

Universidade Federal de São Carlos - Campus Sorocaba

Disciplina de Laboratório de Banco de Dados

Prof^a Sahudy Montenegro González

Alunos:	RA:
Leonardo Del Lama Furlan	587389
Luan Gustavo Maia Dias	587737
Renato Araujo Rizzo	587788

Sorocaba, 16 de Junho de 2018

1 - Introdução	2
2 - Minimundo	2
2.1 - Modelo Relacional	2
3 - Populando o Banco de Dados	2
4 - Especificação das Consultas e Otimizações	3
4.1 - Primeira Consulta	3
4.1.1 - Otimização da Primeira Consulta	4
4.2 - Segunda Consulta	6
4.2.1 - Otimização da Segunda Consulta	7
5 - Desempenho e testes das consultas	7
6 - Programação com Banco de Dados	12
7 - Controle de Acessos de Usuários	13
8 - Outras Informações sobre o Projeto	14
9 - Considerações Finais	14

1 - Introdução

Este relatório tem por objetivo especificar e demonstrar as otimizações e desempenho das consultas ao banco de dados de uma transportadora. Inclui uma descrição do minimundo do banco e é abordado como este banco foi populado com dados, as especificações das consultas em si, as otimizações realizadas nelas e por fim, os testes demonstrando o desempenho melhorado das consultas.

2 - Minimundo

A transportadora de cargas realiza viagens identificadas por um código único e a data de saída. Cada viagem é realizada por um motorista que possui: CPF (único), nome e CNH. Cada motorista dirige um único veículo por viagem, que possui: placa (único), renavam, chassi, modelo, cor, ano e capacidade.

Em cada viagem são realizadas diversas entregas de mercadorias, e todas elas possuem um destinatário. Cada mercadoria é identificada por: número da Nota-Fiscal (único), quantidade de volumes para um mesmo destinatário(quantidade de caixas, pacotes, fardos, etc.) que sempre será 1 ou mais e seu peso sempre maior que zero.

Toda mercadoria possui somente um destinatário e este pode possuir uma ou mais mercadorias. Todo destinatário só existe se atrelado a pelo menos uma mercadorias e possui: nome, CPF ou CNPJ e vários contatos telefônicos.

2.1 - Modelo Relacional

Motorista (CPF, CNH, nome_motorista)

Veiculo (placa, ano, capacidade, modelo, renavam, cor, chassi)

Viagem (codigo_viagem, data_viagem)

Motorista_Viagem (CPF, placa, codigo_viagem)
placa referencia Veiculo (placa)
CPF referencia Motorista (CPF)
cod_viagem referencia Viagem(codigo_viagem)

Entrega (codigo_viagem, NF)
cod_viagem referencia Viagem (codigo_viagem)
NF referencia Mercadoria(NF)

Mercadoria (NF, peso, qtd_volumes, data_compra, cpf_cnpj)
CPF_CNPJ referencia Destinatario (CPF_CNPJ)

Destinatario (CPF_CNPJ, nome_destinatario, endereco)

Contato_Destinatarior (CPF_CNPJ, contato)
CPF_CNPJ referencia Destinatario (CPF_CNPJ)

3 - Populando o Banco de Dados

Para popular o banco de dados, utilizamos o *site Database Test Data Generator* (databasetestdata.com) por conta do alto volume de dados que devia ser produzido para o trabalho. Trata-se de uma ferramenta que gera dados sintéticos personalizados de acordo com a escolha do usuário.

No entanto a maior parte dos dados foi gerada com *scripts* personalizados em C para geração de tuplas. Para a criação de dados numéricos os *scripts* em C tinham uma grande vantagem em tempo de geração de dados. Utilizamos o *site* apenas para geração de nomes e endereços, dados que seriam muito complicados para serem gerados por um *script* em C.

