Informe tarea 4

Grupo 51 [Paulo Sánchez - Samuel Noble]

Actividad 2

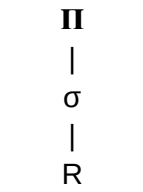
Ejercicio 2.3)

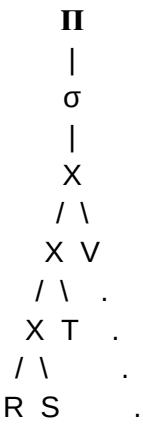
Parte 1

a)

La consulta 1 y 2 traen los nombres de las empresas y nombres de los responsables de las empresas relacionados con estas primeras, que cumplen que pertenecen al rubro ‘Interactive ␣ entertainment’.

b)



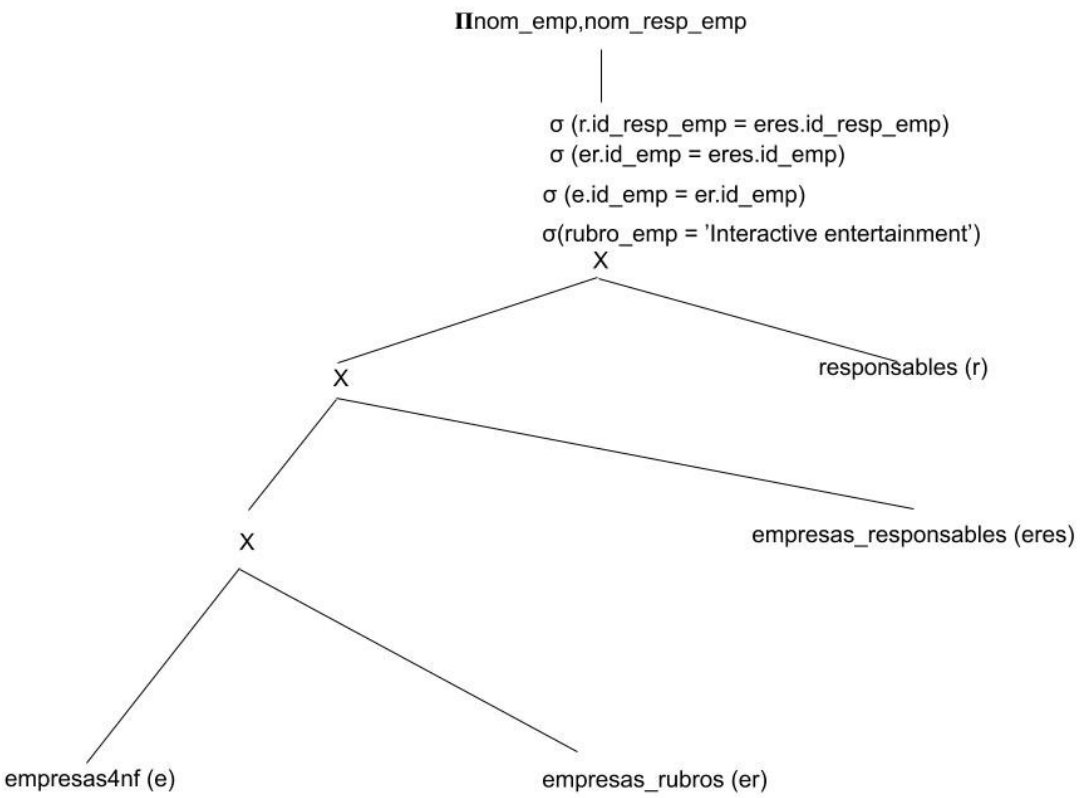


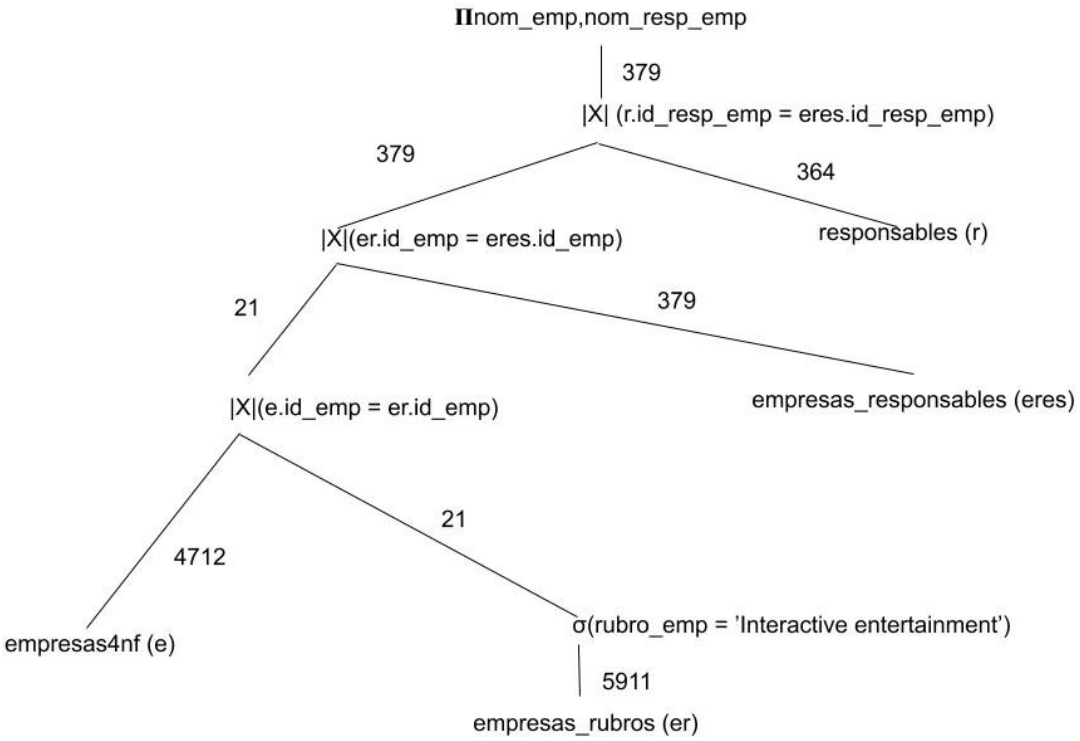
c)

Arbol canonico de la primera consulta

Π nom_emp,nom_resp_emp
| 26
σ(rubro_emp = ’ Interactive entertainment ’)
| 9210
empresas

Arbol canonico de la segunda consulta





d)

En el arbol canónico de la consulta uno no hay nada para hacer.

En el árbol canónico de la consulta 2: En principio empresas4nf no se podría intercambiar por empresas_rubros porque si bien luego de la selección en empresas_rubro quedan 21 tuplas, estas 21 tuplas no están al nivel de las hojas y es por esto que no se podrían cambiar de lugar. Además tampoco se pueden mover otras tablas de lugar ya que resulta imposible hacer los joins correspondientes.

Parte 2

a)

Para la consulta 1, se hará:

- Una búsqueda linea por la condición *rubro_emp = 'Interactive entertainment'* dado que no existe un índice nombrado para esta columna, en la columna empresas_rubros.

Para la consulta 2, se hará:

- Una búsqueda lineal por la condición *rubro_emp = 'Interactive entertainment'* dado que no existe un índice nombrado para esta columna, en la columna empresas_rubros.

- Utilizaremos Index Join para el join entre empresas_rubros y empresas4nf debido a que id_emp es clave en empresas4nf.
