
Trabalho de Otimização de Consultas em Sistemas Gerenciadores de Banco de Dados Relacionais

Alunos: Erick Henrique, Iagor e Marina

O trio escolheu o PostgreSQL como Sistema de Gerenciamento de Banco de Dados (SGBD) devido à sua robustez em armazenamento e manipulação de dados. A base de dados selecionada inclui informações detalhadas sobre todos os veículos que passaram por alguns dos radares de Belo Horizonte¹ do dia 06 de agosto de 2023. Os dados registrados abrangem o tipo de veículo (carro, caminhão, moto), o id dos radares, a velocidade dos veículos, a velocidade permitida pelo radar, o endereço do radar e o sentido do radar. Esta base de dados contém mais de um 1,5 milhões de linhas.

Toda a estrutura usada para criar a base de dados original, a base de dados otimizada e os scripts auxiliares podem ser encontrados no repositório do GitHub².

Estruturas de Dados no PostgreSQL

Armazenamento de Dados

O PostgreSQL utiliza diversas estruturas para o armazenamento eficiente dos dados. A principal estrutura é a tabela, que organiza os dados em linhas e colunas, permitindo operações de CRUD (Create, Read, Update, Delete) de maneira otimizada. Além disso, o PostgreSQL pode usar clusters, que agrupam tabelas com base em um índice comum, otimizando consultas complexas que acessam múltiplas tabelas simultaneamente.

Indexação

Para melhorar o desempenho das consultas, o PostgreSQL oferece vários tipos de índices:

- **Índices Únicos e Não Únicos:** Enquanto os índices únicos garantem a unicidade dos dados em uma coluna específica, os índices não únicos são utilizados para melhorar a performance de buscas, permitindo valores duplicados.
- **Índices Organizados em Tabelas (Clustered):** Nestes índices, a ordem das linhas na tabela segue a ordem do índice, melhorando a eficiência de consultas sequenciais.
- **Índices Bitmap:** São especialmente úteis em colunas com poucos valores distintos, permitindo combinações eficientes de condições em consultas.

¹ Disponível em: <https://dados.pbh.gov.br/dataset/contagens-volumetricas-de-radares>

² <https://github.com/ErickHDdS/OtimizacaoBancoDados>

Paralelização e Particionamento

Para maximizar o uso de recursos de hardware e lidar com grandes volumes de dados, o PostgreSQL suporta:

- **Paralelização de Consultas:** Essa técnica divide a execução de uma consulta entre múltiplos núcleos de CPU, acelerando o processamento de grandes conjuntos de dados.
- **Particionamento de Tabelas:** Essa funcionalidade divide grandes tabelas em partções menores, cada uma podendo ser gerenciada separadamente, o que melhora a performance e facilita a manutenção.

Algoritmos de SELECT e JOIN

O PostgreSQL disponibiliza diversos algoritmos para executar operações de seleção e junção de dados:

Algoritmos de SELECT

- **Seq Scan:** Realiza uma leitura sequencial de toda a tabela, utilizado quando não há índices aplicáveis.
- **Index Scan:** Utiliza índices para acessar dados de forma mais rápida, reduzindo a necessidade de ler todas as linhas.
- **Bitmap Heap Scan:** Combina condições de múltiplos índices para filtrar registros de maneira eficiente.

Algoritmos de JOIN

- **Nested Loop Join:** Combina cada linha de uma tabela com todas as linhas de outra, eficiente para conjuntos de dados pequenos.
- **Hash Join:** Cria tabelas hash para combinar registros, ideal para grandes conjuntos de dados onde as tabelas não estão ordenadas.
- **Merge Join:** Realiza junções em tabelas já ordenadas, sendo eficiente para grandes volumes de dados.

Visualizador de plano de execução - Postgres Explain Visualizer 2 (PEV2)

O Postgres Explain Visualizer 2 (PEV2)³ é uma ferramenta open-source desenvolvida para ajudar os desenvolvedores e administradores de banco de dados a visualizarem e entenderem os planos de execução de consultas no PostgreSQL. Ele fornece uma interface