A quantidade de registros gerados e o tamanho do arquivo desses registros estão descritos na Tabela 1:

Tabela	Quantidade de registros	Tamanho do Arquivo
Contato_Destinario	1 milhão	94.237 KB
Destinatarios	1 milhão	128.940 KB
Entregas	1 milhão	70.313 KB
Mercadorias	1 milhão	126.572 KB
Motorista_Viagem	1 milhão	99.610 KB
Motorista	1,2 milhão	109.36 KB
Veiculos	1 milhão	163.963 KB
Viagens	1 milhão	81.055 KB

Tabela 1

4 - Especificação das Consultas e Otimizações

Foram especificadas duas consultas distintas para o banco de dados da transportadora. Ambas satisfazem os seguintes critérios exigidos: cinco campos de visualização do resultado; envolvem mais de duas tabelas em condições de seleção; são parametrizadas (possuem condições com parâmetros); A consulta 1 possui busca absoluta e a consulta 2 possui busca relativa; ambas utilizam operadores de condições e a primeira consulta possui ordenação do resultado. Para tentar otimizar, utilizamos índices, que serão abordados com mais detalhes em suas respectivas seções.

4.1 - Primeira Consulta

Recuperar todos os veículos que levaram todas as mercadorias cujos peso e quantidade de volumes se encaixam dentro dos limites escolhidos.

Campos de visualização do resultado: peso, qtd_volumes, placa, modelo, capacidade.

Campos de busca (ou das condições): peso (absoluta), qtd_volumes (absoluta).

Operadores das condições: peso (>, <) e qtd_volumes (>, <).

Para restringir a busca, utilizamos valores 0 a 35 para o campo parametrizado de peso e os valores 0 a 5 para o campo parametrizado de quantidade de volumes (qtd_volumes).

```
1. SELECT mercadoria.peso AS peso,
2.         mercadoria.qtd_volumes AS qtd_volumes,
3.         mercadoria_entrega.placa AS placa,
4.         mercadoria_entrega.modelo AS modelo,
5.         mercadoria_entrega.capacidade AS capacidade
6. FROM mercadoria
7.     INNER JOIN (
8.         SELECT entrega.nf AS nf,
9.                veiculo_viagem.placa AS placa,
10.               veiculo_viagem.modelo AS modelo,
11.               veiculo_viagem.capacidade AS capacidade
12. FROM entrega
13.     INNER JOIN (
14.         SELECT veiculo.placa AS placa,
15.                veiculo.modelo AS modelo,
16.                veiculo.capacidade AS capacidade,
17.                motorista_viagem.codigo_viagem AS codigo_viagem
18. FROM veiculo
19.     INNER JOIN motorista_viagem
20.         ON veiculo.placa = motorista_viagem.placa
21.     ) AS veiculo_viagem
22.     ON entrega.codigo_viagem = veiculo_viagem.codigo_viagem
23. ) AS mercadoria_entrega
24.     ON mercadoria.nf = mercadoria_entrega.nf
25. WHERE peso > <peso_minimo> AND peso < <peso_maximo> AND qtd_volumes >
        <qtd_minima> AND qtd_volumes < <qtd_maxima>
26. ORDER BY peso;
```

Script 1

4.1.1 - Otimização da Primeira Consulta

Para otimizar a primeira consulta analisamos as tabelas envolvidas, os atributos usados nas condições de seleção e os atributos usados nas condições de junção.

