Universidade de São Paulo
Escola de Artes, Ciências e Humanidades
Disciplina: Laboratório de Banco de Dados
Profª Dra. Fátima Nunes.

Administração de Condomínio

Parte III - Artefato D

Fernando K. G. de Amorim – 10387644
João Guilherme da Costa Seike – 9784634
Lucas Pereira Castelo Branco – 10258772
Victor Gomes de O. M. Nicola – 9844881

São Paulo
2019

Objetivo do artefato D:

- **(Artefato do tipo texto)** Escolha **1 consulta** dentre as quatro definidas na Parte I do trabalho. Reescreva sua consulta usando duas estratégias equivalentes (veja a teoria estudada em sala de aula), portanto criando **2 novas consultas**, e execute novamente as análises de plano e custo. Verifique se o SGBD foi capaz de chegar no mesmo plano de consulta para execução de cada consulta equivalente. Analise o comportamento obtido (os mesmos itens de relatórios e *prints* do item (b) devem ser expostos aqui para fins de comparação). **O texto deve ser entregue em formato PDF no e-Disciplinas. O código deverá ser entregue em formato ASCII (.txt) no e-Disciplinas (ver observação em vermelho abaixo).**

Neste artefato, utilizamos uma das consultas da primeira parte do trabalho com intuito de reescrever-la a fim de gerar duas novas consultas que possuam o mesmo plano de consulta de execução para cada uma delas, posteriormente comparando se seus planos de execução foram os mesmos. A consulta utilizada para ser reescrita, foi a consulta que faz a contagem de marcas de veículos, limitada a 3 marcas de carros.

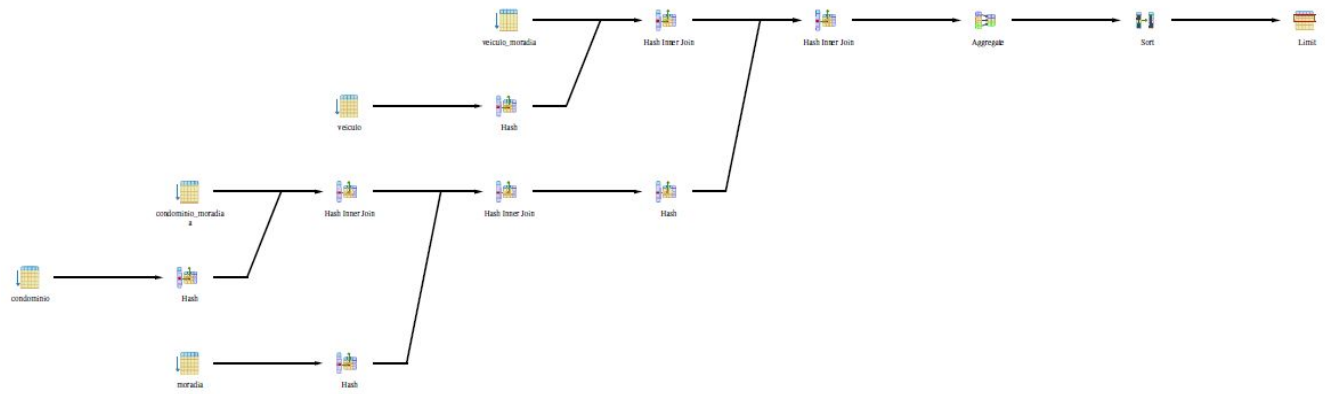
Consulta original:

```
SELECT v.marca, COUNT(v.*) FROM adm_condominio.Veiculo AS v
  INNER JOIN adm_condominio.Veiculo_Moradia AS vM ON vM.fk_id_veiculo = v.id_veiculo
  INNER JOIN adm_condominio.Moradia AS m ON m.id_moradia = vM.fk_id_moradia AND
m.tipo_moradia = 'a'
  INNER JOIN adm_condominio.Condominio_Moradia AS cM ON cM.fk_id_moradia = m.id_moradia
  INNER JOIN adm_condominio.Condominio AS c ON c.id_condominio = cM.fk_id_condominio AND
c.tipo_condominio = 'e'
  GROUP BY v.marca
  ORDER BY 2
  LIMIT 3;
```

