHERE color='amarillo'))[dorsal,numetapa]
T2 := (T1 JOIN CICLISTA)[dorsal,nombre,nomequipo,numetapa]
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT C.dorsal,C.nombre,C.nomequipo,E.numetapa
FROM   CICLISTA C,ETAPA E,LLEVA LL,MAILLOT M
WHERE  E.dorsal=LL.dorsal AND E.numetapa=LL.numetapa
AND    LL.código=M.código AND M.color='amarillo'
AND    E.dorsal=C.dorsal;
```

La expresión del cálculo relacional que responde a esta consulta es:

```
CICX.dorsal,CICX.nombre,CICX.nomequipo,ETAPAX.numetapa WHERE
 (ETAPAX.dorsal=CICX.dorsal AND ∃LLEVAX ∃MAILLOTX
  (ETAPAX.dorsal=LLEVAX.dorsal AND
   ETAPAX.numetapa=LLEVAX.numetapa AND LLEVAX.código=MAILLOTX.código
   AND MAILLOTX.color='amarillo'))
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT C.dorsal,C.nombre,C.nomequipo,E.numetapa
FROM   CICLISTA C,ETAPA E
WHERE  E.dorsal=C.dorsal
AND    EXISTS (SELECT * FROM LLEVA LL,MAILLOT M
               WHERE  E.dorsal=LL.dorsal AND
               E.numetapa=LL.numetapa AND LL.código=M.código
               AND M.color='amarillo');
```

2.4) Obtener los datos de las etapas que no comienzan en la misma ciudad en que acaba la etapa anterior.

```
T1 := ETAPA
RDO := ((T1 TIMES ETAPA) WHERE
   ETAPA.numetapa=T1.numetapa+1 AND ETAPA.salida<>T1.llegada)
[ETAPA.numetapa,ETAPA.kms,ETAPA.salida,ETAPA.llegada,ETAPA.dorsal]
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT ETAPAX.numetapa,ETAPAX.kms,ETAPAX.salida,
       ETAPAX.llegada,ETAPAX.dorsal
FROM   ETAPA ETAPAX, ETAPA ETAPAY
WHERE  ETAPAX.numetapa=ETAPAY.numetapa+1
AND    ETAPAX.salida<>ETAPAY.llegada;
```

La expresión del cálculo relacional que responde a esta consulta es:

```
ETAPAX WHERE ∃ETAPAY (ETAPAX.numetapa=ETAPAY.numetapa+1 AND
                      ETAPAX.salida<>ETAPAY.llegada)
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT *
FROM   ETAPA ETAPAX
WHERE  EXISTS (SELECT * FROM ETAPA ETAPAY
               WHERE  ETAPAX.numetapa=ETAPAY.numetapa+1
               AND    ETAPAX.salida<>ETAPAY.llegada);
```

2.5) Obtener el número de las etapas que tienen algún puerto de montaña, indicando cuántos tiene cada una de ellas.

```
SUMMARIZE PUERTO GROUPBY(numetapa) ADD COUNT(*) AS num_puertos
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT numetapa, COUNT(*) AS num_puertos
FROM   PUERTO
GROUP BY numetapa;
```

```
No se puede resolver en el cálculo relacional.
```

2.6) Obtener el nombre y la edad de los ciclistas que han llevado dos o más maillots en una misma etapa.

