CC3201-1 Bases de Datos
Profesores de Cátedra: Matías Toro, Claudio Gutierrez
Estudiantes: Alexander Sacchetti, Andrés Calderón Guardia, Felipe Fierro

Laboratorio 7

September 16, 2023

P1)

(a) Encontrar todas las tuplas de *R*:

- Sin ningun índice:
Se extrae por bloques secuencialmente.

$\lceil \frac{1000000}{B} \rceil$

- B+Tree *unclustered*:
Primero tenemos que recorrer la altura del árbol, lo cual genera h accesos a bloque, luego el segundo término es para acceder a todos los bloques que contienen los punteros, y el último término es para acceder a todos los punteros.

$h + \lceil \frac{1000000}{P} \rceil + 1000000$

- B+Tree *clustered*:
Cada bloque en las hojas del árbol está ocupado al 60% de B.

$\lceil \frac{1000000}{B*0.6} \rceil$

- Hash Index unclustered y clustered:
No se puede realizar esta operación ya que al usar una función de hash solo podemos encontrar valores específicos.

Indefinido

(b) Encontrar todas las tuplas de *R* tal que a<50:

- Sin ningún índice:
Dado que el archivo está desordenado, igualmente tendremos que recorrer todos los bloques.

$\lceil \frac{1000000}{B} \rceil$

- B+Tree *unclustered*:
Primero tenemos que recorrer la altura del árbol, lo cual genera h accesos a bloque, luego el segundo término es para acceder a todos los bloques que contienen los punteros, y el último término es para acceder a todos los punteros.

$h + \lceil \frac{1000000}{P} \rceil + 1000000$

- B+Tree *clustered*:
Cada bloque en las hojas del árbol está ocupado al 60% de B y como solo hay 50 relaciones que cumplen estar en el rango entonces el total pasa deun millón a 50.

$\lceil \frac{50}{B*0.6} \rceil$

- Hash Index unclustered y clustered:
No se puede realizar esta operación ya que al usar una función de hash solo podemos encontrar valores específicos.

Indefinido

(c) Encontrar todas las tuplas de *R* tal que a=50:

- Sin ningún índice:
Dado que el archivo está desordenado, igualmente tendremos que recorrer todos los bloques, pero podemos parar antes si es que el resultado se encuentra antes del final, dado que las llaves son únicas.

$\leq \lceil \frac{1000000}{B} \rceil$

- B+Tree *unclustered*:
Tenemos que recorrer la altura del árbol para hallar el puntero en el cual está ubicado el valor elegido, y luego accedemos al bloque que apunta el puntero.

h+1

- B+Tree *clustered*:
Solamente hay que recorrer la altura del árbol y en la hoja está la tupla que buscamos.

h

- Hash Index unclustered
Al aplicar la función de hash el resultado nos dice que página traer desde el disco para acceder al puntero, y luego desde este tenemos que acceder a su ubicación.

2

- Hash Index clustered:
Aplicando la función de hash el resultado nos da la página a traer desde el disco para acceder a la tupla.

1

P2)

(a) Índices:

En el resultado del comando las llaves primarias pese a que no se especifica, **suelen** estar *agrupadas* mientras que el resto de atributos estarían *desagrupados*.

Sin embargo, al realizar `EXPLAIN ANALYZE` si se ve en el resultado que cada consulta corresponde a btrees, por lo que para todos los casos los índices son árboles B ordenados y balanceados.

1. opt.pelicula:
   - "pelicula10000_pkey" PRIMARY KEY, btree (nombre, anho)
2. opt.actor:
   - "actor10000_pkey" PRIMARY KEY, btree (nombre)
3. opt.personaje:
   - "personaje10000_pkey" PRIMARY KEY, btree (a_nombre, p_nombre, p_anho, personaje)
4. opti.pelicula:
   - "pelicula10000_pkey" PRIMARY KEY, btree (nombre, anho)
   - "pelicula10000_calificacion" btree (calificacion)
   - "pelicula10000_nombre" btree (nombre)
   - "pelicula10000_votos" btree (votos)
5. opti.actor:
   - "actor10000_pkey" PRIMARY KEY, btree (nombre)
   - "actor10000_genero" btree (genero)
   - "actor10000_nombre" btree (nombre)
6. opti.personaje:

- "personaje10000_pkey" PRIMARY KEY, btree (a_nombre, p_nombre, p_anho, personaje)
- "personaje10000_anombre" btree (a_nombre)
- "personaje10000_panho" btree (p_anho)
- "personaje10000_pnombre" btree (p_nombre)
- "personaje10000_pnombreanho" btree (p_nombre, p_anho)

(b)

- Película favorita elegida: *Fight Club*

1. **opt 100 anidado:**

- Consulta:

```
EXPLAIN ANALYZE SELECT DISTINCT p_nombre, p_anho
FROM opt.personaje100
WHERE p_nombre != 'Fight Club'
AND a_nombre IN
(
  SELECT DISTINCT a_nombre
  FROM opt.personaje100
  WHERE p_nombre = 'Fight Club'
  AND p_anho = 1999);
```

- Planificación de consulta:

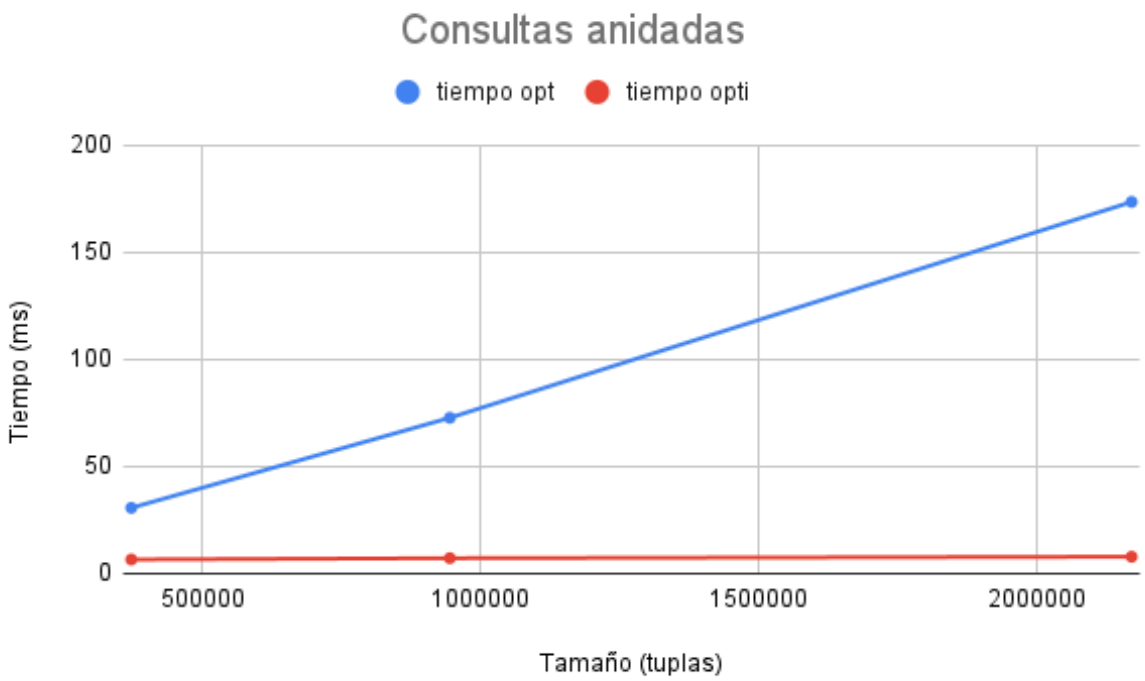
```
Unique (cost=35980.94..35981.04 rows=13 width=19) (actual time=136.696..139.276 rows=899 loops=1)
  -> Sort (cost=35980.94..35980.98 rows=13 width=19) (actual time=136.692..137.702 rows=1073 loops=1)
        Sort Key: personaje100.p_nombre, personaje100.p_anho
        Sort Method: quicksort Memory: 115kB
        -> Nested Loop (cost=35976.33..35980.70 rows=13 width=19)
                    (actual time=130.434..134.204 rows=1073 loops=1)
           -> Unique (cost=35975.90..35975.90 rows=1 width=16)
                    (actual time=130.425..130.838 rows=77 loops=1)
              -> Sort (cost=35975.90..35975.90 rows=1 width=16)
                       (actual time=130.406..130.594 rows=78 loops=1)
                 Sort Key: personaje100_1.a_nombre
                 Sort Method: quicksort Memory: 29kB
                 -> Gather (cost=1000.00..35975.89 rows=1 width=16)
                          (actual time=9.809..130.340 rows=78 loops=1)
                    Workers Planned: 2
                    Workers Launched: 2
                    -> Parallel Seq Scan on personaje100 personaje100_1
                     (cost=0.00..34975.79 rows=1 width=16) (actual time=7.961..124.526 rows=26 loops=3)
                          Filter: (((p_nombre)::text = “Fight Club”::text) AND (p_anho = 1999))
                          Rows Removed by Filter: 723483
           -> Index Only Scan using personaje100_pkey on personaje100 (cost=0.43..4.66 rows=13 width=35)
                                                                  (actual time=0.011..0.023 rows=14 loops=77)
              Index Cond: (a_nombre = (personaje100_1.a_nombre)::text)
              Heap Fetches: 0
```