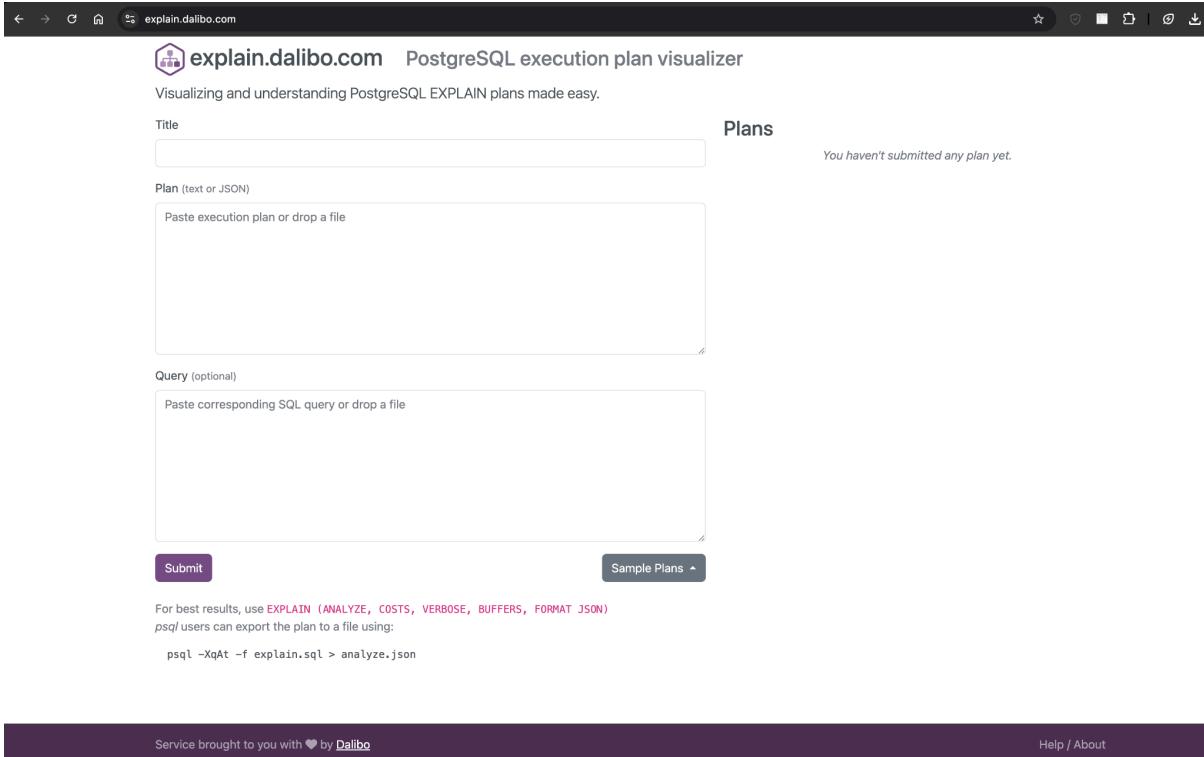
³ Disponível em: <https://github.com/dalibo/pev2>

gráfica intuitiva que facilita a análise de desempenho das consultas, permitindo a identificação rápida de gargalos e oportunidades de otimização. Suas funcionalidades principais são:

1. **Visualização Gráfica:** Transforma os planos de execução do PostgreSQL em gráficos interativos e de fácil compreensão.
2. **Análise Detalhada:** Permite explorar cada passo do plano de execução com informações detalhadas sobre custos, tempos de execução e uso de buffers.
3. **Comparação de Planos:** Possibilita a comparação de diferentes planos de execução para a mesma consulta ou entre consultas diferentes.

Para gerar o plano de execução de uma consulta no PEV2, os seguintes passos devem ser seguidos:

1. Executar a consulta precedida por `EXPLAIN (ANALYZE, COSTS, VERBOSE, BUFFERS, FORMAT JSON)`.
2. Copiar a saída JSON gerada pelo comando acima.
3. Cole a saída JSON dentro do campo Plan no PEV2.
4. Insira a consulta original no campo Query no PEV2.



The screenshot shows the homepage of the explain.dalibo.com website. At the top, there's a navigation bar with icons for back, forward, search, and other site functions. The URL 'explain.dalibo.com' is visible. Below the header, the title 'explain.dalibo.com PostgreSQL execution plan visualizer' is displayed, along with the subtitle 'Visualizing and understanding PostgreSQL EXPLAIN plans made easy.' On the left side, there are three input fields: 'Title' (empty), 'Plan (text or JSON)' containing a placeholder 'Paste execution plan or drop a file', and 'Query (optional)' containing a placeholder 'Paste corresponding SQL query or drop a file'. To the right of these fields is a section titled 'Plans' with the sub-instruction 'You haven't submitted any plan yet.' At the bottom of the page, there are two buttons: 'Submit' and 'Sample Plans ▾'. A note at the bottom left suggests using `EXPLAIN (ANALYZE, COSTS, VERBOSE, BUFFERS, FORMAT JSON)` and provides a command for psql users: `psql -XqAt -f explain.sql > analyze.json`. The footer contains the text 'Service brought to you with ❤ by Dalibo' and 'Help / About'.

Depois de completar esses passos, o PEV2 irá gerar uma visualização gráfica do plano de execução da sua consulta, permitindo uma análise detalhada e visual das operações executadas pelo PostgreSQL.