Criamos índices para cada tabela envolvida na consulta e utilizamos a ferramenta do SGBD para analisar o uso dos índices. Como exibido na imagem abaixo, o índice "idx_mercadoria_peso_qtd_volumes" está sendo utilizado e está ajudando na otimização de seleção para a tabela de mercadoria que possui esses 3 atributos como chave, o que pode ajudar na seleção, já que as colunas dos índices são utilizadas para comparação. Isso pode ser visualizado abaixo:

```
1. "Gather Merge (cost=276137.23..495978.25 rows=1884220 width=27) (actual
   time=27399.073..29025.087 rows=2253573 loops=1)"
2. "  Workers Planned: 2"
3. "    Workers Launched: 2"
4. "      -> Sort (cost=275137.21..277492.49 rows=942110 width=27) (actual
   time=25879.755..26121.796 rows=751191 loops=3)"
5. "        Sort Key: mercadoria.peso"
6. "        Sort Method: external merge  Disk: 47304kB"
7. "          -> Hash Join (cost=115128.77..159110.32 rows=942110 width=27)
   (actual time=22671.106..24988.634 rows=751191 loops=3)"
8. "            Hash Cond: (motorista_viagem.codigo_viagem =
   entrega.codigo_viagem)"
9. "              -> Hash Join (cost=40069.29..65750.05 rows=416078
   width=23) (actual time=6389.321..8067.897 rows=332862 loops=3)"
10. "                Hash Cond: ((motorista_viagem.placa)::text =
   (veiculo.placa)::text)"
11. "                  -> Parallel Seq Scan on motorista_viagem
   (cost=0.00..10530.78 rows=416078 width=12) (actual time=0.156..139.519
   rows=332862 loops=3)"
12. "                    -> Hash (cost=21793.35..21793.35 rows=995435
   width=19) (actual time=6380.070..6380.070 rows=995435 loops=3)"
13. "                      Buckets: 65536  Batches: 16  Memory Usage:
   3835kB"
14. "                        -> Seq Scan on veiculo (cost=0.00..21793.35
   rows=995435 width=19) (actual time=4.185..5598.533 rows=995435 loops=3)"
15. "                          -> Hash (cost=73769.02..73769.02 rows=74197 width=12)
   (actual time=16187.384..16187.384 rows=73915 loops=3)"
16. "                            Buckets: 131072  Batches: 2  Memory Usage: 2609kB"
17. "                              -> Hash Join (cost=35251.29..73769.02 rows=74197
   width=12) (actual time=12443.898..16137.482 rows=73915 loops=3)"
18. "                                Hash Cond: (entrega.nf = mercadoria.nf)"
19. "                                  -> Seq Scan on entrega (cost=0.00..15404.56
   rows=999856 width=12) (actual time=0.144..2551.646 rows=999856 loops=3)"
20. "                                    -> Hash (cost=32674.20..32674.20 rows=148247
   width=16) (actual time=12417.552..12417.552 rows=148751 loops=3)"
21. "                                      Buckets: 131072  Batches: 4  Memory
   Usage: 2814kB"
22. "                                          -> Bitmap Heap Scan on mercadoria
   (cost=9627.26..32674.20 rows=148247 width=16) (actual
   time=199.062..12280.807 rows=148751 loops=3)"
23. "                                            Recheck Cond: ((peso > '0'::double
   precision) AND (peso < '35'::double precision) AND (qtd_volumes > 0) AND
   (qtd_volumes < 5))"
24. "                                              Heap Blocks: exact=16667"
25. "                                                -> Bitmap Index Scan on
   idx_mercadoria_peso_qtd_volumes (cost=0.00..9590.20 rows=148247 width=0)
   (actual time=194.457..194.457 rows=148751 loops=3)"
26. "                                                  Index Cond: ((peso >
   '0'::double precision) AND (peso < '35'::double precision) AND
```

```
(qtd_volumes > 0) AND (qtd_volumes < 5))"
```

Plano de execução 1

Usamos o algoritmo básico de regras heurísticas para quebrar as seleções que tinham condições conjuntivas, reordenamos os nós externos para as seleções mais restritivas sejam executadas primeiro e combinamos operações de produto cartesiano com as subsequentes seleções para formar junções equivalentes.

Após a remoção de aninhamentos chegamos a consulta mostrada no *Script 2*:

```
1. SELECT mercadoria.peso AS peso,
2.     mercadoria.qtd_volumes AS qtd_volumes,
3.     veiculo.placa AS placa,
4.     veiculo.modelo AS modelo,
5.     veiculo.capacidade AS capacidade
6. FROM mercadoria
7.     INNER JOIN entrega
8.     ON mercadoria.nf = entrega.nf
9.     INNER JOIN motorista_viagem
10.    ON entrega.codigo_viagem = motorista_viagem.codigo_viagem
11.    INNER JOIN veiculo
12.    ON veiculo.placa = motorista_viagem.placa
13. WHERE peso >= <peso_minimo> AND peso <= <peso_maximo> AND qtd_volumes >=
    <qtd_minima> AND qtd_volumes <= <qtd_maxima>
14. ORDER BY peso;
```

Script 2

4.2 - Segunda Consulta

Recuperar todas as viagens que foram feitas a cada destinatário.