```
T1 := SUMMARIZE LLEVA GROUPBY(numetapa,dorsal)
      ADD COUNT(*) AS num_maillots
RDO := ( (T1 WHERE num_maillots>=2)
         JOIN CICLISTA)[dorsal,nombre,edad]
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT DISTINCT LL.dorsal,C.nombre,C.edad
FROM   CICLISTA C,LLEVA LL
WHERE  C.dorsal=LL.dorsal
GROUP BY LL.numetapa,LL.dorsal,C.nombre,C.edad
HAVING COUNT(*)>=2;
```

La expresión del cálculo relacional que responde a esta consulta es:

```
CICX.dorsal,CICX.nombre,CICX.edad WHERE ∃LLEVAX ∃LLEVAY
 (CICX.dorsal=LLEVAX.dorsal AND LLEVAX.dorsal=LLEVAY.dorsal AND
  LLEVAX.numetapa=LLEVAY.numetapa AND LLEVAX.código=LLEVAY.código)
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT dorsal,nombre,edad
FROM   CICLISTA C
WHERE  EXISTS (SELECT * FROM LLEVA LLX, LLEVA LLY
               WHERE  C.dorsal=LLX.dorsal
               AND    C.dorsal=LLY.dorsal
               AND    LLX.numetapa=LLY.numetapa
               AND    LLX.código<>LLY.código);
```

2.7) Obtener el nombre y el equipo de los ciclistas que han llevado algún maillot o que han ganado algún puerto.

```
( (LLEVA[dorsal] UNION PUERTO[dorsal])
   JOIN CICLISTA )[dorsal,nombre,nomequipo]
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT C.dorsal,C.nombre,C.nomequipo
FROM   CICLISTA C,LLEVA LL
WHERE  C.dorsal=LL.dorsal
UNION
SELECT C.dorsal,C.nombre,C.nomequipo
FROM   CICLISTA C,PUERTO P
WHERE  C.dorsal=P.dorsal;
```

La expresión del cálculo relacional que responde a esta consulta es:

```
CICX.dorsal,CICX.nombre,CICX.nomequipo WHERE
   ∃LLEVAX (LLEVAX.dorsal=CICX.dorsal) OR
   ∃PUERTOX (PUERTOX.dorsal=CICX.dorsal)
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT dorsal,nombre,nomequipo
FROM   CICLISTA C
WHERE  EXISTS (SELECT * FROM LLEVA LL
               WHERE  C.dorsal=LL.dorsal)
OR     EXISTS (SELECT * FROM PUERTO P
               WHERE  C.dorsal=P.dorsal);
```

Se puede escribir una sentencia equivalente que utiliza el operador IN en lugar de EXISTS, evitando las referencias externas:

```
SELECT dorsal,nombre,nomequipo
FROM   CICLISTA C
WHERE  dorsal IN (SELECT dorsal FROM LLEVA LL)
OR     dorsal IN (SELECT dorsal FROM PUERTO P);
```

2.8) Obtener los datos de los ciclistas que han vestido todos los maillots (no necesariamente en la misma etapa).

```
(LLEVA[dorsal,código] DIVIDEBY MAILLOT[código]) JOIN CICLISTA
```

O también:

```
T1 := SUMMARIZE (LLEVA[dorsal,código]) GROUPBY(dorsal)
      ADD COUNT(*) AS num_maillots
T2 := SUMMARIZE MAILLOT GROUPBY() ADD COUNT(*) AS num_maillots
RDO := T1 JOIN T2 JOIN CICLISTA
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT C.dorsal,C.nombre,C.edad,C.nomequipo
FROM   CICLISTA C,LLEVA LL
WHERE  C.dorsal=LL.dorsal
GROUP BY C.dorsal,C.nombre,C.edad,C.nomequipo
HAVING COUNT(DISTINCT LL.código) =
       (SELECT COUNT(*) FROM MAILLOT);
```

La expresión del cálculo relacional que responde a esta consulta es:

```
CICX WHERE ∀MAILLOTX ∃LLEVAX (MAILLOTX.código=LLEVAX.código AND
                              LLEVAX.dorsal=CICX.dorsal)
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT C.dorsal,C.nombre,C.edad,C.nomequipo
FROM   CICLISTA C
WHERE  NOT EXISTS (SELECT * FROM MAILLOT M
                   WHERE  NOT EXISTS
                        (SELECT * FROM   LLEVA LL
                         WHERE M.código=LL.código AND
                               LL.dorsal=C.dorsal));
```

2.9) Obtener el código y el color de aquellos maillots que sólo han sido llevados por ciclistas de un mismo equipo.