- Loop anidado por registros en segunda instancia para el join entre el resultado del punto anterior y empresas_responsables.
- Index Join para el join entre el resultado del punto anterior y responsables.

b)

Pareciera ser más eficiente la segunda, porque aunque tiene más operaciones intermedias para conseguir el mismo resultado, las búsqueda más costosa es la inicial, la cual contiene menos registros que la búsqueda lineal en la consulta 1.

Actividad 3

Ejercicio 3.2)

a)

ANALIZE: Recoge información estadística de columnas, tablas y bases de datos (según le especifiquemos) para utilizarla en mejorar los planes de ejecución de consultas sobre estas. Por ejemplo:

```
ANALYZE empresas;
```

Analiza la tabla empresas dentro de la base de datos conectada.

Es importante señalar que este análisis no se hace trabajando con índices, sino que con el contenido directo de las tablas. No bloquea la tabla.

EXPLAIN: Retorna el plan de ejecución que se genera al correr la consulta siguiente.
Suma a esta información del plan, el costo inicial (o sea, antes de encontrar la primera fila) y el costo final (luego de encontrar la última).
Además de esta información, tiene algunas posibilidades de agregarle opciones para que complemente la información de costos, tiempos y análisis.

```
EXPLAIN {query};
```

EXPLAIN ANALYZE: El análisis agregado al explain ocasiona que se ejecute la consulta, y luego es analice la misma también en el caso real del procesamiento..
Por ejemplo, para el caso de querer analizar un INSERT, UPDATE, DELETE, puedes encerrar el análisis (dado que éste ejecutará la actualización), en una transacción que finaliza con un rollback de lo hecho:

```
BEGIN;
     EXPLAIN ANALYZE {query};
ROLLBACK;
```

b)