2. **opti 100 anidada:**

- Planificación de la consulta:

```
HashAggregate (cost=13.33..13.46 rows=13 width=19) (actual time=5.208..5.998 rows=899 loops=1)
    Group Key: personaje100.p_nombre, personaje100.p_anho
    Batches: 1 Memory Usage: 129kB
    -> Nested Loop (cost=8.89..13.26 rows=13 width=19) (actual time=0.315..4.067 rows=1073 loops=1)
        -> Unique (cost=8.46..8.47 rows=1 width=16) (actual time=0.288..0.464 rows=77 loops=1)
            -> Sort (cost=8.46..8.46 rows=1 width=16) (actual time=0.286..0.344 rows=78 loops=1)
```

Sort Key: personaje100_1.a_nombre
Sort Method: quicksort Memory: 29kB
-> Index Scan using personaje100_pnombreanho on personaje100 personaje100_1
(cost=0.43..8.45 rows=1 width=16) (actual time=0.033..0.191 rows=78 loops=1)
Index Cond: (((p_nombre)::text = “Fight Club”::text) AND (p_anho = 1999))
-> Index Only Scan using personaje100_pkey on personaje100 (cost=0.43..4.66 rows=13 width=35)
(actual time=0.011..0.024 rows=14 loops=77)
Index Cond: (a_nombre = (personaje100_1.a_nombre)::text)
Heap Fetches: 0



3. **opt 100 no anidado:**

- Planificación de consulta:

Unique (cost=35981.05..35981.19 rows=18 width=19) (actual time=132.307..136.290 rows=899 loops=1)
-> Sort (cost=35981.05..35981.10 rows=18 width=19) (actual time=132.303..134.523 rows=1075 loops=1)
Sort Key: p2.p_nombre, p2.p_anho
Sort Method: quicksort Memory: 116kB
-> Nested Loop (cost=1000.43..35980.67 rows=18 width=19)
(actual time=12.474..131.128 rows=1075 loops=1)
-> Gather (cost=1000.00..35975.89 rows=1 width=16)
(actual time=12.425..126.852 rows=78 loops=1)
Workers Planned: 2
Workers Launched: 2
-> Parallel Seq Scan on personaje100 p1 (cost=0.00..34975.79 rows=1 width=16)
(actual time=8.094..124.441 rows=26 loops=3)
Filter: (((p_nombre)::text = “Fight Club”::text) AND (p_anho = 1999))
Rows Removed by Filter: 723483
-> Index Only Scan using personaje100_pkey on personaje100 p2
(cost=0.43..4.66 rows=13 width=35) (actual time=0.015..0.031 rows=14 loops=78)
Index Cond: (a_nombre = (p1.a_nombre)::text)
Heap Fetches: 0

4. **opti 100 no anidada:**

- Planificación de consulta:

```
HashAggregate (cost=13.32..13.48 rows=16 width=19) (actual time=8.575..9.949 rows=899 loops=1)
    Group Key: p2.p_nombre, p2.p_anho
    Batches: 1 Memory Usage: 129kB
    -> Nested Loop (cost=0.86..13.24 rows=16 width=19) (actual time=0.236..6.389 rows=1075 loops=1)
        -> Index Scan using personaje100_pnombreanho on personaje100 p1 (cost=0.43..8.45 rows=1 width=16)
                                                    (actual time=0.031..0.292 rows=78 loops=1)
            Index Cond: (((p_nombre)::text = "Fight Club"::text) AND (p_anho = 1999))
        -> Index Only Scan using personaje100_pkey on personaje100 p2 (cost=0.43..4.66 rows=13 width=35)
                                                    (actual time=0.016..0.035 rows=14 loops=78)
            Index Cond: (a_nombre = (p1.a_nombre)::text)
            Heap Fetches: 0
```