```
T1 := (LLEVA JOIN CICLISTA)[código,nomequipo]
T2 := SUMMARIZE T1 GROUPBY(código) ADD COUNT(*) AS num_equipos
RDO := ((T2 WHERE num_equipos=1) JOIN MAILLOT)[código,color]
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT LL.código,M.color
FROM   LLEVA LL, CICLISTA C, MAILLOT M
WHERE  LL.dorsal=C.dorsal AND LL.código=M.código
GROUP BY LL.código,M.color
HAVING COUNT(DISTINCT C.nomequipo)=1;
```

La expresión del cálculo relacional que responde a esta consulta es:

```
LLEVAX.código,MAILLOTX.color WHERE ∃CICX
   (CICX.dorsal=LLEVAX.dorsal AND LLEVAX.código=MAILLOTX.código
   AND ∀LLEVAY (IF LLEVAY.código=LLEVAX.código THEN ∃CICY
   (CICY.dorsal=LLEVAY.dorsal AND CICX.nomequipo=CICY.nomequipo)))
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT DISTINCT LLX.código,M.color
FROM   LLEVA LLX, MAILLOT M
WHERE  EXISTS (SELECT * FROM CICLISTA CX
               WHERE  LLX.dorsal=CX.dorsal
               AND    LLX.código=M.código
               AND    NOT EXISTS (SELECT * FROM LLEVA LLY
                                  WHERE LLX.código=LLY.código
                                  AND NOT EXISTS (SELECT *
```

```
                                    FROM  CICLISTA CY
                                    WHERE CY.dorsal=LLY.dorsal
                                    AND   CX.nomequipo=CY.nomequipo)));
```

Una sentencia equivalente que utiliza menos subconsultas es la siguiente:

```
SELECT DISTINCT LLX.código,M.color
FROM   LLEVA LLX, CICLISTA C, MAILLOT M
WHERE  LLX.dorsal=C.dorsal AND LLX.código=M.código
AND    C.nomequipo = ALL (SELECT C.nomequipo
                          FROM   LLEVA LLY, CICLISTA C
                          WHERE  LLY.dorsal=C.dorsal
                          AND    LLY.código=LLX.código);
```

2.10) Obtener los números de las etapas que no tienen puertos de montaña.

```
ETAPA[numetapa] MINUS PUERTO[numetapa]
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT numetapa FROM ETAPA
MINUS
SELECT numetapa FROM PUERTO;
```

En la siguiente sentencia se utiliza el operador NOT IN en lugar de la operación MINUS:

```
SELECT numetapa
FROM   ETAPA
WHERE  numetapa NOT IN (SELECT numetapa FROM PUERTO);
```

La expresión del cálculo relacional que responde a esta consulta es:

```
ETAPAX.numetapa WHERE NOT ∃PUERTOX
          (PUERTOX.numetapa=ETAPAX.numetapa)
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT E.numetapa
FROM   ETAPA E
WHERE  NOT EXISTS (SELECT *
                   FROM PUERTO P
                   WHERE P.numetapa=E.numetapa);
```

2.11) Obtener la edad media de los ciclistas que han ganado alguna etapa.

```
SUMMARIZE (ETAPA JOIN CICLISTA) GROUPBY() ADD AVG(edad) AS edad_med
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT AVG(C.edad)
FROM   CICLISTA C, ETAPA E
WHERE  C.dorsal=E.dorsal;
```

No se puede resolver en el cálculo relacional.

2.12) Obtener el nombre de los puertos de montaña que tienen una altura superior a la altura media de todos los puertos.

