Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
Segundo Semestre de 2018

IIC 2413 – Bases de Datos
Interrogación 2 Rúbrica

Pregunta 1: Procedimientos almacenados

Esquema:

```
R(a int, b int, PRIMARY KEY(a, b))
S(b int, c int, PRIMARY KEY(b, c))
```

Consulta:

```
SELECT *
FROM R LEFT JOIN S
ON R.b = S.b;
```

Un procedimiento almacenado que puede servir es el siguiente:

```
CREATE OR REPLACE FUNCTION
outer_a_mano ()
RETURNS TABLE (a int, b1 int, b2 int, c int) AS $$
DECLARE
  tuple_r RECORD;
  tuple_s RECORD;
  valor_en_s BOOLEAN;
BEGIN
  CREATE TABLE outer_table(a int, b1 int, b2 int, c int);

  FOR tuple_r IN SELECT * FROM R LOOP
    valor_en_s := FALSE;
    FOR tuple_s IN SELECT * FROM S LOOP
      IF tuple_r.b = tuple_s.b THEN
        INSERT INTO outer_table VALUES(tuple_r.a, tuple_r.b, tuple_s.b, tuple_s.c);
        valor_en_s := TRUE;
      END IF;
    END LOOP;
    IF NOT(valor_en_s) THEN
        INSERT INTO outer_table VALUES(tuple_r.a, tuple_r.b, NULL, NULL);
      END IF;
  END LOOP;

  RETURN QUERY SELECT * FROM outer_table;

  DROP TABLE outer_table;

END
$$ language plpgsql
```

Pregunta 2: Recursión y Programación + SQL

El esquema era:

```
Caminos(
  ciudad_origen VARCHAR(100),
  ciudad_destino VARCHAR(100),
  nombre_camino VARCHAR(100)
)
```

Este indica los pares de ciudades conectadas entre sí, y el nombre del camino que las conecta. La siguiente consulta:

```
WITH RECURSIVE Alcanzo(co, cd, n_camino) AS (
  SELECT * FROM Caminos
  UNION
  SELECT C.ciudad_origen, A.cd, A.n_camino
  FROM Caminos C, Alcanzo A
  WHERE C.ciudad_destino = A.co AND C.nombre_camino = A.n_camino
)
SELECT * FROM Alcanzo;
```

Representa todas las ciudades alcanzables entre si, en que se puede llegar utilizando siempre el mismo camino. El algoritmo en python que la representa puede ser el siguiente:

```
1  import psycopg2
2
3  conn = psycopg2.connect(
4    database = "mibase",
5    user = "miusuario",
6    password = "mipass",
7    host = "localhost",
8    port = "5432")
9
10 cur = conn.cursor()
11 cur2 = conn.cursor()
12 cur.execute("CREATE TABLE Alcanzo(co VARCHAR(100), \
13   cd VARCHAR(100), \
14   n_camino VARCHAR(100))")
15
16 cur.execute("INSERT INTO Alcanzo (SELECT * FROM Caminos)")
17
18 while True:
19     cur.execute("SELECT COUNT(*) FROM Alcanzo")
20     count_previo = cur.fetchone()[0]
21
22     cur.execute("SELECT C.ciudad_origen, A.cd, A.n_camino \
23       FROM Caminos C, Alcanzo A \
24       WHERE C.ciudad_destino = A.co AND C.nombre_camino = A.n_camino \
25       AND (C.ciudad_origen, A.cd, A.n_camino) NOT IN \
26         (SELECT * FROM Alcanzo) \
27     ")
28
29     insert_query = "INSERT INTO Alcanzo Values( \
30       %(co)s, %(cd)s, %(n_camino)s \
31     )"
32
33     t = cur.fetchone()
34     while t:
35         cur2.execute(insert_query,
36           {"co": t[0], "cd": t[1], "n_camino": t[2]}
37         )
38         t = cur.fetchone()
39
40     cur.execute("SELECT COUNT(*) FROM Alcanzo")
41     count_post = cur.fetchone()[0]
42
```

```
43      if count_previo == count_post:
44          break
45
46
47  cur.execute("SELECT * FROM Alcanzo")
48
49  t = cur.fetchone()
50  while t:
51      print(t)
52      t = cur.fetchone()
53
54  cur.execute("DROP TABLE Alcanzo")
55
56  conn.commit()
57  conn.close()
```

Bonus Los nombres de las autopistas son:

- Ruta 5: Panamericana, que conecta Chile desde el norte hasta Chiloé.
- Ruta 7: Carretera Austral, que va desde Puerto Montt hasta Villa O'Higgins.
- Ruta 9: Carretera del Fin del Mundo, que va desde Torres del Paine hasta Punta Arenas y más allá.
- Ruta 78: Autopista del Sol, que conecta Santiago con San Antonio.

Pregunta 3: Índices y algoritmos

Índices [2.5 pts]

Los costos eran los siguientes:

1. h, pues sólo había que bajar por el árbol.
2. $\frac{1{,}000{,}000}{100} = 1{,}000$, porque hay que recorrer toda la relación. Si consideramos que la relación está solamente almacenada en el B+Tree, una posible respuesta también es $\frac{1{,}000{,}000}{100*0{,}75}$.
3. $1 + 1 = 2$, porque hay que ir al bucket y luego ir a buscar la dirección que el puntero me señala.
4. $h + \frac{10{,}000}{100*0{,}75}$, pues hay que bajar por el árbol y recorrer las hojas al 75 %.
5. $10 + \alpha$, porque debo preguntar por 10 edades y visitaré 10 buckets cada uno en una página. α es el número de personas que tienen entre 20 y 30 años, ya que por cada una de ellas recibí un puntero y tuve que ir al disco duro.

Respecto a tener un índice *clustered* para las edades, esto efectivamente aceleraría las consultas, pero significa volver a guardar la relación, por lo que estaría usando el doble de espacio.

Algoritmos [3.5 pts]

Considerando las relaciones `R(a int, b int)` y `S(b int, c int)`, sabemos que al tener un B+Tree sobre su llave primaria, entonces ese atributo está ordenado, por lo que siempre podemos avanzar a la hora de iterar cuando se está haciendo el join:

```
1  import psycopg2
2
3  conn = psycopg2.connect(
4    database = "mibase",
5    user = "miuser",
6    password = "mipass",
7    host = "localhost",
8    port = "5432")
```

```
9
10  cur = conn.cursor()
11  cur2 = conn.cursor()
12
13  cur.execute("SELECT * FROM R ORDER BY b")
14  cur2.execute("SELECT * FROM S ORDER BY b")
15
16  t1 = cur.fetchone()
17  t2 = cur2.fetchone()
18
19  while t1 and t2:
20      if t1[1] == t2[0]:
21          print("{} - {}".format(t1, t2))
22          t1 = cur.fetchone()
23          t2 = cur2.fetchone()
24      elif t1[1] < t2[0]:
25          t1 = cur.fetchone()
26      else:
27          t2 = cur2.fetchone()
28
29  conn.commit()
30  conn.close()
```

El costo de hacer un join de este estilo sería $R_p + S_p$. También es posible considerar la altura de cada árbol en la respuesta, pero no es necesario ya que este valor es muy pequeño en comparación al número de páginas que potencialmente tiene cada relación.