Práctica SQL (DML) - RESUELTA

Sea la base de datos de proveedores, partes y proyectos, cuyo **esquema conceptual** es:

S (S#, SNOMBRE, SITUACIÓN, CIUDAD)
P (P#, PNOMBRE, COLOR, PESO, CIUDAD)
J (J#, JNOMBRE, CIUDAD)
SPJ (S#, P#, J#, CANT)

En los ejercicios siguientes escribir una proposición o conjunto de proposiciones en SQL para la operación indicada.

Consultas sencillas

1. Obtener los detalles completos de todos los proyectos.

```
SELECT    J#, JNOMBRE, CIUDAD    FROM    J;
```

Consulta alternativa: SELECT * FROM J porque se quiere visualizar todos los campos (atributos) de la tabla J.

2. Obtener los detalles completos de todos los proyectos de Londres.

```
SELECT    J#, JNOMBRE, CIUDAD   FROM   J
          WHERE CIUDAD = 'Londres';
```

Se asigna una condición con el WHERE para ver sólo las filas (tuplas) que la cumplen.

3. Obtener los números de los proveedores que suministran (o envían) partes al proyecto J1, ordenados por número de proveedor.

```
SELECT    DISTINCT S#          FROM   SPJ
      WHERE  J# = 'J1'
      ORDER  BY S# ;
```

4. Obtener todos los envíos en los cuales la cantidad está en el intervalo de 300 a 750 inclusive.

```
SELECT S#, P#, J#, CANT          FROM   SPJ
     WHERE  CANT >= 300    AND        CANT <= 750;
```

300 <= cant <= 750

Alternativa: Usar WHERE Between (300, 750)
Recordar: Ver si lo admite el DBMS y ver cómo maneja los límites del intervalo

5. Obtener una lista de todas las combinaciones parte-color/parte-ciudad, eliminando todas las parejas color/ciudad repetidas.

```
SELECT DISTINCT COLOR, CIUDAD  FROM   P ;
```

El uso de DISTINCT hace que no se muestren las filas repetidas (filas con los mismos valores)

Reuniones (JOIN)

6. **Obtener todas las 3-uplas número de proveedor/número de parte/ número de proyecto tales que el proveedor, la parte y el proyecto indicados estén todos en la misma ciudad (cosituados).**

Se debe analizar en la expresión de la consulta que se quiere hacer, dónde se encuentran los datos a ver (que se listan en SELECT) y dónde se encuentran los datos a los cuales se les pondrán condiciones (indicadas en el WHERE). Estas serán las tablas que van en FROM.

```
SELECT S#, P#, J#              FROM S, P, J
      WHERE  S.CIUDAD = P.CIUDAD    AND      P.CIUDAD = J.CIUDAD;
```

Se piden 2 (dos) condiciones que deben cumplirse al mismo tiempo, por eso se usa AND que indica que se cumpla la condición primera y la condición segunda para brindar las filas de la respuesta.

7. **Obtener todas las 3-uplas número de proveedor/número de parte/ número de proyecto tales que el proveedor, la parte y el proyecto indicados no estén todos cosituados.**

```
SELECT S#, P#, J#    FROM S, P, J
       WHERE  NOT   (S.CIUDAD = P.CIUDAD    AND   P.CIUDAD=J.CIUDAD);
```

Consulta alternativa:

```
SELECT S#, P#, J#      FROM  S, P, J
WHERE  S.CIUDAD <> P.CIUDAD   OR  P.CIUDAD <> J.CIUDAD OR S.CIUDAD <> J.CIUDAD;
```

8. **Obtener todas las 3-uplas número de proveedor/número de parte/ número de proyecto tales que el proveedor, la parte y el proyecto indicados estén todos en diferente ciudad.**

```
SELECT S#, P#, J#    FROM  S, P, J
       WHERE         S.CIUDAD <> P.CIUDAD
       AND     P.CIUDAD <> J.CIUDAD
       AND    J.CIUDAD <> S.CIUDAD
```

9. **Obtener los números de las partes suministradas por algún proveedor de Londres.**

```
SELECT DISTINCT P#         FROM SPJ, S
      WHERE    SPJ.S# = S.S#         AND    CIUDAD = 'Londres';
```

10. **Obtener los números de las partes suministradas por un proveedor de Londres a un proyecto en Londres.**

```
SELECT DISTINCT P#         FROM SPJ, S, J
       WHERE   SPJ.S# = S.S#    AND   SPJ.J# = J.J#
       AND   S.CIUDAD = 'Londres'   AND   J.CIUDAD = ' Londres';
```

11. **Obtener todas las parejas de nombres de ciudad tales que un proveedor de la primera ciudad suministre partes a un proyecto en la segunda ciudad.**

```
SELECT  DISTINCT  S.CIUDAD,  J.CIUDAD       FROM  S, SPJ, J
       WHERE     S.S# = SPJ.S#     AND    SPJ.J# = J.J#;
```

12. Obtener los números de las partes suministradas a un proyecto por un proveedor situado en la misma ciudad que el proyecto.

```
SELECT DISTINCT  P#       FROM SPJ, S, J
        WHERE        SPJ.S# = S.S#   AND   SPJ.J# = J.J#
        AND   S.CIUDAD = J.CIUDAD;
```