Campos de visualização do resultado: cpf_cnpj_destinatario, nome_destinatario, cpf_motorista, cnh_motorista, nome_motorista.

Campos de busca (ou das condições): nome_destinatario (relativa).

Operadores das condições: nome_destinatario (ILIKE).

```
1. SELECT destinatario.cpf_cnpj AS cpf_cnpj_destinatario,
2.     destinatario.nome_destinatario AS nome_destinatario,
3.     mercadoria_motorista.cpf AS cpf_motorista,
4.     mercadoria_motorista.cnh AS cnh_motorista,
5.     mercadoria_motorista.nome_motorista AS nome_motorista
6. FROM destinatario
7.     INNER JOIN (
8.         SELECT mercadoria.cpf_cnpj AS destinatario_cpf_cnpj,
9.             mercadoria_entrega.cpf AS cpf,
10.            mercadoria_entrega.cnh AS cnh,
```

```

11.      mercadoria_entrega.nome_motorista AS nome_motorista
12. FROM mercadoria
13. INNER JOIN (
14.     SELECT entrega.nf AS nf,
15.            moto_viagem.cpf AS cpf,
16.            moto_viagem.cnh AS cnh,
17.            moto_viagem.nome_motorista AS nome_motorista
18.     FROM entrega
19.     INNER JOIN (
20.          SELECT motorista.cpf AS cpf,
21.                 motorista.cnh AS cnh,
22.                 motorista.nome AS nome_motorista,
23.                 motorista_viagem.codigo_viagem AS codigo_viagem
24.          FROM motorista
25.          INNER JOIN motorista_viagem
26.          ON motorista.cpf = motorista_viagem.cpf
27.        ) AS moto_viagem
28.     ON entrega.codigo_viagem = moto_viagem.codigo_viagem
29.   ) AS mercadoria_entrega
30.  ON mercadoria.nf = mercadoria_entrega.nf
31.   GROUP BY      mercadoria.cpf_cnpj,      mercadoria_entrega.cpf,
32.                mercadoria_entrega.cnh, mercadoria_entrega.nome_motorista
33.  ON destinatario.cpf_cnpj = mercadoria_motorista.destinatario_cpf_cnpj
34. WHERE nome_destinatario ILIKE <nome_destinatario>;

```

Script 3

4.2.1 - Otimização da Segunda Consulta

Para otimizar a primeira consulta analisamos as tabelas envolvidas, os atributos usados nas condições de seleção os atributos de usados nas condições de junção. Criamos índices para cada tabela envolvida na consulta e utilizamos a ferramenta do SGBD para analisar o uso dos índices. Para essa consulta observamos que antes de aplicar outra solução de otimização, o SGBD estava utilizando o índice "idx_destinatario_cpf_cnpj" da tabela destinatário como mostrado abaixo (*Plano de execução 2* não otimizado e *Plano de Execução 1* otimizado):

```

1. "Merge Join (cost=2587026.29..6210618.74 rows=1829878 width=57) (actual
   time=256131.753..401283.238 rows=1766774 loops=1)"
2. "  Merge Cond: (mercadoria.cpf_cnpj = destinatario.cpf_cnpj)"
3. "    -> Group (cost=2587025.87..5739969.89 rows=30469407 width=40) (actual
       time=227900.278..298659.676 rows=29614460 loops=1)"
4. "      Group Key: mercadoria.cpf_cnpj, motorista.cpf"
5. "      -> Gather Merge (cost=2587025.87..5613014.03 rows=25391172
       width=40) (actual time=227900.276..279749.549 rows=30176360 loops=1)"
6. "        Workers Planned: 2"
7. "        Workers Launched: 2"
8. "      -> Group (cost=2586025.84..2681242.74 rows=12695586
       width=40) (actual time=223359.309..244699.758 rows=10058787 loops=3)"