Plano de execução da consulta original:

```
Limit  (cost=328.45..328.46 rows=3 width=16) (actual time=0.255..0.258 rows=3 loops=1)
  ->  Sort  (cost=328.45..328.55 rows=41 width=16) (actual time=0.254..0.254 rows=3 loops=1)
        Sort Key: (count(v.*))
        Sort Method: quicksort  Memory: 25kB
        ->  HashAggregate  (cost=327.51..327.92 rows=41 width=16) (actual time=0.244..0.245 rows=4 loops=1)
              Group Key: v.marca
              ->  Hash Join  (cost=59.65..229.16 rows=19671 width=76) (actual time=0.231..0.237 rows=4 loops=1)
                    Hash Cond: (vm.fk_id_moradia = m.id_moradia)
                    ->  Hash Join  (cost=2.80..41.45 rows=2260 width=80) (actual time=0.111..0.116 rows=20 loops=1)
                          Hash Cond: (vm.fk_id_veiculo = v.id_veiculo)
                          ->  Seq Scan on veiculo_moradia vm  (cost=0.00..32.60 rows=2260 width=8) (actual time=0.020..0.020 rows=20 loops=1)
                          ->  Hash  (cost=1.80..1.80 rows=80 width=80) (actual time=0.074..0.074 rows=80 loops=1)
                                Buckets: 1024  Batches: 1  Memory Usage: 17kB
                                ->  Seq Scan on veiculo v  (cost=0.00..1.80 rows=80 width=80) (actual time=0.020..0.050 rows=80 loops=1)
                    ->  Hash  (cost=47.06..47.06 rows=783 width=8) (actual time=0.108..0.108 rows=9 loops=1)
                          Buckets: 1024  Batches: 1  Memory Usage: 9kB
                          ->  Hash Join  (cost=5.20..47.06 rows=783 width=8) (actual time=0.094..0.104 rows=9 loops=1)
                                Hash Cond: (cm.fk_id_moradia = m.id_moradia)
                                ->  Hash Join  (cost=2.34..40.99 rows=1195 width=4) (actual time=0.051..0.060 rows=19 loops=1)
                                      Hash Cond: (cm.fk_id_condominio = c.id_condominio)
                                      ->  Seq Scan on condominio_moradia cm  (cost=0.00..32.60 rows=2260 width=8) (actual time=0.011..0.014 rows=40 loops=1)
                                      ->  Hash  (cost=1.88..1.88 rows=37 width=4) (actual time=0.025..0.025 rows=37 loops=1)
                                            Buckets: 1024  Batches: 1  Memory Usage: 10kB
                                            ->  Seq Scan on condominio c  (cost=0.00..1.88 rows=37 width=4) (actual time=0.010..0.018 rows=37 loops=1)
                                                  Filter: (tipo_condominio = 'e'::bpchar)
                                                  Rows Removed by Filter: 33
                                ->  Hash  (cost=2.13..2.13 rows=59 width=4) (actual time=0.033..0.033 rows=59 loops=1)
                                      Buckets: 1024  Batches: 1  Memory Usage: 11kB
                                      ->  Seq Scan on moradia m  (cost=0.00..2.13 rows=59 width=4) (actual time=0.012..0.021 rows=59 loops=1)
                                            Filter: (tipo_moradia = 'a'::bpchar)
                                            Rows Removed by Filter: 31
Planning Time: 0.590 ms
Execution Time: 0.431 ms
```

Imagem do plano de execução da consulta original:



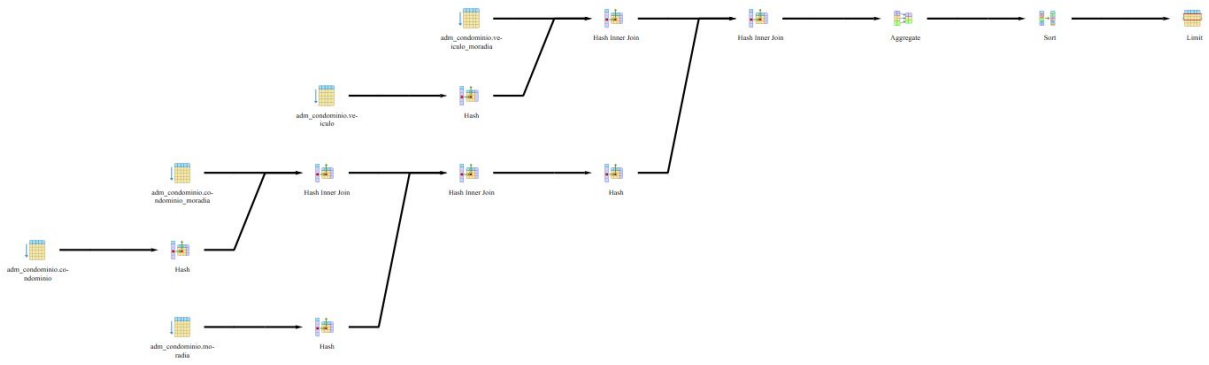
Consulta reformulada 1:

```
SELECT v.marca, COUNT(v.*) FROM adm_condominio.Veiculo AS v,
        adm_condominio.Veiculo_Moradia AS vM,
        adm_condominio.Moradia AS m,
        adm_condominio.Condominio_Moradia AS cM,
        adm_condominio.Condominio AS c
        WHERE vM.fk_id_veiculo = v.id_veiculo
        AND m.id_moradia = vM.fk_id_moradia
        AND m.tipo_moradia = 'a'
        AND cM.fk_id_moradia = m.id_moradia
        AND c.id_condominio = cM.fk_id_condominio
        AND c.tipo_condominio = 'e'
        GROUP BY v.marca
        ORDER BY 2
        LIMIT 3;
```

Plano de execução da consulta reformulada 1:

```
Limit  (cost=328.45..328.46 rows=3 width=16) (actual time=0.567..0.572 rows=3 loops=1)
  ->  Sort  (cost=328.45..328.55 rows=41 width=16) (actual time=0.566..0.567 rows=3 loops=1)
        Sort Key: (count(v.*))
        Sort Method: quicksort  Memory: 25kB
        ->  HashAggregate  (cost=327.51..327.92 rows=41 width=16) (actual time=0.550..0.553 rows=4 loops=1)
              Group Key: v.marca
              ->  Hash Join  (cost=59.65..229.16 rows=19671 width=76) (actual time=0.511..0.539 rows=4 loops=1)
                    Hash Cond: (vm.fk_id_moradia = m.id_moradia)
                    ->  Hash Join  (cost=2.80..41.45 rows=2260 width=80) (actual time=0.178..0.201 rows=20 loops=1)
                          Hash Cond: (vm.fk_id_veiculo = v.id_veiculo)
                          ->  Seq Scan on veiculo_moradia vm  (cost=0.00..32.60 rows=2260 width=8) (actual time=0.022..0.026 rows=20 loops=1)
                          ->  Hash  (cost=1.80..1.80 rows=80 width=80) (actual time=0.132..0.132 rows=80 loops=1)
                                Buckets: 1024  Batches: 1  Memory Usage: 17kB
                                ->  Seq Scan on veiculo v  (cost=0.00..1.80 rows=80 width=80) (actual time=0.020..0.087 rows=80 loops=1)
                    ->  Hash  (cost=47.06..47.06 rows=783 width=8) (actual time=0.310..0.310 rows=9 loops=1)
                          Buckets: 1024  Batches: 1  Memory Usage: 9kB
                          ->  Hash Join  (cost=5.20..47.06 rows=783 width=8) (actual time=0.260..0.301 rows=9 loops=1)
                                Hash Cond: (cm.fk_id_moradia = m.id_moradia)
                                ->  Hash Join  (cost=2.34..40.99 rows=1195 width=4) (actual time=0.127..0.160 rows=19 loops=1)
                                      Hash Cond: (cm.fk_id_condominio = c.id_condominio)
                                      ->  Seq Scan on condominio_moradia cm  (cost=0.00..32.60 rows=2260 width=8) (actual time=0.026..0.037 rows=40 loops=1)
                                      ->  Hash  (cost=1.88..1.88 rows=37 width=4) (actual time=0.081..0.081 rows=37 loops=1)
                                            Buckets: 1024  Batches: 1  Memory Usage: 10kB
                                            ->  Seq Scan on condominio c  (cost=0.00..1.88 rows=37 width=4) (actual time=0.028..0.060 rows=37 loops=1)
                                                  Filter: (tipo_condominio = 'e'::bpchar)
                                                  Rows Removed by Filter: 33
                                ->  Hash  (cost=2.13..2.13 rows=59 width=4) (actual time=0.114..0.114 rows=59 loops=1)
                                      Buckets: 1024  Batches: 1  Memory Usage: 11kB
                                      ->  Seq Scan on moradia m  (cost=0.00..2.13 rows=59 width=4) (actual time=0.027..0.075 rows=59 loops=1)
                                            Filter: (tipo_moradia = 'a'::bpchar)
                                            Rows Removed by Filter: 31
Planning time: 0.930 ms
Execution time: 0.745 ms
```