13. Obtener los números de los proyectos a los cuales suministra partes por lo menos un proveedor situado en una ciudad distinta.

```
SELECT DISTINCT J.J#      FROM SPJ, S, J
        WHERE        SPJ.S# = S.S#   AND SPJ.J# = J.J#
        AND   S.CIUDAD <> J.CIUDAD;
```

14. Obtener todas las parejas de números de parte tales que algún proveedor suministre las dos partes indicadas.

```
SELECT  SPJX.P#  , SPJY.P#          FROM  SPJ SPJX, SPJ  SPJY
        WHERE     SPJX.S# = SPJY.S#     AND     SPJX.P# > SPJY.P#;
```

La condición resaltada en amarillo hace que las filas del tipo

~~P1 P1~~
~~P1 P2~~
~~P2 P2~~

no se muestren y de esta forma queda sólo la fila

P2 P1

Aunque esta condición no afecta la correcta ejecución de la consulta, su inclusión hace que el resultado mostrado al usuario humano sea más entendible por él (sacamos "ruido").

<u>Funciones de agregados</u>

Estas funciones **se pueden utilizar solamente en dos lugares** de la sentencia SELECT:

- como un dato a visualizar (columna) en la lista de columnas del SELECT
- como una condición en el HAVING que indique qué grupos se desea mostrar.

NO se pueden **utilizar** como condición en el **WHERE**.

15. Obtener el número total de proyectos a los cuales suministra partes el proveedor S1.

```
SELECT COUNT (DISTINCT J#)    FROM    SPJ
      WHERE    S#   = 'S1';
```

16. Obtener la cantidad total de la parte P1 suministrada por el proveedor S1.

```
SELECT SUM (CANT)       FROM SPJ
      WHERE  P#  = 'P1'    AND     S#       = 'S1';
```

17. Para cada parte suministrada a un proyecto, obtener el número de parte, el número de proyecto y la cantidad total correspondiente.

```
SELECT P#, J#, SUM (CANT)       FROM     SPJ
        GROUP    BY P#, J#;
```

18. Obtener los números de las partes suministradas a algún proyecto tales que la cantidad promedio suministrada sea mayor que 320.

```
SELECT DISTINCT P#        FROM SPJ
        GROUP BY P#, J#
        HAVING        AVG (CANT) > 320;
```

Operativamente, lo que hace el motor (gestor) de base de datos (en inglés DBMS, en español SGBD) utilizado, es generar una tabla temporaria para cada grupo y aplicar la condición dada en el HAVING a cada una de ellas para mostrar o no ese grupo. Estas tablas temporarias tienen como "nombre" de tabla los valores que se pide agrupar.
Para la instancia que tenemos, se generan los grupos:

P1J1

P#	J#	CANT
P1	J1	200

P1J4

P#	J#	CANT
P1	J4	700
P1	J4	100

P2J2

P#	J#	CANT
P2	J2	200

P2J4, P3J1, P3J2, P3J3, P3J4, P3J5, P3J6, P3J7, P4J2, P4J4, P5J5, P5J7, P6J3, P6J4 y P6J7.

Luego se aplica la condición dada en el HAVING y se mantienen únicamente los grupos que la cumplen. Por ejemplo, el grupo P1J1 tiene una sola fila, con un valor de CANT de 200, con lo cual el promedio (AVG) es 200. Así, la condición del HAVING es FALSA y el grupo P1J1 se descarta. En cambio, el grupo P1J4, que tiene dos filas, da un promedio de cantidad de 400, en este caso, el grupo se mantiene.
Cuando termina de evaluar todos los grupos, aplica el pedido de visualización de columnas dado en la parte SELECT de la sentencia.

La parte HAVING de la sentencia SELECT es una condición que se aplica a los grupos, esto es DEBE haber un GROUP BY para utilizar el HAVING.

En el caso de haber agregado una condición de WHERE en la consulta, **ANTES** de armar los grupos aplica esa condición quedando menos filas.
Por ejemplo: si pedimos en la consulta anterior que sean las partes provistas por el proveedor S1, la consulta queda:

```
SELECT DISTINCT P#        FROM SPJ
        WHERE  S# = 'S1'
        GROUP BY P#, J#
        HAVING        AVG (CANT) > 320;
```

y la tabla sobre la cual aplica el procedimiento descrito para el agrupado (GROUP BY) y su condición (HAVING) es:

S#	P#	J#	CANT
S1	P1	J1	200
S1	P1	J4	700

Diversas

19. Obtener todos los envíos para los cuales la cantidad no sea nula.

```
SELECT S#, P#, J#, CANT    FROM     SPJ
        WHERE CANT IS NOT NULL
```

La anterior es la respuesta "oficial". Sin embargo, ésta también funciona:

```
SELECT  S#, P#, J#, CANT   FROM SPJ
        WHERE  CANT = CANT;
```

20. Obtener números de proyecto y ciudades en los cuales la segunda letra del nombre de la ciudad sea una "o".

```
SELECT        J#, CIUDAD            FROM    J
        WHERE CIUDAD LIKE   '_o%';
```

Subconsultas

Todas estas soluciones expresadas con subconsultas también pueden resolverse con reuniones (join). Sugerencia: resolver los ejercicios siguientes utilizando reuniones para verificarlo.