```
T1 := SUMMARIZE PUERTO GROUPBY() ADD AVG(altura) AS alt_media
RDO := ((PUERTO TIMES T1) WHERE altura>alt_media)[nompuerto]
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT nompuerto
FROM   PUERTO
WHERE  altura > (SELECT AVG(altura)
                 FROM   PUERTO);
```

No se puede resolver en el cálculo relacional.

2.13) Obtener las poblaciones de salida y de llegada de las etapas donde se encuentran los puertos con mayor pendiente.

```
T1 := SUMMARIZE PUERTO GROUPBY() ADD MAX(pendiente) AS pendiente
RDO := (T1 JOIN PUERTO JOIN ETAPA)[numetapa,salida,llegada]
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT DISTINCT E.numetapa,E.salida,E.llegada
FROM   ETAPA E,PUERTO P
WHERE  E.numetapa=P.numetapa
AND    P.pendiente = (SELECT MAX(pendiente)
                      FROM   PUERTO);
```

La expresión del cálculo relacional que responde a esta consulta es:

```
ETAPAX.numetapa,ETAPAX.salida,ETAPAX.llegada WHERE
 ∃PUERTOX (PUERTOX.numetapa=ETAPAX.numetapa AND
           ∀PUERTOY (PUERTOY.pendiente<=PUERTOX.pendiente))
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT E.numetapa,E.salida,E.llegada
FROM   ETAPA E
WHERE  EXISTS (SELECT * FROM PUERTO PX
               WHERE  E.numetapa=PX.numetapa
               AND    NOT EXISTS
                          (SELECT * FROM PUERTO PY
                           WHERE PY.pendiente>PX.pendiente));
```

Una sentencia equivalente que evita subconsultas con referencias externas es la siguiente:

```
SELECT DISTINCT E.numetapa,E.salida,E.llegada
FROM   ETAPA E,PUERTO P
WHERE  E.numetapa=P.numetapa
AND    P.pendiente >= ALL(SELECT pendiente
                          FROM   PUERTO);
```

2.14) Obtener el dorsal y el nombre de los ciclistas que han ganado los puertos de mayor altura.

```
T1 := SUMMARIZE PUERTO GROUPBY() ADD MAX(altura) AS altura
RDO := (T1 JOIN PUERTO JOIN CICLISTA)[dorsal,nombre]
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT DISTINCT C.dorsal,C.nombre
FROM   PUERTO P,CICLISTA C
WHERE  P.dorsal=C.dorsal
AND    P.altura = (SELECT MAX(altura) FROM PUERTO);
```

La expresión del cálculo relacional que responde a esta consulta es:

```
CICX.dorsal,CICX.nombre WHERE ∃PUERTOX (PUERTOX.dorsal=CICX.dorsal
        AND ∀PUERTOY (PUERTOY.altura<=PUERTOX.altura))
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT C.dorsal,C.nombre
FROM   CICLISTA C
WHERE  EXISTS (SELECT * FROM PUERTO PX
               WHERE  PX.dorsal=C.dorsal
               AND    NOT EXISTS (SELECT * FROM PUERTO PY
                                  WHERE PY.altura>PX.altura));
```

Una sentencia equivalente que evita subconsultas con referencias externas es la siguiente:

```
SELECT DISTINCT C.dorsal,C.nombre
FROM   PUERTO P,CICLISTA
WHERE  P.dorsal=C.dorsal
AND    P.altura >= ALL(SELECT altura FROM PUERTO);
```

2.15) Obtener los datos de las etapas cuyos puertos (todos) superan los 1300 metros de altura.