Otimizador PgTune

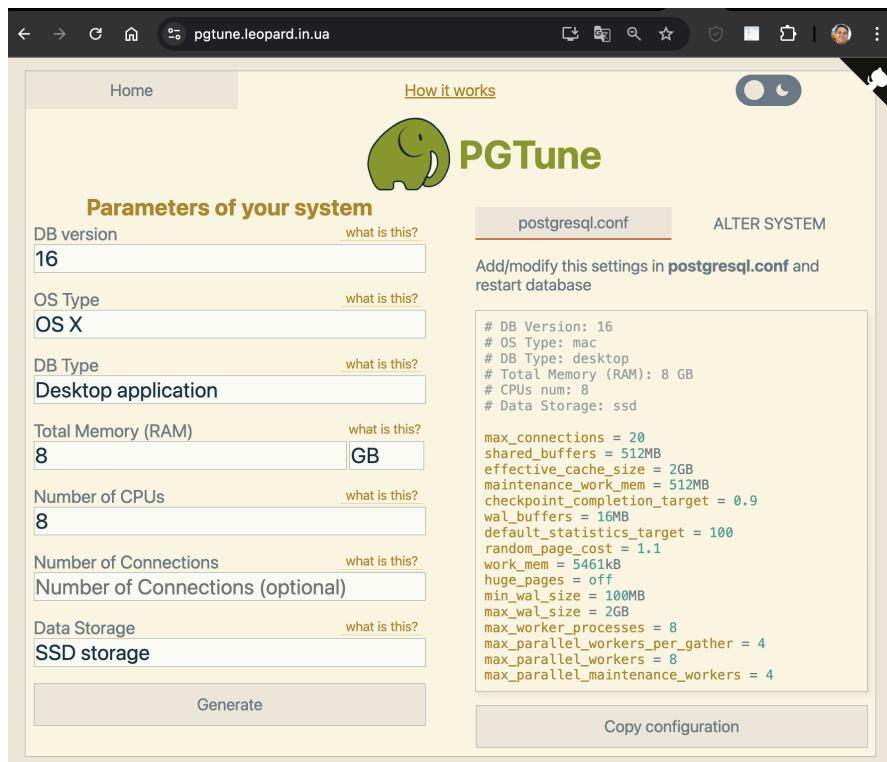
O otimizador escolhido pelo grupo foi o PgTune⁴, uma ferramenta de otimização de desempenho para o banco de dados PostgreSQL. Seus métodos de otimização se baseiam principalmente em Heurísticas Baseadas em Boas Práticas e Equilíbrio de Recursos do sistema (como memória e CPU), de maneira a evitar gargalos e maximizar o desempenho geral do banco de dados.

Para realizar essas otimizações, o otimizador ajusta automaticamente os parâmetros de configuração do PostgreSQL com base nas especificações do hardware do servidor e no tipo de carga de trabalho que o banco de dados deve suportar:

1. **Coleta de Informações do Sistema:** informações sobre o sistema onde o PostgreSQL está rodando, como a quantidade de memória RAM, o número de CPUs e o tamanho do armazenamento disponível.
2. **Perfil de Carga de Trabalho:** tipo de carga de trabalho que o banco de dados irá enfrentar, como "OLTP" (Transações Online), "DSS" (Sistemas de Suporte a Decisões), "Web", "Desenvolvimento", entre outros.
3. **Cálculos de Ajuste:** cálculo sobre as informações fornecidas. Os valores ideais para vários parâmetros de configuração do PostgreSQL, como `shared_buffers`, `work_mem`, `maintenance_work_mem`, `effective_cache_size`, entre outros.
4. **Geração de Configuração Otimizada:** arquivo de configuração (ou sugestões de parâmetros) que pode ser aplicado ao PostgreSQL para melhorar seu desempenho.

A imagem abaixo apresenta um exemplo de uso do software otimizador, onde inserimos as informações da máquina e ele retorna configurações que devem ser inseridas no arquivo `postgresql.conf`:

⁴ Disponível em: <https://pgtune.leopard.in.ua/>



The screenshot shows the PGtune web interface at pgtune.leopard.in.ua. The page displays various system parameters for tuning a PostgreSQL database. On the left, there's a form with fields for DB version (16), OS Type (OS X), DB Type (Desktop application), Total Memory (RAM) (8 GB), Number of CPUs (8), Number of Connections (optional), Data Storage (SSD storage), and a 'Generate' button. On the right, there are tabs for 'postgresql.conf' and 'ALTER SYSTEM'. The 'postgresql.conf' tab shows a configuration snippet with several parameters set to specific values. The 'ALTER SYSTEM' tab shows a list of commands that can be run to apply these settings.

```

# DB Version: 16
# OS Type: mac
# DB Type: desktop
# Total Memory (RAM): 8 GB
# CPUs num: 8
# Data Storage: ssd

max_connections = 20
shared_buffers = 512MB
effective_cache_size = 2GB
maintenance_work_mem = 512MB
checkpoint_completion_target = 0.9
wal_buffers = 16MB
default_statistics_target = 100
random_page_cost = 1.1
work_mem = 5461kB
huge_pages = off
min_wal_size = 100MB
max_wal_size = 2GB
max_worker_processes = 8
max_parallel_workers_per_gather = 4
max_parallel_workers = 8
max_parallel_maintenance_workers = 4

```

O software também retorna alguns comandos que podem ser executados, alterando o arquivo `postgresql.auto.conf`, que também é lido pelo SGBD:

- `ALTER SYSTEM SET max_connections = '20';`
- `ALTER SYSTEM SET shared_buffers = '512MB';`
- `ALTER SYSTEM SET effective_cache_size = '2GB';`
- `ALTER SYSTEM SET maintenance_work_mem = '512MB';`
- `ALTER SYSTEM SET checkpoint_completion_target = '0.9';`
- `ALTER SYSTEM SET wal_buffers = '16MB';`
- `ALTER SYSTEM SET default_statistics_target = '100';`
- `ALTER SYSTEM SET random_page_cost = '1.1';`
- `ALTER SYSTEM SET work_mem = '5461kB';`
- `ALTER SYSTEM SET huge_pages = 'off';`
- `ALTER SYSTEM SET min_wal_size = '100MB';`
- `ALTER SYSTEM SET max_wal_size = '2GB';`
- `ALTER SYSTEM SET max_worker_processes = '8';`
- `ALTER SYSTEM SET max_parallel_workers_per_gather = '4';`
- `ALTER SYSTEM SET max_parallel_workers = '8';`
- `ALTER SYSTEM SET max_parallel_maintenance_workers = '4';`

Consultas

Para analisar o tempo de execução, o grupo realizou 5 consultas diferentes, cada uma executada 10 vezes. Essas consultas incluíram operações de agrupamento, operações de conjunto, seleção de múltiplas colunas, junção interna e junção externa. Nenhuma consulta repetiu as mesmas operações. Para cada consulta, foi apresentado o tempo médio de 10 execuções, tanto antes da otimização, quanto depois dela. Todos os planos de execução foram gerados através do software PEV2.

Sem otimização

Para um primeiro momento, foram feitas as consultas sem otimização, ou seja, com os dados puros. O resultado se encontra nas tabelas abaixo.

1. Total de infrações por tipo de veículo e por radar: tempo médio de 720 ms.



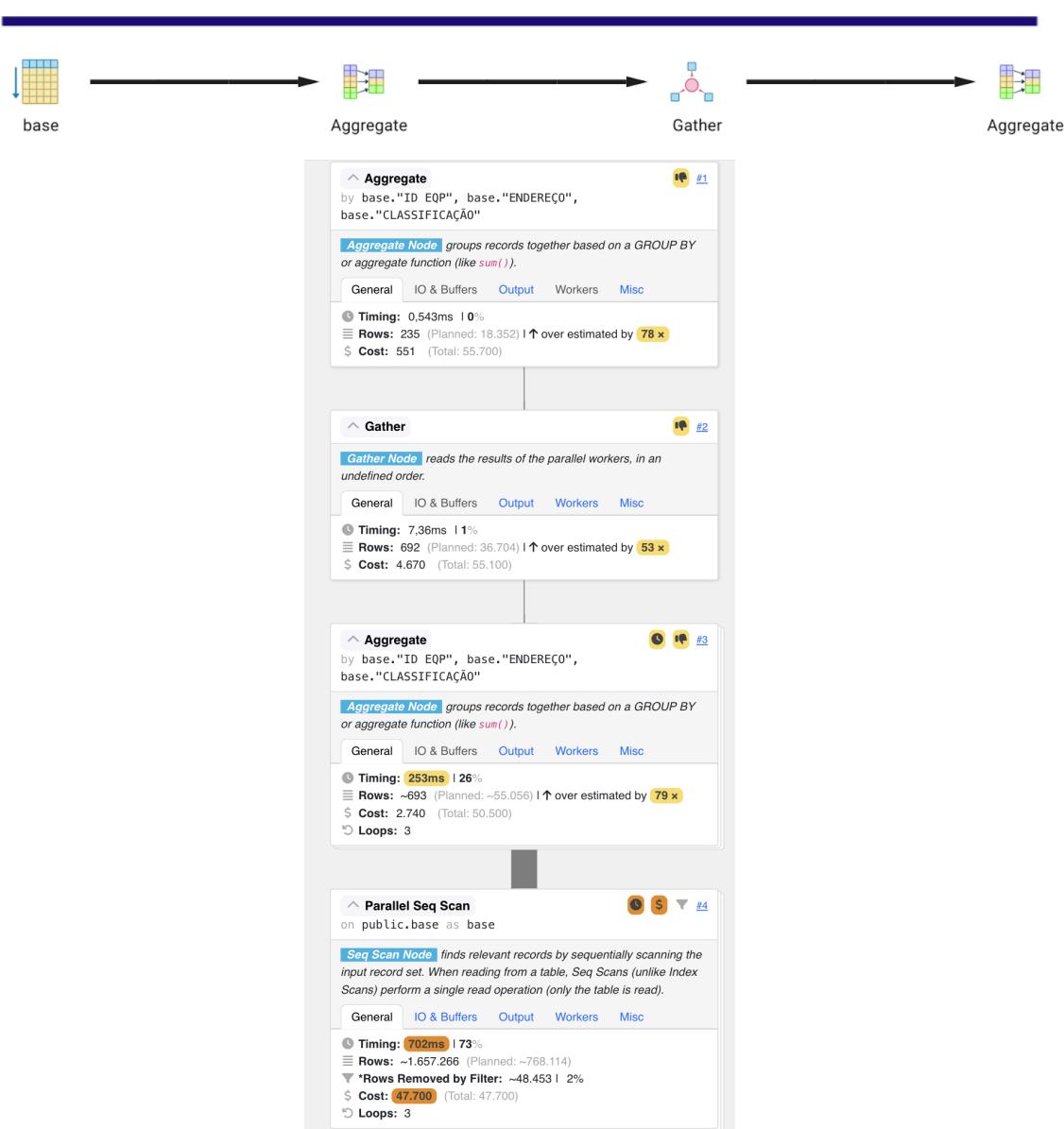
```

1 SELECT "ID EQP", "ENDERECO", "CLASSIFICAÇÃO", COUNT(*) as
   total_infracoes
2 FROM base
3 WHERE "VELOCIDADE DA VIA" > "VELOCIDADE AFERIDA" AND "VELOCIDADE
   AFERIDA" <= 200
4 GROUP BY "ID EQP", "ENDERECO", "CLASSIFICAÇÃO";