Pero, ¿cuáles son las **ventajas de utilizar subconsultas**???

- Permite dividir el problema en **subproblemas**, dado que se pueden ir resolviendo partes de la consulta en una subconsulta.
- Mejora la **performance** dado que el resultado de una subconsulta es un valor o una lista de valores. Al hacer una reunión (join) se generan tablas más grandes a las que luego se aplican las condiciones del WHERE y/o los GROUP BY, esto implica utilizar mayor espacio de memoria y es posible que aumente la cantidad de accesos disco-memoria y memoria-disco para resolver la consulta. Esto se verá mejor en el tema de Optimización.

Para evaluar condiciones en las subconsultas, los operadores son **IN**, **NOT IN**, **=** (si aseguro que el resultado es **UN SOLO valor**), < y > (si el resultado de la subconsulta es un **número** y lo utilizo para evaluar una condición contra un campo numérico).

Las subconsultas se pueden anidar.

21. Obtener los nombres de los proyectos a los cuales suministra partes el proveedor S1.

```
SELECT JNOMBRE            FROM      J
      WHERE J#   IN (SELECT J#   FROM  SPJ  WHERE S# = 'S1' ) ;
```

Resultado de la subconsulta:

J#
J1
J4

Respuesta de la consulta:

JNOMBRE
Clasificador
Consola

22. Obtener los colores de las partes suministradas por el proveedor S1.

```
SELECT DISTINCT COLOR          FROM     P
      WHERE P# IN  (SELECT P#  FROM  SPJ      WHERE S# = 'S1'  )  ;
```

23. Obtener los números de las partes suministradas a cualquier proyecto en Londres.

```
SELECT DISTINCT P#        FROM     SPJ
      WHERE    J#   IN       (SELECT J#    FROM     J     WHERE   CIUDAD  =  'Londres' ) ;
```

Resultado de la subconsulta (que se ejecuta UNA SOLA vez) que consiste de 2 valores J5 y J7, que se almacenan en un espacio temporario de memoria:

J#
J5
J7

Respuesta:

P#
P3
P5
P6

24. Obtener los números de los proyectos donde se utilice al menos una de las partes suministradas por el proveedor S1.

```
SELECT DISTINCT J#         FROM      SPJ
       WHERE    P#   IN (SELECT P#  FROM SPJ     WHERE S# = 'S1'  )  ;
```

25. Obtener los números de los proveedores que suministren por lo menos una de las partes suministradas por al menos uno de los proveedores que suministran por lo menos una parte roja.

```
SELECT DISTINCT S#        FROM     SPJ   WHERE  P# IN
       (SELECT P#    FROM SPJ     WHERE  S#   IN
              ( SELECT S#    FROM SPJ  WHERE P# IN
                     (SELECT P#  FROM    P  WHERE COLOR  =  'Rojo'  ) ) ) ;
```

Sugerencia: Escribir las subconsultas, empezando desde la última condición de la consigna hacia la izquierda. Esto es, escribirlas en el orden siguiente:

- **Primero: partes rojas.** Se obtiene una lista de los P# que son rojos.
- **Segundo: proveedores que las suministran.** Se obtiene una lista de los proveedores (S#) que proveen esos P#.
- **Tercero: partes suministradas.** Se obtiene una lista de las partes (P#) que son provistas por los proveedores encontrados en la lista de **Segundo**.
- **Cuarto: RESPUESTA.** Esto es la lista de los S# que suministran las partes de **Tercero.**

26. Obtener los números de proveedores cuya situación sea inferior a la del proveedor S1.

```
SELECT S#    FROM S
      WHERE  SITUACIÓN <   (SELECT SITUACION   FROM    S   WHERE S# = 'S1' ) ;
```

Observaciones:

- el campo (o atributo) SITUACIÓN está definido como numérico, así que lo que hace el < es evaluar que el valor de este campo sea más chico que el que corresponde a S1.
- El resultado de la subconsulta es un valor numérico, por eso se puede usar el <.

27. Obtener los números de los proyectos cuya ciudad sea la primera en la lista alfabética de las ciudades donde hay proyectos.

```
SELECT J#     FROM  J
        WHERE  CIUDAD  =  (SELECT MIN (CIUDAD)  FROM  J ) ;
```

Observaciones:

- en la subconsulta se utiliza una función de agregados (MINimo) que, utilizando los códigos ASCII, compara cada string de caracteres contenido en el campo CIUDAD y determina cuál de los strings es el más chico.
- Las funciones MIN y MAX se pueden utilizar entonces para comparar strings de caracteres (o valores numéricos quedándose con el número más chico o el más grande). El resultado será siempre un único valor.
- El resultado de la subconsulta es un **único valor**, la primera en la lista de ciudades donde están ubicados los proyectos. Como es un único valor se puede usar el =.

