PRÁCTICA 1

Sistemas de Gestión de Bases de Datos

18 OCTUBRE

PRÁCTICA 1 – EJERCICIO 8
Mario Ventura Burgos 43223476-J
Grado en Ingeniería Informática (GIN 3)
CURSO 2023-2024

1. CÓDIGO Y RESULTADO

Abriremos la herramienta de Query Tool en pgAdmin (aunque también se puede hacer en sql shell) y añadimos “EXPLAIN” al inicio de la consulta 2 del examen cuya solución se encuentra en el archivo mvb135_3.sql y obtenemos lo siguiente:

INPUT:

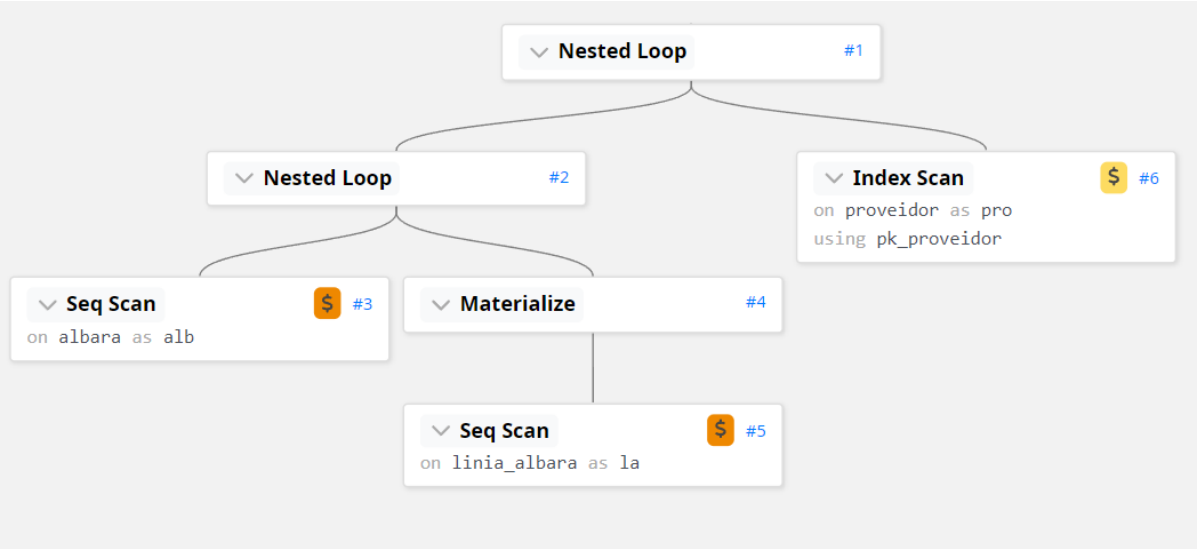
```
SELECT ALB.codi, PRO.nom
FROM Albara ALB
    JOIN Proveidor PRO
    ON PRO.nif = ALB.nif_pro    /*Hay que entrar en la tabla ya que se quiere algo de
ella*/
        JOIN Linia_ALbara LA
        ON LA.codi_alb = ALB.codi
        AND LA.preu IS NULL     /*Sin especificar precio = columna precio con valor
NULL*/
WHERE ALB.facturat = 'S';       /*Facturados = columna facturat con valor 'S'*/
```

OUTPUT:

```
Nested Loop  (cost=0.15..46.24 rows=1 width=130)
  ->  Nested Loop  (cost=0.00..40.05 rows=1 width=52)
        Join Filter: (alb.codi = la.codi_alb)
        ->  Seq Scan on albara alb  (cost=0.00..21.00 rows=4 width=52)
              Filter: ((facturat)::text = 'S'::text)
        ->  Materialize  (cost=0.00..18.82 rows=4 width=12)
              ->  Seq Scan on linia_albara la  (cost=0.00..18.80 rows=4 width=12)
                    Filter: (preu IS NULL)
  ->  Index Scan using pk_proveidor on proveidor pro  (cost=0.15..6.17 rows=1 width=158)
        Index Cond: ((nif)::text = (alb.nif_pro)::text)
```

2. ANÁLISIS DEL QUERY PLAN

Al igual que en el caso anterior, el plan de ejecución está organizado en una estructura jerárquica que se muestra mediante tabulaciones en el output. Sin embargo, se facilita la comprensión y se hace más legible si se representa en forma de árbol. El árbol obtenido para este plan de ejecución es el siguiente:



Como se puede apreciar, el árbol que representa el plan de ejecución cuenta con 6 nodos, representados por pgAdmin (o, en su defecto, sql shell) mediante las 10 líneas obtenidas como output. Analicemos el resultado nodo por nodo:

1. **NESTED LOOP (nodo #1)** → La línea 1 del plan de ejecución. Realmente es lo último que se hará. Representa la operación de JOIN entre las tablas Albara y Proveidor. Es un bucle anidado que se espera que tenga un coste de 46,24 **(cost=0.15..46.24).** Se espera que devuelva una fila **(rows=1)** con anchura de 160 bytes **(width=130)**

2. **NESTED LOOP (nodo #2)** → Las líneas 2 y 3 del Query Plan. Es un bucle anidado que tendrá lugar dentro del bucle del nodo 1. Se encarga de unir los resultados de las operaciones anteriores y representa la operación de JOIN entre las tablas Proveidor y Linia_Albara. El criterio de unión es el establecido en el Join Filter: **Join Filter: (alb.codi = la.codi_alb)**, que se encuentra en la línea 3 del plan de ejecución obtenido.
   Los valores de coste estimado, número de filas y anchura son: **cost=0.00..40.05 rows=1 width=52**

3. **SEQ SCAN (nodo #3)** → Las líneas 4 y 5 del plan de ejecución. Este nodo representa un escaneo secuencial en la tabla Albara en el que se aplica un filtro en la columna *facturat*, que se busca que tenga el valor "S" (representando que el albarán ha sido facturado). El hecho de

aplicar este filtro hace que se reduzca la cantidad de filas escaneadas ya que todos los albaranes que tengan valor *facturat* = "N" se ignorarán. Esta operación se realiza dentro del Nested Loop del nodo 2.
El coste estimado, la cantidad de filas y la anchura son los siguientes: **cost=0.00..21.00 rows=4 width=52** y el filtro: **Filter: ((facturat)::text = 'S'::text)**

4. **MATERIALIZE** → La línea 6 del Query Plan. Este paso tiene lugar "dentro" del Nested Loop del nodo 2 y consiste en materializar temporalmente el resultado de la operación que lo precede para su uso en la operación de JOIN (la operación que lo precede es el nodo 5, que explicaremos a continuación).
   El coste de esto es de **cost=0.00..18.82,** se estima una cantidad de filas **rows=4** con una anchura de **width=12** bytes

5. **SEQ SCAN (nodo #5)** → Las líneas 7 y 8 del Query Plan. En este nodo se realiza un escaneo secuencial sobre la tabla Linia_albara (LA) con un filtro en la columna preu, ya que se busca todos los albaranes facturados que tienen alguna línea sin especificar el precio. Este filtro se especifica en la línea 8: **Filter: (preu IS NULL)**.
   Esta operación precede al Materalize del nodo 4, que materializa temporalmente el resultado de este escaneo secuencial para usarlo el JOIN.
   El coste de este escaneo secuencial es de **cost=0.00..18.80**, se espera una cantidad de **rows=4** filas y con una anchura en bytes de **width=12**.

6. **INDEX SCAN**→ La novena y la décima línea del plan de ejecución. En este nodo se realiza un Index Scan en la tabla Proveidor, es decir, escaneo utilizando un índice. La condición de índice se basa en la igualdad de las columnas nif en la tabla Proveidor (PRO) y nif_pro en la tabla Albara (ALB). Para realizar esta operación, se hace uso de la CONSTRAINT "pk_proveidor", definida en el código SQL de creación de las tablas de la base de datos (ejercicio mvb135_1.sql).
   Este escaneo se realiza dentro del Nested Loop del nodo 1 y, por tanto, las condiciones del bucle anidado también se contemplarán. Representa la búsqueda mediante índice del proveedor asociado con la fila en Albara
   El coste de este escaneo de índice es de **cost=0.15..6.17** unidades de coste, y se estima obtener una cantidad de **rows=1** filas con una anchura en bytes de **width=158**.

En resumen, PostgreSQL ha usado dos bucles anidados en los nodos 1 y 2, siendo estos dos bucles las operaciones de mayor coste de la consulta (el primer Loop el que más). Dentro del segundo Loop, que a su vez está dentro del primer Loop, se realizan dos escaneos secuenciales: uno en la tabla Albara y otro en Linia_Albara, y se unen mediante un Materialize. Por último, haciendo uso de la CONSTRAINT pk_proveidor, se realiza un escaneo de índice sobre la tabla Proveidor bajo unas condiciones de unión (Index Cond) especificadas en la última línea del plan de ejecución.