```

Consulta	Tempo (ms)
1	722
2	718
3	713
4	714
5	777
6	736
7	724
8	722
9	715
10	716

Mapa e plano de execução:



2. Velocidade média aferida em cada radar: tempo médio de 723 ms.

```

● ● ●

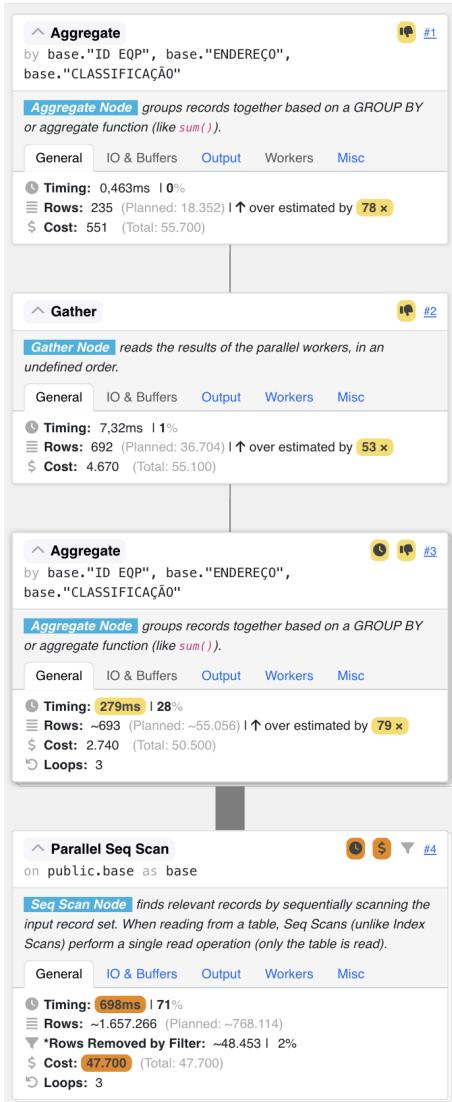
1 SELECT "ID_EQP", "ENDERECO", "CLASSIFICAÇÃO", COUNT(*) as
   total_infracoes
2 FROM base
3 WHERE "VELOCIDADE DA VIA" > "VELOCIDADE AFERIDA" AND "VELOCIDADE
   AFERIDA" <= 200
4 GROUP BY "ID_EQP", "ENDERECO", "CLASSIFICAÇÃO";

```

Consulta	Tempo (ms)
1	826
2	724
3	731
4	716
5	735
6	720
7	722
8	719
9	718
10	737

Mapa e plano de execução:





3. Os 5 primeiros radares que contém mais informações de leituras de veículos do tipo AUTOMÓVEL: tempo médio de 493 ms.



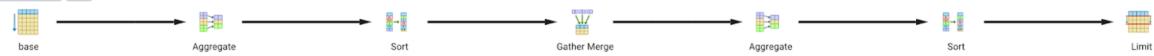
```

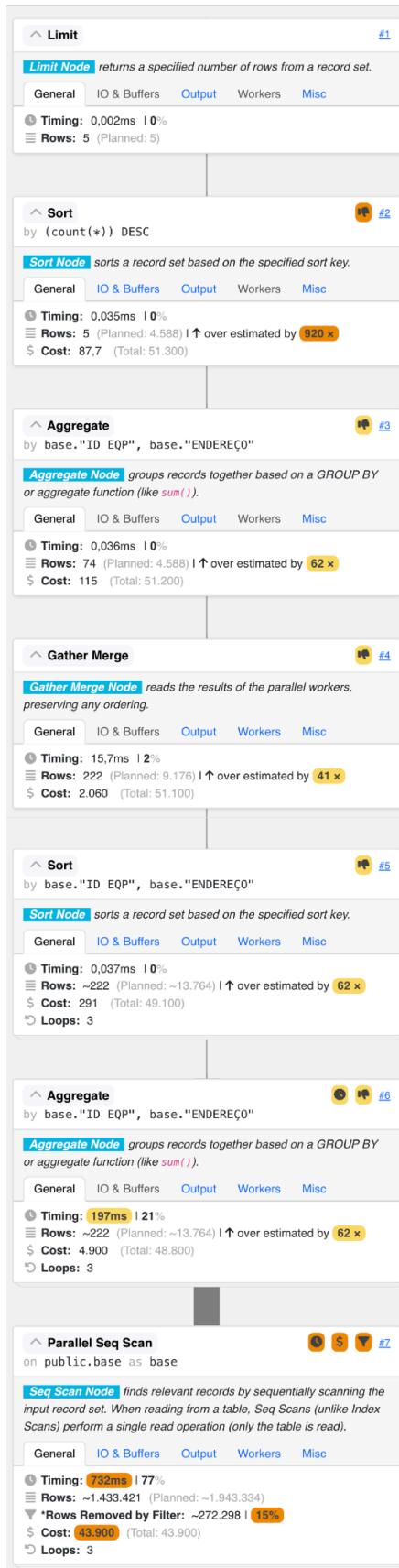
1 SELECT "ID_EQP", "ENDERECO", COUNT(*) as total_veiculos
2 FROM base
3 WHERE "CLASSIFICAÇÃO" = 'AUTOMÓVEL'
4 GROUP BY "ID_EQP", "ENDERECO"
5 ORDER BY total_veiculos DESC
6 LIMIT 5;