```



```

9. "                Group Key: mercadoria.cpf_cnpj, motorista.cpf"
10. "                -> Sort (cost=2586025.84..2617764.81 rows=12695586
    width=40) (actual time=223359.304..238806.196 rows=10159106 loops=3)"
11. "                Sort Key: mercadoria.cpf_cnpj, motorista.cpf"
12. "                Sort Method: external merge  Disk: 531264kB"
13. "                -> Hash Join (cost=188389.88..393789.85
    rows=12695586 width=40) (actual time=35292.859..49769.121 rows=10159106
    loops=3)"
14. "                Hash Cond:
    (motorista_viagem.codigo_viagem = entrega.codigo_viagem)"
15. "                -> Hash Join (cost=44771.75..72823.52
    rows=416078 width=35) (actual time=2123.725..10433.597 rows=332862
    loops=3)"
16. "                Hash Cond: (motorista_viagem.cpf =
    motorista.cpf)"
17. "                -> Parallel Seq Scan on
    motorista_viagem (cost=0.00..10530.78 rows=416078 width=12) (actual
    time=0.037..6078.139 rows=332862 loops=3)"
18. "                -> Hash (cost=21567.89..21567.89
    rows=1199989 width=31) (actual time=2079.044..2079.044 rows=1199989
    loops=3)"
19. "                Buckets: 65536  Batches: 32
    Memory Usage: 2914kB"
20. "                -> Seq Scan on motorista
    (cost=0.00..21567.89 rows=1199989 width=31) (actual time=0.118..672.351
    rows=1199989 loops=3)"
21. "                -> Hash (cost=126236.93..126236.93
    rows=999856 width=13) (actual time=33005.543..33005.543 rows=999856
    loops=3)"
22. "                Buckets: 131072  Batches: 16
    Memory Usage: 3945kB"
23. "                -> Hash Join
    (cost=76737.20..126236.93 rows=999856 width=13) (actual
    time=8354.758..32576.817 rows=999856 loops=3)"
24. "                Hash Cond: (entrega.nf =
    mercadoria.nf)"
25. "                -> Seq Scan on entrega
    (cost=0.00..15404.56 rows=999856 width=12) (actual time=0.230..19327.601
    rows=999856 loops=3)"
26. "                -> Hash
    (cost=40059.42..40059.42 rows=1997742 width=17) (actual
    time=8302.651..8302.651 rows=1997742 loops=3)"
27. "                Buckets: 65536
    Batches: 32  Memory Usage: 3640kB"
28. "                -> Seq Scan on
    mercadoria (cost=0.00..40059.42 rows=1997742 width=17) (actual
    time=0.112..1067.632 rows=1997742 loops=3)"
29. "                -> Index Scan using idx_destinatario_cpf_cnpj on destinatario
    (cost=0.42..71332.34 rows=60056 width=26) (actual time=2060.553..94962.081
    rows=58190 loops=1)"
30. "                Filter: ((nome_destinatario)::text ~~* 'e%':::text)"
31. "                Rows Removed by Filter: 941784"

```

Plano de execução 2

```

1. "Hash Join (cost=164175.78..252898.10 rows=1829890 width=57) (actual
    time=15888.328..19450.847 rows=1818363 loops=1)"
2. "  Hash Cond: (motorista_viagem.codigo_viagem = entrega.codigo_viagem)"
3. "  -> Hash Join (cost=44771.75..91690.77 rows=998587 width=35) (actual
    time=8239.928..10689.297 rows=998587 loops=1)"
4. "    Hash Cond: (motorista_viagem.cpf = motorista.cpf)"
5. "    -> Seq Scan on motorista_viagem (cost=0.00..16355.87
    rows=998587 width=12) (actual time=0.035..637.510 rows=998587 loops=1)"