Imagem do plano de execução da consulta reformulada 1:



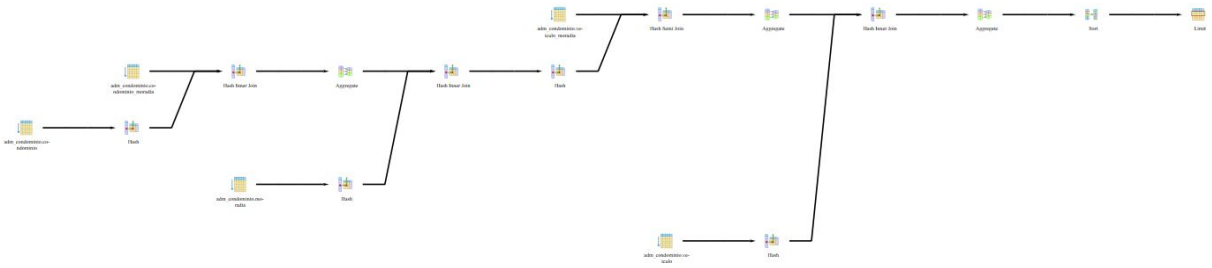
Consulta reformulada 2:

```
SELECT tab.marca, COUNT(tab.*)
FROM
  (SELECT * FROM adm_condominio.Veiculo AS veic
  WHERE veic.id_veiculo IN
  (SELECT vM.fk_id_veiculo FROM adm_condominio.Veiculo_Moradia AS vM
  WHERE vM.fk_id_moradia IN
  (SELECT mor.id_moradia FROM adm_condominio.Moradia AS mor
  WHERE mor.tipo_moradia = 'a' AND mor.id_moradia IN
  (SELECT cond_mor.fk_id_moradia FROM adm_condominio.Condominio_Moradia AS cond_mor
  WHERE cond_mor.fk_id_condominio IN
  (SELECT cond.id_condominio FROM adm_condominio.Condominio AS cond
  WHERE cond.tipo_condominio = 'e'))))) AS tab
  GROUP BY tab.marca
  ORDER BY 2
  LIMIT 3;
```