```
T1 := (PUERTO WHERE altura>1300)[numetapa]
       MINUS
      (PUERTO WHERE altura<=1300)[numetapa]
RDO := T1 JOIN ETAPA
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT E.*
FROM   PUERTO P, ETAPA E
WHERE  P.numetapa=E.numetapa AND P.altura>1300
MINUS
SELECT E.*
FROM   PUERTO P, ETAPA E
WHERE  P.numetapa=E.numetapa AND P.altura<=1300;
```

En la siguiente sentencia se utiliza el operador NOT IN en lugar de la operación MINUS:

```
SELECT DISTINCT E.*
FROM   PUERTO P, ETAPA E
```

```
WHERE  P.numetapa=E.numetapa AND P.altura>1300
AND    E.numetapa NOT IN (SELECT numetapa
                          FROM   PUERTO
                          WHERE  altura<=1300);
```

La expresión del cálculo relacional que responde a esta consulta es:

```
ETAPAX WHERE ∃PUERTOX (PUERTOX.numetapa=ETAPAX.numetapa) AND
 ∀PUERTOX (IF PUERTOX.numetapa=ETAPAX.numetapa
          THEN PUERTOX.altura>1300)
```

La sentencia SQL que corresponde a esta expresión es la siguiente:

```
SELECT *
FROM   ETAPA E
WHERE  EXISTS (SELECT * FROM PUERTO P
               WHERE  P.numetapa=E.numetapa)
AND    NOT EXISTS (SELECT * FROM PUERTO P
                   WHERE  P.numetapa=E.numetapa
                   AND    P.altura<=1300);
```

Una sentencia equivalente es la siguiente:

```
SELECT *
FROM   ETAPA E
WHERE  EXISTS (SELECT * FROM PUERTO P
               WHERE  P.numetapa=E.numetapa)
AND    1300 < ALL (SELECT P.altura FROM PUERTO P
                   WHERE  P.numetapa=E.numetapa);
```

El predicado EXISTS es necesario porque, cuando una etapa no tiene puertos de montaña, la subconsulta del operador ALL no devuelve ninguna fila y la expresión 1300 < ALL(null) se evalúa a verdadero (funciona como el ∀ del cálculo relacional); es por ello que se debe mantener la restricción de que la etapa tenga algún puerto de montaña.

El predicado EXISTS es equivalente a la siguiente expresión, en la que no se utiliza referencia externa en la subconsulta:

```
WHERE  numetapa IN (SELECT numetapa FROM PUERTO P)
```

El predicado de la consulta anteior:

```
1300 < ALL (SELECT P.altura FROM PUERTO P
            WHERE  P.numetapa=E.numetapa)
```

es equivalente a este otro predicado:

```
1300 < (SELECT MIN(P.altura) FROM PUERTO P
        WHERE  P.numetapa=E.numetapa)
```

En este caso, cuando una estapa no tiene puertos, la subconsulta devuelve null y la expresión 1300<null se evalúa a falso, por lo que esta etapa no se obtiene en el resultado de la consulta principal. Esto permite prescindir del predicado EXISTS (o el equivalente NOT IN), obteniéndose la siguiente sentencia SQL:

```
SELECT *
FROM   ETAPA E
WHERE  1300 < (SELECT MIN(P.altura) FROM PUERTO P
               WHERE  P.numetapa=E.numetapa);
```

2.16) Obtener el nombre de los ciclistas que pertenecen a un equipo de más de cinco ciclistas y que han ganado alguna etapa, indicando también cuántas etapas han ganado.

```
T1 := SUMMARIZE CICLISTA GROUPBY(nomequipo) ADD COUNT(*) AS num_cic
T2 := (T1 WHERE num_cic>5)[nomequipo]
RDO := SUMMARIZE (CICLISTA JOIN ETAPA JOIN T2)
       GROUPBY(dorsal,nombre) ADD COUNT(*) AS num_etapas_ganadas
```

La sentencia SQL equivalente es la siguiente:

```
SELECT C.dorsal,C.nombre,COUNT(*) AS num_etapas_ganadas
FROM   CICLISTA C,ETAPA E
WHERE  C.dorsal=E.dorsal
AND    C.nomequipo IN (SELECT nomequipo
                       FROM CICLISTA
                       GROUP BY nomequipo
                       HAVING COUNT(*)>5)
GROUP BY C.dorsal,C.nombre;
```

No se puede resolver en el cálculo relacional.