```

```

6. "      -> Hash (cost=21567.89..21567.89 rows=1199989 width=31) (actual
   time=8237.071..8237.071 rows=1199989 loops=1)"
7. "          Buckets: 65536 Batches: 32 Memory Usage: 2914kB"
8. "      -> Seq Scan on motorista (cost=0.00..21567.89
   rows=1199989 width=31) (actual time=0.028..7525.652 rows=1199989 loops=1)"
9. "      -> Hash (cost=118242.43..118242.43 rows=60048 width=30) (actual
   time=7644.909..7644.909 rows=59613 loops=1)"
10. "          Buckets: 65536 Batches: 2 Memory Usage: 2399kB"
11. "      -> Hash Join (cost=87783.93..118242.43 rows=60048 width=30)
   (actual time=6154.882..7614.115 rows=59613 loops=1)"
12. "          Hash Cond: (entrega.nf = mercadoria.nf)"
13. "      -> Seq Scan on entrega (cost=0.00..15404.56 rows=999856
   width=12) (actual time=0.024..830.441 rows=999856 loops=1)"
14. "      -> Hash (cost=85346.22..85346.22 rows=119977 width=34)
   (actual time=6153.478..6153.478 rows=119456 loops=1)"
15. "          Buckets: 65536 Batches: 4 Memory Usage: 2495kB"
16. "      -> Gather (cost=24266.65..85346.22 rows=119977
   width=34) (actual time=3513.367..6054.673 rows=119456 loops=1)"
17. "          Workers Planned: 2"
18. "          Workers Launched: 2"
19. "      -> Hash Join (cost=23266.65..72348.52
   rows=49990 width=34) (actual time=3512.819..5922.565 rows=39819 loops=3)"
20. "          Hash Cond: (mercadoria.cpf_cnpj =
   destinatario.cpf_cnpj)"
21. "      -> Parallel Seq Scan on mercadoria
   (cost=0.00..28405.93 rows=832392 width=17) (actual time=19.698..1746.533
   rows=665914 loops=3)"
22. "      -> Hash (cost=22104.95..22104.95
   rows=60056 width=26) (actual time=3422.957..3422.957 rows=58191 loops=3)"
23. "          Buckets: 65536 Batches: 2 Memory
   Usage: 2180kB"
24. "      -> Seq Scan on destinatario
   (cost=0.00..22104.95 rows=60056 width=26) (actual time=14.458..3364.797
   rows=58191 loops=3)"
25. "          Filter:
   ((nome_destinatario)::text ~~* 'e%':::text)"
26. "          Rows Removed by Filter:
   941805"

```

Plano de execução 3

Após a remoção de aninhamentos chegamos a consulta mostrada no *Script 4*:

```

1. SELECT destinatario.nome_destinatario AS nome_destinatario,
2.        destinatario.cpf_cnpj AS cpf_cnpj_destinatario,
3.        motorista.cpf AS cpf_motorista,
4.        motorista.cnh AS cnh_motorista,
5.        motorista.nome AS nome_motorista
6. FROM destinatario
7.      INNER JOIN mercadoria
8.      ON destinatario.cpf_cnpj = mercadoria.cpf_cnpj
9.      INNER JOIN entrega
10.     ON entrega.nf = mercadoria.nf
11.      INNER JOIN motorista_viagem
12.     ON motorista_viagem.codigo_viagem = entrega.codigo_viagem
13.      INNER JOIN motorista
14.     ON motorista.cpf = motorista_viagem.cpf

```

```
15. WHERE nome_destinatario ILIKE <nome_destinatario>;
```

Script 4

5 - Desempenho e testes das consultas

Para analisar o desempenho das duas consultas, utilizamos dois computadores com configurações distintas descritas abaixo:

Computador 1:

Processador: Intel I3-3110M 2.40 GHz 3M cache

Memória: 6GB RAM

Armazenamento: HDD 500GB (mecânico - 80mb/s de leitura)

SO: Windows 10 64 bits

Consulta 1 - Computador 1			
	Consulta Inicial	Consulta Otimizada	Diferença (%)
Tempo 1	28.057ms	8.414ms	77,01
Tempo 2	26.013ms	7.297ms	71,95
Tempo 3	33.168ms	7.788ms	76,502
Tempo 4	29.103ms	8.353ms	71,30
Tempo 5	34.436ms	7.235ms	78,99
Média	30.155ms	7.817ms	73,75

Tabela 2

Na Tabela 2 é possível observar que na primeira consulta houve ganho de 73,75% de tempo em média.

Consulta 2 - Computador 1			
	Consulta Inicial	Consulta Otimizada	Diferença (%)
Tempo 1	401.730ms	11.709ms	97,08
Tempo 2	400.510ms	10.826ms	97,29
Tempo 3	400.724ms	10.714ms	97,32
Tempo 4	400.812ms	11.504ms	97,12
Tempo 5	401.215ms	11.105ms	97,23

Média	400.998ms	11.171ms	97,21
-------	-----------	----------	-------

Tabela 3

Na *Tabela 3* é possível observar que na segunda consulta houve ganho de 97,21% de tempo em média.