```

Consulta	Tempo (ms)
1	494
2	496
3	490
4	497
5	492
6	488
7	508
8	489
9	501
10	490

Mapa e plano de execução:





4. Média de tamanho de veículos em cada radar: tempo médio de 643 ms.



```

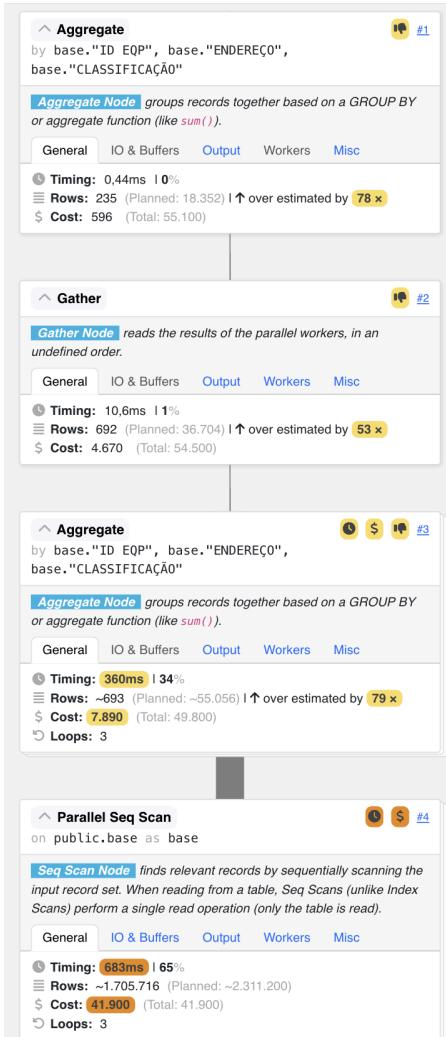
1 SELECT "ID_EQP", "ENDERECO", "CLASSIFICAÇÃO", AVG("TAMANHO") as
media_tamanho
2 FROM base
3 GROUP BY "ID_EQP", "ENDERECO", "CLASSIFICAÇÃO";

```

Consulta	Tempo (ms)
1	661
2	644
3	651
4	646
5	638
6	642
7	646
8	639
9	637
10	640

Mapa e plano de execução:





5. Média de tamanho dos veículos em todos os radares por classificação, exceto veículos do tipo MOTO: tempo médio de 368 ms.