El costo mostrado en el plan de ejecución es en accesos a las páginas de disco.
Además, el plan muestra los accesos que se hacen y cómo se realiza la consulta deseada (que métodos, índices y más utilizan).

c)

"Seq Scan on empresas (cost=0.00..1180.13 rows=26 width=22)"
" Filter: ((rubro_emp)::text = 'Interactive entertainment'::text)"
Costo total: 1180.13

"Nested Loop (cost=141.70..159.14 rows=2 width=23)"
" -> Nested Loop (cost=141.43..158.34 rows=2 width=60)"
"   Join Filter: ((er.id_emp)::text = (e.id_emp)::text)"
"   -> Hash Join (cost=141.15..151.93 rows=2 width=132)"
"   Hash Cond: ((eres.id_emp)::text = (er.id_emp)::text)"
"   -> Seq Scan on empresas_responsables eres (cost=0.00..9.79 rows=379 width=87)"
"   -> Hash (cost=140.89..140.89 rows=21 width=45)"
"   -> Seq Scan on empresas_rubros er (cost=0.00..140.89 rows=21 width=45)"
"   Filter: ((rubro_emp)::text = 'Interactive entertainment'::text)"
"   -> Index Scan using empresas4nf_pkey on empresas4nf e (cost=0.28..3.19 rows=1 width=62)"
"   Index Cond: ((id_emp)::text = (eres.id_emp)::text)"
" -> Index Scan using responsables_pkey on responsables r (cost=0.27..0.40 rows=1 width=49)"
"   Index Cond: ((id_resp_emp)::text = (eres.id_resp_emp)::text)"
Costo total: 764,57

Es menor el costo de la segunda consulta.

d)

Para la primer consulta:

- El método es una búsqueda secuencial sobre la tabla empresas (dado que no hay índice sobre esa columna).

Para la segunda consulta:

- Utiliza el método Nested Loop que es el más básico de PostgreSQL para analizar los joins. Éste recorre todas las filas de ambas tablas encontrando las coincidencias de la condición referida en el Join Filter.
- También utiliza Hash Join que es cuando una de las tablas está dentro de un hash en memoria y se toman las filas de la otra tabla para filtrar (según la condición dada en Hash Cond) contra el hash de la primera tabla. Más eficiente que el anterior.
- Seq Scan, que ya fue explicado en la primera consulta.
- Index Scan, que es un escaneo de las filas utilizando el índice de utilidad para la consulta (por ejemplo, responsables_pkey ó empresas4nf_pkey) que buscará las filas bajo la condición dada en Index Cond.

e)

Al deshabilitar el escaneo secuencial, vemos los siguientes planes:

```
"Seq Scan on empresas  (cost=10000000000.00..10000001180.13 rows=26 width=22)"
"  Filter: ((rubro_emp)::text = 'Interactive entertainment'::text)"
```

```
"Nested Loop  (cost=1.11..395.51 rows=2 width=23)"
"  ->  Nested Loop  (cost=0.84..394.72 rows=2 width=60)"
"        Join Filter: ((er.id_emp)::text = (e.id_emp)::text)"
"        ->  Merge Join  (cost=0.55..388.31 rows=2 width=132)"
"              Merge Cond: ((eres.id_emp)::text = (er.id_emp)::text)"
"              ->  Index Only Scan using empresas_responsables_pkey on
empresas_responsables eres  (cost=0.27..53.79 rows=379 width=87)"
"              ->  Index Only Scan using empresas_rubros_pkey on empresas_rubros er
(cost=0.28..333.50 rows=21 width=45)"
"                    Index Cond: (rubro_emp = 'Interactive entertainment'::text)"
"        ->  Index Scan using empresas4nf_pkey on empresas4nf e  (cost=0.28..3.19 rows=1
width=62)"
"              Index Cond: ((id_emp)::text = (eres.id_emp)::text)"
"  ->  Index Scan using responsables_pkey on responsables r  (cost=0.27..0.40 rows=1
width=49)"
"        Index Cond: ((id_resp_emp)::text = (eres.id_resp_emp)::text)"
```

Donde en la primer consulta igualmente se realizó un escaneo secuencial (dado que no tiene otra opción (pero con un costo infinito); mientras que en la segunda consulta, se mudan los **Hash Join** por **Merge Join** y los **Seq Scan** por **Index Only Scan**, que es un escaneo basado en el índice de esa tabla.
En este caso, el costo pasa a ser mayor en la anterior oportunidad.
El cambio es dado ya que no puede utilizar dicho método de búsqueda, por lo que se refiere a las siguiente mejor opción para hacerlo.