EXISTS (EXISTE)

EXISTS operador que da VERDADERO si el resultado de la subconsulta TIENE FILAS, y FALSO si la subconsuta NO TIENE filas

28. Repetir el ejercicio 23 (Obtener los números de las partes suministradas a cualquier proyecto en Londres) utilizando EXISTS en la solución.

```
SELECT  DISTINCT P#        FROM SPJ
        WHERE  EXISTS      ( SELECT  *   FROM J
                    WHERE   J# = SPJ.J#   AND   CIUDAD = 'Londres' ) ;
```

Forma de resolución:

- PRIMERA Fila de SPJ:

Toma el valor de J# y resuelve la subconsulta. Esta subconsulta NO DA FILAS ⇒ el EXISTS es FALSO (en el WHERE J.J#=SPJ.J# PERO CIUDAD ≠ 'Londres") ⇒ NO MUESTRA el valor de P# (P1).

- SEGUNDA a SEXTA Filas de SPJ:

Idem primera Fila: NO DA FILAS ⇒ FALSO ⇒ NO MUESTRA los valores de P#.

- SEPTIMA Fila de SPJ:

la subconsulta da una FILA (en la tabla J, J5 está ubicado en Londres ⇒ el WHERE es VERDADERO. Por lo tanto MUESTRA el valor de P3 (correspondiente a P# para la fila 7)

- Y así sigue hasta terminar de recorrer las 24 filas de la tabla SPJ. Con lo cual tendremos una evaluación VERDADERO del EXISTS también en las filas 9, 14, 17, 18 y 19.

Es decir: **se resuelve la subconsulta tantas veces como filas tenga la tabla que se recorre externamente**, que en este caso es la tabla SPJ**. Esto es una DESVENTAJA** (comparar contra el uso del IN en la consulta 23, donde la subconsulta se ejecuta UNA SOLA vez)**.**

Nro Fila	S#	P#	J#	CANT
1	S1	P1	J1	200
2	S1	P1	J4	700
3	S2	P3	J1	400
4	S2	P3	J2	200
5	S2	P3	J3	200
6	S2	P3	J4	500
7	S2	P3	J5	600
8	S2	P3	J6	400
9	S2	P3	J7	800
10	S2	P5	J2	100
11	S3	P3	J1	200
12	S3	P4	J2	500
13	S4	P6	J3	300
14	S4	P6	J7	300
15	S5	P2	J2	200
16	S5	P2	J4	100
17	S5	P5	J5	500
18	S5	P5	J7	100
19	S5	P5	J7	100
20	S5	P1	J4	100
21	S5	P3	J4	200
22	S5	P4	J4	800
23	S5	P5	J4	400
24	S5	P6	J4	500

J#	JNOMBRE	CIUDAD
J1	Clasificador	París
J2	Perforadora	Roma
J3	Lectora	Atenas
J4	Consola	Atenas
J5	Compaginador	Londres
J6	Terminal	Oslo
J7	Cinta	Londres

Resultado, una vez aplicado el DISTINCT:

P#
P3
P5
P6

Observaciones:

- Notar la **diferencia de sintaxis** en el uso de los operadores IN y EXISTS.
- **IN** (o NOT IN) requiere **indicar el nombre del campo** (o atributo) en el cual se desea evaluar si su valor es o no es uno de la lista de valores obtenida con la subconsulta. Esto es equivalente a evaluar la **pertenencia** (o no pertenencia en el caso del NOT IN) de un valor a un conjunto ($x \in$ {conjunto de valores: x_1, x_2, ... x_n} o $x \notin$ {conjunto de valores: x_1, x_2, ... x_n}). **"WHERE Campo IN (resultado de la subconsulta)"** será **VERDADERO** si el valor sobre el que se está posicionado del campo es alguno ($\in$) de los obtenidos en la subconsulta (conjunto).

– En cambio, **EXISTS** (o NOT EXISTS) evalúa la existencia (o la no existencia) de resultados de la subconsulta, por esto **NO SE INDICA el nombre de ningún campo** y el **SELECT de la subconsulta puede llevar un *** como lista de valores a nombrar: **"WHERE EXISTS (resultado de la subconsulta)"**. EXISTS será **VERDADERO** si al ejecutar la subconsulta el resultado de ésta es **una o más filas**.

– Además, para que este operador funcione correctamente, **DEBE vincularse la ejecución de la subconsulta a algún o algunos valores** correspondientes a atributos de la fila de **tabla abierta en la consulta externa** en la cual está posicionada. Esto no es necesario en el caso de utilizar el operador IN.