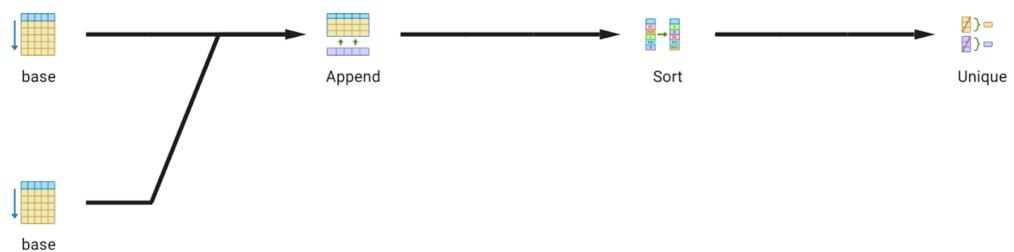
```

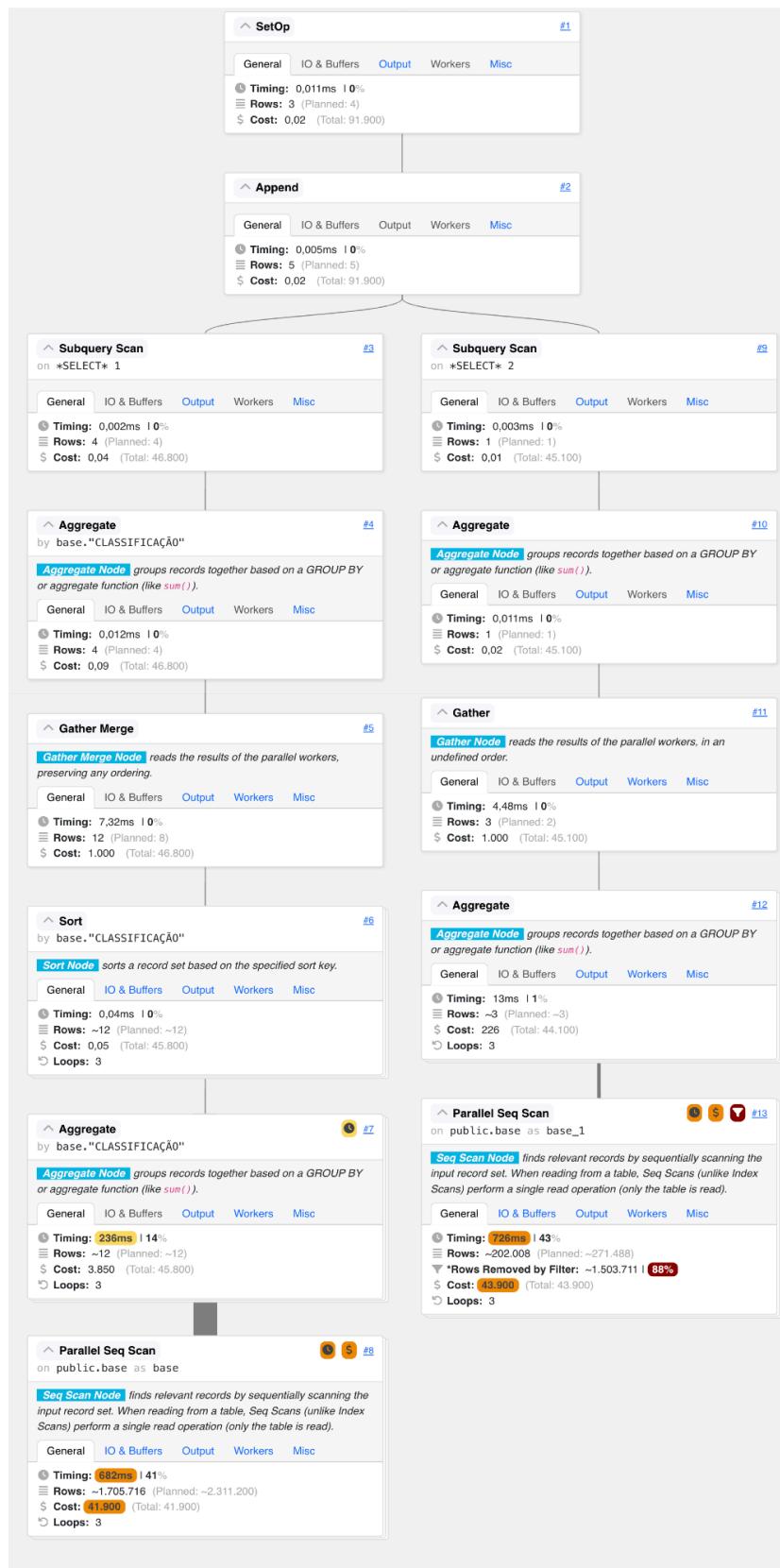
1 SELECT "CLASSIFICAÇÃO", AVG("TAMANHO") as media_tamanho
2 FROM base
3 GROUP BY "CLASSIFICAÇÃO"
4 EXCEPT
5 SELECT "CLASSIFICAÇÃO", AVG("TAMANHO") as media_tamanho
6 FROM base
7 WHERE "CLASSIFICAÇÃO" = 'MOTO'
8 GROUP BY "CLASSIFICAÇÃO";

```

Consulta	Tempo (ms)
1	4.785
2	4.600
3	4.700
4	4.586
5	4.801
6	4.589
7	4.829
8	4.581
9	4.790
10	4.607

Mapa e plano de execução:





Com otimização

Posteriormente, o grupo aplicou as seguintes otimizações para melhorar o desempenho da base de dados:

- Particionamento: dividimos a base de dados em duas tabelas:
 1. **Tabela de Radares**: endereço, longitude e latitude, id do equipamento, número de série e velocidade máxima.
 2. **Tabela de Leituras**: horário, faixa, velocidade aferida, classificação (AUTOMÓVEL, CAMINHÃO / ÔNIBUS, MOTO), tamanho e radar_id.
- Criação de Índices: aplicamos índices para acelerar as consultas ao permitir que o PostgreSQL encontre rapidamente as linhas desejadas, evitando varreduras completas das tabelas. Foram criados os índices:
 - `CREATE INDEX idx_leituras_radarid ON leituras (radar_id);`
 - `CREATE INDEX idx_radares_ideqp ON radares (id_eqp);`
 - `CREATE INDEX idx_leituras_velocidadeafeira ON leituras (velocidade_aferida);`
 - `CREATE INDEX idx_radares_velocidadepermitida ON radares (velocidade_permitida);`
- Execução de análises estatísticas:
 - `ANALYZE leituras;`
 - `ANALYZE radares;`