Plano de execução da consulta reformulada 2:

```
Limit  (cost=112.34..112.35 rows=3 width=16) (actual time=0.657..0.663 rows=3 loops=1)
  ->  Sort  (cost=112.34..112.44 rows=40 width=16) (actual time=0.656..0.656 rows=3 loops=1)
        Sort Key: (count(ROW(veic.id_veiculo, veic.placa, veic.cidade, veic.estado, veic.marca, veic.modelo, veic.cor)))
        Sort Method: quicksort  Memory: 25kB
        ->  HashAggregate  (cost=111.42..111.82 rows=40 width=16) (actual time=0.636..0.638 rows=4 loops=1)
              Group Key: veic.marca
              ->  Hash Join  (cost=107.58..111.22 rows=40 width=44) (actual time=0.615..0.621 rows=4 loops=1)
                    Hash Cond: (vm.fk_id_veiculo = veic.id_veiculo)
                    ->  HashAggregate  (cost=104.78..106.78 rows=200 width=4) (actual time=0.434..0.437 rows=4 loops=1)
                          Group Key: vm.fk_id_veiculo
                          ->  Hash Semi Join  (cost=50.85..101.96 rows=1130 width=4) (actual time=0.416..0.425 rows=4 loops=1)
                                Hash Cond: (vm.fk_id_moradia = mor.id_moradia)
                                ->  Seq Scan on veiculo_moradia vm  (cost=0.00..32.60 rows=2260 width=8) (actual time=0.023..0.027 rows=20 loops=1)
                                ->  Hash  (cost=50.48..50.48 rows=30 width=8) (actual time=0.356..0.356 rows=9 loops=1)
                                      Buckets: 1024  Batches: 1  Memory Usage: 9kB
                                      ->  Hash Join  (cost=46.84..50.48 rows=30 width=8) (actual time=0.321..0.337 rows=9 loops=1)
                                            Hash Cond: (cond_mor.fk_id_moradia = mor.id_moradia)
                                            ->  HashAggregate  (cost=43.98..45.98 rows=200 width=4) (actual time=0.214..0.222 rows=19 loops=1)
                                                  Group Key: cond_mor.fk_id_moradia
                                                  ->  Hash Join  (cost=2.34..40.99 rows=1195 width=4) (actual time=0.177..0.199 rows=19 loops=1)
                                                        Hash Cond: (cond_mor.fk_id_condominio = cond.id_condominio)
                                                        ->  Seq Scan on condominio_moradia cond_mor  (cost=0.00..32.60 rows=2260 width=8) (actual time=0.022..0.028 rows=40 loops=1)
                                                        ->  Hash  (cost=1.88..1.88 rows=37 width=4) (actual time=0.131..0.131 rows=37 loops=1)
                                                              Buckets: 1024  Batches: 1  Memory Usage: 10kB
                                                              ->  Seq Scan on condominio cond  (cost=0.00..1.88 rows=37 width=4) (actual time=0.088..0.112 rows=37 loops=1)
                                                                    Filter: (tipo_condominio = 'e'::bpchar)
                                                                    Rows Removed by Filter: 33
                                            ->  Hash  (cost=2.13..2.13 rows=59 width=4) (actual time=0.086..0.086 rows=59 loops=1)
                                                  Buckets: 1024  Batches: 1  Memory Usage: 11kB
                                                  ->  Seq Scan on moradia mor  (cost=0.00..2.13 rows=59 width=4) (actual time=0.027..0.057 rows=59 loops=1)
                                                        Filter: (tipo_moradia = 'a'::bpchar)
                                                        Rows Removed by Filter: 31
                    ->  Hash  (cost=1.80..1.80 rows=80 width=44) (actual time=0.156..0.156 rows=80 loops=1)
                          Buckets: 1024  Batches: 1  Memory Usage: 15kB
                          ->  Seq Scan on veiculo veic  (cost=0.00..1.80 rows=80 width=44) (actual time=0.048..0.081 rows=80 loops=1)
Planning time: 1.176 ms
Execution time: 1.689 ms
```

Imagem do plano de execução da consulta reformulada 2:



Avaliação sobre as consultas e seus planos:

Faremos em seguida, comparações em relação às consultas, os planos, tempos e afins, com o objetivo de entender as diferenças entre elas. A consulta original e as reformuladas trazem o mesmo resultado como exigido pelo artefato, as reformuladas foram escritas com outra sintaxe para observar se existe alguma alteração no plano de execução, a original se utiliza da sintaxe de *INNER JOIN* para fazer o produto cartesiano entre as tabelas e encontrar o identificador correspondente em ambas as tabelas para gerar a tabela somente com as tuplas que satisfazem as condições da consulta com a sintaxe de *ON* e *WHERE*.

A primeira consulta reformulada faz a busca utilizando a junção natural colocando todas as tabelas após o comando *FROM* com separação por vírgulas, indicando que será feito um produto cartesiano entre elas. A seguir, encontra-se a cláusula *WHERE* em que são colocadas todas as condições para encontrar as tuplas que se encaixam nas respectivas tabelas. Os comandos de *ON* foram substituídos por condições de igualdade entre colunas, unidos pelo operador lógico *AND*.

A segunda consulta reformulada faz a busca do resultado utilizando *SUBSELECTs* para substituir o que seriam os *INNER JOINs* da consulta original, ela começa do *SUBSELECT* mais interior e traz os resultados "recursivamente", fazendo uma consulta dentro da outra, utilizando o resultado da anterior para filtrar os dados e achar as tuplas compatíveis com as condições apresentadas. O comando *WHERE* das subconsultas utiliza a condição que se quer satisfazer com um comando de busca *IN* que varre o uma lista (ou tabela) de valores, em busca de achar correspondências, no caso, passamos um *SUBSELECT* que retorna um resultado e assim a consulta acima pode vasculhar em busca das informações necessárias a ela para satisfazer as condições que a passaram.

Comparemos agora os planos de execução de cada uma das consultas apresentadas neste artefato, nos quesitos de métodos, tempo e memória utilizados. Observando visualmente as consultas pelas imagens, nota-se uma equivalência de planos entre a original e a reformulada 1, e uma divergência de plano de execução para a reformulada 2 das outras duas consultas.