Computador 2:

Processador: Intel I5-3570 3.40 GHz 6M cache

Memória: 16GB RAM

Armazenamento: SSD 250GB (450 mb/s de leitura)

SO: Windows 10 64 bits

Consulta 1 - Computador 2			
	Consulta Inicial	Consulta Otimizada	Diferença (%)
Tempo 1	9.950ms	7.642ms	23,20
Tempo 2	8.658ms	7.581ms	12,44
Tempo 3	8.600ms	7.649ms	11,05
Tempo 4	8.550ms	7.564ms	11,53
Tempo 5	8.659ms	7.864ms	9,18
Média	8.883ms	7.660ms	15,97

Tabela 4

Na *Tabela 4* é possível observar que na primeira consulta houve ganho de 15,97% de tempo em média.

Consulta 2 - Computador 2			
	Consulta Inicial	Consulta Otimizada	Diferença (%)
Tempo 1	90.000ms	5.794ms	93,56
Tempo 2	103.000ms	5.804ms	94,36
Tempo 3	92.000ms	5.784ms	93,71
Tempo 4	89.000ms	5.779ms	93,51
Tempo 5	110.000ms	5.953ms	94,59
Média	96.800ms	5.822ms	93,95

Tabela 5

Na *Tabela 5* é possível observar que na segunda consulta houve ganho de 93,95% de tempo em média.

O uso de dois computadores, um com armazenamento mecânico e configurações mais simples e outro com SSD com configurações mais robustas, foi interessante para saber o quanto as otimizações afetam para diferentes tipos de computadores que podem ser utilizados para servir o banco de dados.

6 - Programação com Banco de Dados

Foram geradas duas *stored procedures*, uma para cada consulta. O que cada *stored procedure* é realizar a seleção referente àquela consulta.

A *Stored Procedure* para a primeira consulta, possui como entrada o peso mínimo e máximo, e a quantidade de volumes mínima e máxima que foi carregada pelo veículo, além do limite e o *offset* necessários para retornar uma parte do resultado total da seleção, como na paginação. O *script* se encontra abaixo:

```
1. CREATE OR REPLACE FUNCTION consultar_veiculos(_peso_minimo REAL,
   _peso_maximo REAL, _qtd_volumes_minimo INTEGER, _qtd_volumes_maximo
   INTEGER, _limit INTEGER, _offset INTEGER)
2. RETURNS TABLE(peso real, qtd_volumes integer, placa varchar(7), modelo
   varchar(20), capacidade real) AS $$
3. BEGIN
4.     RETURN QUERY
5.     SELECT mercadoria.peso AS peso,
6.            mercadoria.qtd_volumes AS qtd_volumes,
7.            veiculo.placa AS placa,
8.            veiculo.modelo AS modelo,
9.            veiculo.capacidade AS capacidade
10.    FROM mercadoria
11.     INNER JOIN entrega
12.     ON mercadoria.nf = entrega.nf
13.     INNER JOIN motorista_viagem
14.     ON entrega.codigo_viagem = motorista_viagem.codigo_viagem
15.     INNER JOIN veiculo
16.     ON veiculo.placa = motorista_viagem.placa
17.     WHERE mercadoria.peso >= _peso_minimo AND mercadoria.peso <=
   _peso_maximo AND mercadoria.qtd_volumes >= _qtd_volumes_minimo AND
   mercadoria.qtd_volumes <= _qtd_volumes_maximo
18.     ORDER BY peso
19.     LIMIT _limit
20.     OFFSET _offset;
21. END;
22. $$ LANGUAGE plpgsql;
```