29. Repetir el ejercicio 24 (Obtener los números de los proyectos donde se utilice al menos una de las partes suministradas por el proveedor S1) utilizando EXISTS en la solución.

```
SELECT  DISTINCT UNO.J#          FROM    SPJ UNO
        WHERE EXISTS       ( SELECT  *   FROM    SPJ DOS
                WHERE   DOS.P# = UNO.P#    AND  DOS.S# = 'S1'  ) ;
```

Observaciones:

- Como se recurre a la misma tabla SPJ para la consulta exterior y para la subconsulta, es necesario **renombrarla**: se la llama UNO para la consulta exterior y DOS para la subconsulta. De esta forma se tiene la MISMA tabla, pero en dos espacios de almacenamiento distintos, llamados UNO y DOS, respectivamente.
- La única diferencia con el procedimiento del ejercicio anterior es que recorre la tabla SPJ guardada en el espacio denominado UNO, y ejecuta la subconsulta contra la misma tabla SPJ pero guardada en el espacio llamado DOS.
- De esta forma se pueden comparar valores de un atributo que estén en distintas filas de una misma tabla.

30. Obtener los números de los proyectos a los cuales no suministra ninguna parte roja ninguno de los proveedores de Londres.

```
SELECT  J#    FROM  J
        WHERE  NOT EXISTS        ( SELECT *    FROM   SPJ
                WHERE  J# = J.J#
                        AND   P#  IN
                                ( SELECT P#  FROM  P
                                        WHERE COLOR  = 'Rojo')
                        AND   S#   IN
                                ( SELECT S#  FROM  S
                                        WHERE CIUDAD  = 'Londres' ) ) ;
```

Observaciones:

- Dado que se desea evaluar que **no suministre**, se utiliza el **NOT EXISTS** (esto es que el resultado de la subconsulta **NO** de filas). Si el resultado NO DA filas, el NOT EXISTS es VERDADERO.
- La subconsulta en verde, dará verdadero si el valor de P# es uno de los valores de P# correspondiente a aquellas partes de color rojo.
- La subconsulta en celeste, dará verdadero si el valor de S# es uno de los valores de S# correspondiente a aquellos proveedores ubicados en Londres.

- Si NO HAY FILAS en SPJ donde el valor de J# de J donde se está ubicado, está acompañado por un P# que verifique la subconsulta verde y también esté acompañado por un S# que verifique la subconsulta celeste, entonces el NOT EXISTS será verdadero y se muestra ese valor de J#.
- El DISTINCT en la consulta externa no es necesario porque en la tabla J cada valor de J# aparece una sola vez.

31. Obtener los números de los proyectos para los cuales S1 es el único proveedor.

```
SELECT  DISTINCT J#       FROM  SPJ   UNO
        WHERE   NOT EXISTS      ( SELECT *    FROM  SPJ   DOS
                WHERE  DOS.J# = UNO.J#  AND DOS.S# <>  'S1' )  ;
```

Observaciones:

- Lo que se pretende con esta consulta es que, para un dado proyecto (que no se conoce ya que es lo que se desea averiguar), si hubiera varias filas en la tabla SPJ donde aparece ese J# en **TODAS** esas filas el valor de S# sea **S1**. Esto se logra con el NOT EXISTS planteado, porque se buscan aquellas filas de SPJ donde el valor de S# no sea S1 para un J# dado.

¿Cómo se resuelve la consulta?

1. Se posiciona en la primera fila de UNO.
2. Se toma el valor de J# = 'J1'.
3. Se ejecuta la subconsulta, donde la condición del WHERE será: DOS.J# = 'J1' AND DOS.S# <> 'S1'.
4. Si hay **ALGUNA** fila en DOS en la cual este WHERE es VERDADERO, quiere decir que J1 es provisto por OTROS proveedores, además de S1. Así, como se obtienen filas, el resultado del **NOT EXISTS es FALSO** y descarta el valor de J1 de la primera fila de UNO. Si no hay **NINGUNA** fila en DOS donde el WHERE es verdadero significa que no hay ninguna fila donde J1 sea provista por un proveedor distinto de S1 y el **NOT EXISTS es VERDADERO**, por lo cual se conserva el valor de J1 como parte de la respuesta final.
5. Los cuatro puntos anteriores (1. a 4.) se repiten para las 24 filas de la tabla UNO.

*** EJERCICIOS ADICIONALES ***
(No están en los enunciados de la práctica)

I. Obtener los números de las partes suministradas a todos los proyectos en Londres.

Se resuelve una condición de **TODOS**.
¿Qué significa esto? Pensemos:

- La tabla J tiene 2 (dos) proyectos ubicados en ´Londres' (en la instancia son J5 y J7).
- Se quieren ver los P# tales que en una fila de SPJ estén acompañados por J5 y en otra fila estén acompañados por J7. Es decir, que existan por lo menos esas dos filas. No interesa si hay más filas donde la parte esté acompañada por algún/os proyecto/s más. Por ejemplo, si una parte aparece en 5 filas donde está acompañada por J5, J7, J1, J4 y J3, como aparecen los dos proyectos ubicados en Londres, esta parte P# cumple lo solicitado en la consulta.
- En principio, el dato de los J# que estén ubicados en Londres es DESCONOCIDO. Por esto debemos construir la consulta para poder responder a la consigna sin saber previamente cuáles son los valores de J# que la cumplen. Además, la consulta debe proveer la condición de que al menos exista una fila en SPJ donde un valor de P# está acompañada por uno de los J# y haya otra(s) filas donde ese valor de P# esté acompañada por el (los) otro(s) valores de P#.