Devido ao fato do plano de execução da consulta original e da reformulada 1 serem o mesmo, falaremos somente uma vez sobre ele para depois compararmos com o plano de execução gerado pela consulta reformulada 2. No plano de execução destas consultas (original e reformulada 1), são buscadas as tabelas de Condomínio_Moradia e Condomínio primeiro e colocadas em buckets Hash, então é feito um Hash Inner Join para localizar as tuplas que correspondem às duas tabelas. Em seguida, o SGBD busca a tabela de Moradia e a coloca em buckets Hash, no próximo passo ele faz novamente um Hash Inner Join para localizar as tuplas correspondentes entre as duas tabelas de acordo com a coluna exigida, porém desta vez é feita com o resultado do Hash Inner Join entre Condomínio_Moradia e Condomínio, o resultado passa novamente por um Hash a fim de guardar as tuplas encontradas. Inicia-se a outra parte da consulta, na qual a tabela de Veículo passa pela Hash, o SGBD busca Veiculo_Moradia para fazer o Hash Inner Join entre elas, as tuplas resultantes passam por um Hash Inner Join com as tuplas entre Moradia e Condomínio_Moradia com Condomínio, feita na parte anterior para chegar nas tuplas que serão ordenadas e agrupadas. Em seguida, elas serão agrupadas pelo atributo de 'marca' da tabela Veículos, contabilizadas para serem ordenadas por 'marca' e limitadas por 3 marcas de veículos somente, como dado na consulta.

Com o plano das consultas original e reformulada 1 explicado, vamos para o plano de execução da consulta reformulada 2. O plano de execução da reformulada 2 começa idêntico ao dos planos anteriores, com

as tabelas Condomínio e Condomínio_Moradia sendo colocados em buckets de Hash para ser feito um Hash Inner Join entre elas e descobrir as tuplas correspondentes entre as tabelas, porém é feita uma agregação neste caso antes do próximo Hash Inner Join entre estas tuplas e a tabela de Moradia. Dando continuidade, será executado um Hash Inner Join entre as tuplas da consulta até o momento e a tabela de Veículo_Moradia, reduzindo a quantidade de tuplas e fazendo uma agregação novamente, e finalmente a última e mais importante tabela é buscada para ser feito o último Hash Inner Join entre as tuplas e ser feita a agregação que contará as linhas de marcas e então ordenará pelas marcas de Veículo e limitada por 3 ao final da consulta.

É possível observar que a diferença entre os planos se dá pela exigência de seguir a ordem de execução da consulta, na original e reformulada 1, o SGBD consegue tentar otimizar o máximo possível a consulta fazendo Inner Joins anteriormente e somente ao final agregar, ordenar e limitar. Com o plano de execução da reformulada 2, nota-se que ele segue estritamente os comandos passados pela consulta, fazendo as buscas e em uma tentativa de otimizar a mesma, agregar anteriormente para poder reduzir o escopo das tuplas geradas, já que deve seguir as regras passadas pela consulta, tendo 2 agregações a mais do que no outro plano de execução.

Em relação ao tempo gasto pelos planos, o primeiro feito pela consulta original, é o mais rápido de todos, com tempo de planejamento de 0.590 ms e de execução de 0.431 ms. A consulta reformulada 1 segue em segundo lugar com tempo de planejamento de 0.930 ms e execução de 0.745 ms, mais lento do que a original apesar de ter o mesmo plano de execução, isso pode ser resultado da sintaxe apresentada pela consulta que ocasionou uma demora significativa no planejamento, quase 0.4 ms a mais do que a original. E por fim, a reformulada 2, ela demorou em seu tempo de planejamento, 1.176 ms para gerar o plano de execução e 1.689 ms para executar este plano gerado, consideravelmente mais demorado do que as outras duas, mas também resultado da necessidade de seguir estritamente os comandos passados pela consulta, observa-se que a última consulta foi a mais demorada para ser executada em comparação com as outras.

Consultas	Tempo de planejamento	Tempo de execução
Original	0.590 ms	0.431 ms
Reformulada 1	0.930 ms	0.745 ms
Reformulada 2	1.176 ms	1.689 ms

A conclusão é de que a primeira consulta, a original, é a mais eficiente de todas, em termos de planejamento e execução, comparada às outras.