Script 5

A *Stored Procedures* para a segunda consulta, possui como entrada o nome do destinatário (busca relativa), além do limite e o *offset* necessários para retornar uma parte do resultado total da seleção, como na paginação. O *script* se encontra abaixo:

```
1. CREATE OR REPLACE FUNCTION consultar_viagens(_nome_destinatario
   VARCHAR(50), _limit INTEGER, _offset INTEGER)
2. RETURNS TABLE(nome_destinatario varchar(50), cpf_cnpj_destinatario
   numeric(14), cpf_motorista numeric(11), cnh_motorista numeric(11),
   nome_motorista varchar(50)) AS $$
3. BEGIN
4.     RETURN QUERY
5.     SELECT destinatario.nome_destinatario AS nome_destinatario,
6.            destinatario.cpf_cnpj AS cpf_cnpj_destinatario,
7.            motorista.cpf AS cpf_motorista,
8.            motorista.cnh AS cnh_motorista,
9.            motorista.nome AS nome_motorista
10.    FROM destinatario
11.     INNER JOIN mercadoria
12.     ON destinatario.cpf_cnpj = mercadoria.cpf_cnpj
13.     INNER JOIN entrega
14.     ON entrega.nf = mercadoria.nf
15.     INNER JOIN motorista_viagem
16.     ON motorista_viagem.codigo_viagem = entrega.codigo_viagem
17.     INNER JOIN motorista
18.     ON motorista.cpf = motorista_viagem.cpf
19.   WHERE destinatario.nome_destinatario ILIKE _nome_destinatario
20.   LIMIT _limit
21.  OFFSET _offset;
22. END;
23. $$ LANGUAGE 'plpgsql';
```

Script 6

7 - Controle de Acessos de Usuários

Utilizamos de controle de privilégios de usuários para fornecer maior segurança ao nosso sistema, e escolhemos implementar um *Role* e monitorar as permissões do mesmo. Garantimos apenas o privilégio sobre *Select*, visto que não há necessidade do uso de *Deletes*, *Insert* ou quaisquer outros comandos.

```
1. CREATE ROLE app_role_select WITH LOGIN PASSWORD 'postgres';
2. GRANT SELECT ON ALL TABLES IN SCHEMA PUBLIC TO app_role_select;
```

Script 7

8 - Outras Informações sobre o Projeto

Para o projeto foi utilizado o banco de dados *PostgreSQL*, a linguagem de programação de *back-end* foi *Javascript* através do ambiente *Node.js* e de *front-end* foi utilizado o *Angular 5* com *TypeScript*.

A tela da aplicação para a primeira consulta é a seguinte:

Peso	Volumes	Placa	Modelo	Capacidade
20	2	EEQ8875	Grand Prix	41.2
20	2	GQL4445	Tacoma	35.34
20	2	MGP2966	Jetta	52.13
20	2	RVD2633	Neon	54.51
20	2	XKB9955	Justy	77.45
20	2	DEH5689	Sidekick	23.25
20	2	ITN1219	XF	52.54

Figura 2

A tela da aplicação para a segunda consulta é a seguinte:

Nome Destinatário	CPF/CNPJ Destinatário	CNH Motorista	Nome Motorista	CPF Motorista
Albartho Wahnnar MEI	14815708554277	85972665927	Floyd Howell MD	10320570583
Albartho Wahnnar MEI	14815708554277	19340790061	Jalen Gaylord	12446774143
Albartho Wahnnar MEI	14815708554277	23178722405	Mr. Macy Lubowitz	13337925526
Albartho Wahnnar MEI	14815708554277	80926528116	Keeley Wilderman	17146783512
Albartho Wahnnar MEI	14815708554277	35727281713	Louisa Wiegand MD	17339747030
Albartho Wahnnar MEI	14815708554277	58482325966	Mrs. Lisette Medhurst	18608656776
Albartho Wahnnar MEI	14815708554277	51669035845	Miss Ludie Bruen	19904130069

Figura 3

9 - Considerações finais

Após todo o trabalho de otimização do banco de dados foi possível obter boas melhoras, antes a segunda consulta com certos parâmetros chegava a demorar em média 7 minutos em um *notebook* com armazenamento mecânico, enquanto que com a otimização foi possível realizar a mesma consulta com cerca de 10 segundos. A primeira consulta também teve sua melhoria também, antes esta consulta com certos parâmetros demorava em média 30 segundos e foi possível diminuir para cerca de 7 segundos.