¿Cómo lo resolvemos? Esto se puede resolver de al menos dos maneras:
Forma 1: utilizando un doble NOT EXISTS anidado.
Forma 2: utilizando GROUP BY y condiciones HAVING a los grupos.

Forma 1: doble NOT EXISTS anidado

```
SELECT  DISTINCT P#   FROM   SPJ   UNO
WHERE    NOT EXISTS   ( SELECT *   FROM   J
        WHERE   CIUDAD = 'Londres'
                    AND    NOT EXISTS
                        ( SELECT *   FROM          SPJ DOS
                            WHERE  DOS.P# = UNO.P#
                                AND   DOS.J# = J.J# ) ) ;
```

Funcionamiento de esta consulta:
Tengamos presentes las tablas SPJ (apertura UNO), J y SPJ (apertura DOS)

1. Se abre UNO y nos ubicamos en la Fila 1 (S#='S1', P# = P1, J# = J1, CANT = 200). Aclaración: en lo que sigue ignoro la columna CANT porque no es relevante para la resolución. El valor que vamos a utilizar es el de P#.
2. **Primera subconsulta**: se abre la tabla J y se aplica la condición CIUDAD = 'Londres' del WHERE para quedarse únicamente con las filas correspondientes a J5 y J7. Se toma la primera fila que quedó y que corresponde a J# = J5.
3. **Segunda subconsulta**: se abre la tabla DOS y se resuelve con los valores P1 y J5. Esto es, evaluar en la tabla DOS si hay filas donde P#=P1 y J#=J5. En la instancia dada esto NO OCURRE. Por lo tanto la segunda subconsulta NO da FILAS y el **NOT EXISTS** antepuesto a esta segunda subconsulta **acumula un VERDADERO**.
4. Se vuelve a la primera subconsulta y se pasa a la fila siguiente de donde se toma J# =J7. Es decir, se recorre TODA la tabla del FROM de la primer subconsulta, que en nuestro caso está reducida a dos filas. Si no apareciera la condición de CIUDAD = 'Londres', se recorrerían las 7 filas de la tabla J.
5. Se vuelve a resolver la **segunda subconsulta**, ahora con los valores P1 y J7. En la instancia dada esto TAMPOCO OCURRE. Por lo tanto la segunda subconsulta NO da FILAS, por lo que se **agrega otro VERDADERO** al **NOT EXISTS** antepuesto a esta segunda subconsulta.
6. Quiere decir entonces que la **primer subconsulta da filas**, y la aplicación del **NOT EXISTS** resulta en **FALSO** y el valor de **P#= P1 NO APARECERÁ EN LA RESPUESTA**.
7. Se pasa a la Fila 2 de UNO y se repiten los pasos 2. a 6. Y así se sigue hasta terminar de recorrer la tabla UNO.

¿Cuándo se tendrá un valor de P# que responda a la consulta? Repetiré los pasos anteriores para un caso donde se obtenga un P#.

1. Se abre UNO y nos ubicamos en la Fila 7 (S#='S2', P# = P3, J# = J5). El valor que vamos a utilizar es el de P#.
2. Primera subconsulta: se abre la tabla J y se aplica la condición CIUDAD = 'Londres' para quedarse únicamente con las filas correspondientes a J5 y J7. Se toma la primera fila que quedó y que corresponde a J# = J5.
3. **Segunda subconsulta**: se abre la tabla DOS y se resuelve con los valores P3 y J5. Esto es, evaluar en la tabla DOS si hay filas donde P#=P3 y J#=J5. En la instancia dada esto OCURRE. Por lo tanto la segunda subconsulta da FILAS y se **acumula un FALSO** en el **NOT EXISTS** antepuesto a esta segunda subconsulta.
4. Se vuelve a la primera subconsulta y se pasa a la fila siguiente de donde se toma J# =J7.

5. Se vuelve a resolver la **segunda subconsulta**, ahora con los valores P3 y J7. En la instancia dada esto TAMBIÉN OCURRE. Por lo tanto la segunda subconsulta da FILAS, por lo que se **agrega otro FALSO** en el **NOT EXISTS** antepuesto a esta segunda subconsulta.
6. Como se terminaron las filas de la primera subconsulta, se cierra la ejecución de la segunda subconsulta, se aplica un OR de los resultados obtenidos en 3 y 5, lo cual arroja un resultado SIN FILAS (en 3 y en 5 se obtuvieron FALSOS ⇒ no hay filas).
7. Quiere decir entonces que en la primer subconsulta la aplicación del NOT EXISTS resulta en **VERDADERO** y el valor de **P#= P3 APARECERÁ EN LA RESPUESTA**.

Lo mismo pasará cuando se evalúe P5.

¿Qué pasa si una parte es provista a uno de los proyectos (por ejemplo J7) y no al otro? Es decir, no se cumple el TODOS los proyectos de Londres. Por ejemplo, el caso de P6 que es suministrado a J7, pero no a J5. La parte de la tabla SPJ donde ocurre esto (**P#=P6**) es la siguiente:

Nro Fila	S#	P#	J#	CANT
13	S4	P6	J3	300
14	S4	P6	J7	300
24	S5	P6	J4	500

Repetiré los pasos:

1. Se abre UNO y nos ubicamos en la Fila 13 (S#='S4', P# = P6, J# = J3). El valor que vamos a utilizar es el de P#.
2. Primera subconsulta: se abre la tabla J y se aplica la condición CIUDAD = 'Londres' para quedarse únicamente con las filas correspondientes a J5 y J7. Se toma la primera fila que quedó y que corresponde a J# = J5.
3. **Segunda subconsulta**: se abre la tabla DOS y se resuelve con los valores P6 y J5. Esto es, evaluar en la tabla DOS si hay filas donde P#=P3 y J#=J5. En la instancia dada esto NO OCURRE. Por lo tanto la segunda subconsulta no da FILAS y se **acumula un VERDADERO** en el **NOT EXISTS**.
4. Se vuelve a la primera subconsulta y se pasa a la fila siguiente de donde se toma J# =J7.
5. Se vuelve a resolver la **segunda subconsulta**, ahora con los valores P6 y J7. En la instancia dada esto OCURRE. Por lo tanto la segunda subconsulta da FILAS, por lo que se **agrega un FALSO** en el **NOT EXISTS**.
6. Como se terminaron las filas de la primera subconsulta, se cierra la ejecución de la segunda subconsulta, se aplica un OR de los resultados obtenidos en 3 y 5, lo cual arroja un resultado que TIENE FILAS (una en este ejemplo, por el VERDADERO obtenido en el paso 3).
7. Quiere decir entonces que la **primer subconsulta da filas**, y la aplicación del **NOT EXISTS** resulta en **FALSO** y el valor de **P#= P6 NO APARECERÁ EN LA RESPUESTA**.

Observación:

- Una gran **desventaja** de esta forma es que la segunda subconsulta, se ejecuta muchas veces. Para este ejemplo, donde se tienen muy pocos datos (pocas filas), si SPJ tiene 24 filas y se aplica la restricción en J a un valor de CIUDAD que arroja sólo dos filas, la segunda subconsulta se ejecuta 48 veces!! Si no se aplicara la condición en J a los valores de CIUDAD (en el WHERE de la primera subconsulta), como la tabla J tiene 7 filas, la segunda subconsulta se ejecutará 24*7 = 168 veces. Para un caso más real, una tabla de trabajos (como J) puede tener 5000 filas y una tabla de suministros (como SPJ) puede tener 150000 filas, esto producirá que la segunda subconsulta en el peor de los casos, si no ponemos algunas condiciones a estas J y SPJ, se ejecute 750 millones de veces!!! ⇒ pésima performance!
  Una forma de mejorar esto es pensar con nociones de cálculo relacional (qué quiero y NO cómo lo resuelvo) y esto nos lleva a la Forma 2, que será más eficiente y con mejor performance.

Forma 2: utilizando GROUP BY y condiciones HAVING a los grupos.

```
SELECT P#  FROM SPJ, J
  WHERE SPJ.J# = J.J#  AND   J.CIUDAD = 'Londres'
  GROUP BY P#
  HAVING  COUNT (DISTINCT SPJ.J#) =
                    (SELECT COUNT(J#) FROM J WHERE CIUDAD = 'Londres');
```

Funcionamiento de esta consulta:

1. Abre las tablas SPJ y J y hace la reunión (join) de ambas, y luego aplica la condición de CIUDAD = 'Londres'l
2. En esta tabla temporaria con esquema (SPJ.S#, SPJ.P#, SPJ.J#, SPJ.CANT, J.J#, J.JNOMBRE, J.CIUDAD), hace el **agrupado** por P#. Esto significa que generará un conjunto de tablas temporarias con el mismo esquema pero cada una tendrá en sus filas el mismo valor para el campo P#. Esto es: habrá una tabla temporaria X que contendrá todas las filas donde el valor de P# sea P1, habrá otra tabla temporaria Y que contendrá todas las filas donde el valor de P# sea P2, y así hasta P6.

Aclaración: como los nombres de los campos S#, P#, CANT, JNOMBRE y CIUDAD corresponden a una sola de las tablas abiertas en el FROM, no utilizo el nombre de la tabla (SPJ o J) para desambiguar el nombre del campo en la escritura de la consulta.

3. En cada grupo (en cada tabla temporaria), se evalúa la condición pedida en el **HAVING**: que cuente cuántos valores distintos de J# hay COUNT (DISTINCT SPJ.J#) y lo compare con la cantidad de filas de J donde se verifique el resultado provisto por la consulta (SELECT COUNT(J#) FROM J WHERE CIUDAD = 'Londres').
4. Si se cumple la igualdad, muestra el valor de P#.

Observaciones:

- Entonces, esta forma de resolver el TODOS lo que hace es: contar cuántos proyectos están ubicados en Londres (un número que conserva en memoria), arma una tabla que tiene los datos a ver y los datos utilizar (con la reunión), filtra esta tabla dejando sólo las filas donde la ciudad del proyecto sea Londres, y se fija para cada valor de P# si la cantidad de valores de J# coincide o no con la cuenta obtenida antes.
- La condición **CIUDAD = 'Londres'** se debe colocar tanto en la consulta como en la subconsulta de comparación del HAVING. Si no se coloca esta condición en ambos lugares, los resultados serán erróneos. Pensarlo.
- COUNT (**DISTINCT** SPJ.J#): Se debe usar el DISTINCT aquí porque dada la forma de la tabla SPJ, una misma parte puede ser provista al mismo proyecto por varios proveedores, con lo cual habrá varias filas donde se repiten los valores de (P#, J#); como el COUNT cuenta filas si no se coloca el DISTINCT el resultado será erróneo. Por ejemplo: para P3 es suministrado al proyecto J1 por los proveedores S2 y S3, esto significa que el COUNT() de los proyectos será 2, lo cual es falso porque debe contar solamente 1 proyecto.

II. Obtener los números de los proveedores que suministren la misma parte a todos los proyectos.

```
SELECT  DISTINCT S#       FROM    SPJ UNO  WHERE EXISTS
        ( SELECT   P# FROM   SPJ DOS   WHERE  NOT EXISTS
                ( SELECT  J#  FROM  J    WHERE  NOT EXISTS
                        ( SELECT * FROM SPJ TRES WHERE
                                TRES.S# = UNO.S#   AND   TRES.P# = DOS.P#
                                AND      TRES.J# = J.J# ) ) ) ;
```

Este SELECT puede enunciarse: "obtener todos los proveedores (UNO.S#) tales que exista una parte (DOS.P#) tal que no exista ningún proyecto (J.J#) tal que el proveedor no suministre la parte al proyecto. Es decir: los proveedores tales que exista alguna parte suministrada por ellos a todos los proyectos.
Notar el uso de "SELECT P# " y "SELECT J# " en dos de las referencias NOT EXISTS. "SELECT *" no sería incorrecto, pero "SELECT P# " se acerca más a la formulación intuitiva: debe existir una parte (identificada con un número de parte), no sólo una fila en la tabla de envíos.

III. Obtener los números de los proyectos a los cuales se suministren por lo menos todas las partes suministradas por el proveedor S1.

```
SELECT   DISTINCT J#       FROM   SPJ   UNO
WHERE   NOT EXISTS  (SELECT  P#  FROM  SPJ   DOS
                WHERE    DOS.S# = 'S1'
                    AND   NOT EXISTS   ( SELECT *   FROM   SPJ   TRES
                                                WHERE TRES.P# = DOS.P#
                                                AND   TRES.J# = UNO.J# ) ) ;
```

*** VOLVEMOS A LOS ENUNCIADOS DE LA PRÁCTICA ***

Unión

32. Construir una lista ordenada de todas las ciudades en las cuales esté situado por lo menos un proveedor, una parte o un proyecto.

```
SELECT CIUDAD FROM S
        UNION
SELECT CIUDAD FROM P
        UNION
SELECT CIUDAD FROM J
        ORDER BY 1 ;
```

Toda cláusula ORDER BY en una proposición SELECT donde haya UNION deberá identificar la/s columna/s de ordenamiento por su posición ordinal, no por nombre, ya que las columnas de resultado en una unión SQL se consideran anónimas.

Operaciones de actualización

33. Cambiar a gris el color de todas las partes rojas.

```
UPDATE  P
        SET  COLOR = 'Gris'   WHERE  COLOR = 'Rojo'  ;
```

34. Eliminar todos los proyectos para los cuales no haya envíos.

```
DELETE
        FROM   J    WHERE   J#   NOT IN   ( SELECT    J#  FROM SPJ ) ;
```

35. Insertar un nuevo proveedor (S10) en la tabla S. El nombre y la ciudad son Salazar y Nueva York, respectivamente; la situación no se conoce todavía.

```
INSERT
    INTO   S ( S#, SNOMBRE, CIUDAD)   VALUES  ('S10', 'Salazar', 'Nueva York') ;
```

O bien:

```
INSERT
INTO   S ( S#, SNOMBRE, SITUACION, CIUDAD)
       VALUES ('S10', 'Salazar', NULL, 'Nueva York')  ;
```

----- ooooo -----

Resolver el ejercicio I de la Práctica 05: Ejercicios adicionales

Este es un ejercicio tipo para las evaluaciones (parciales y/o finales).

I. Considerando la base de datos compuesta por las siguientes relaciones:

COMPETICION (P#, DESCRIPCION, CATEGORIA)
CLUB (C#, NOM_C, PRESUPUESTO)
PARTICIPACION (C#, P#, PUESTO)

Dar soluciones en SQL y en AR a las siguientes consultas:

1. Obtener los nombres de los clubes con presupuesto mayor que 5 millones y que hayan participado en competiciones de categoría igual a 2.

2. Obtener los nombres de los clubes que **sólo han conseguido el primer puesto**.

3. Obtener los nombres de los clubes que han participado en **todas las competiciones**.

4. Obtener los nombres de los clubes que han participado en las **competiciones P1 y P2 (**en ambas).

5. Obtener el nombre y presupuesto de los clubes que **no han conseguido un primer puesto**.