1. Total de infrações por tipo de veículo e por radar: tempo médio de 256 ms.

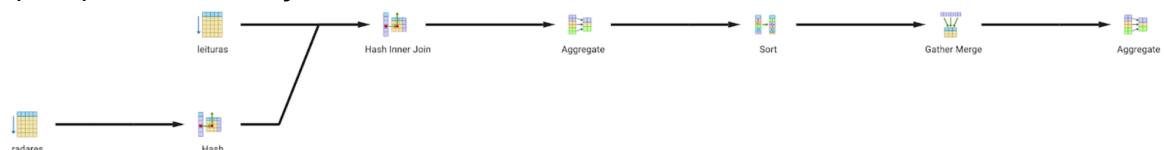


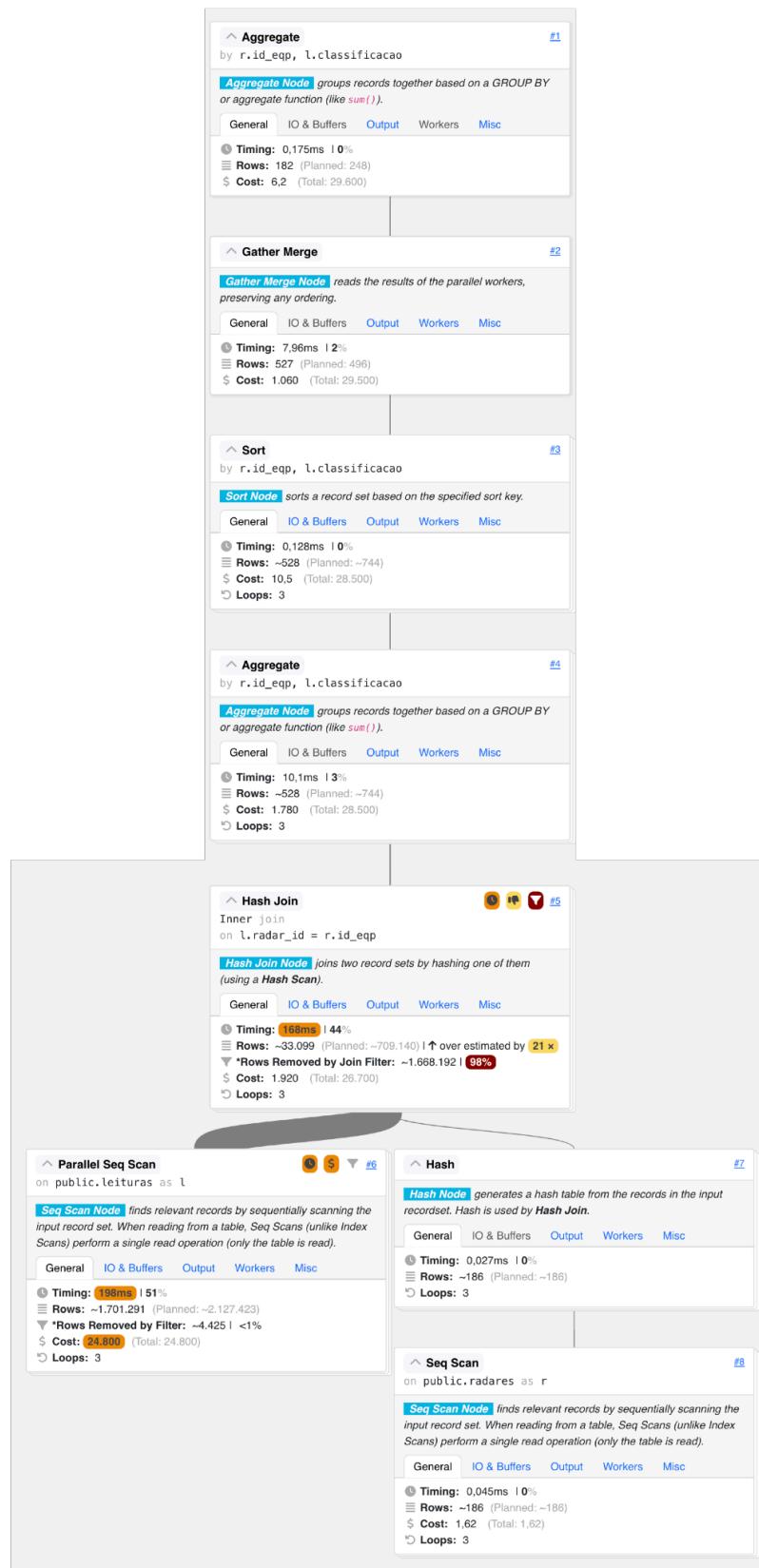
```
SELECT r.id_eqp, r.endereco, l.classificacao, COUNT(*) AS total_infracoes
FROM radares r
INNER JOIN leituras l ON r.id_eqp = l.radar_id
WHERE l.velocidade_aferida > r.velocidade_permitida
AND l.velocidade_aferida <= 200
GROUP BY r.id_eqp, r.endereco, l.classificacao;
```

Consulta	Tempo(ms) - Tabelas com melhorias de particionamento, índices e estatísticas
1	248
2	233
3	240

Consulta	Tempo(ms) - Tabelas com melhorias de particionamento, índices e estatísticas
4	280
5	280
6	279
7	258
8	265
9	251
10	254

Mapa e plano de execução:





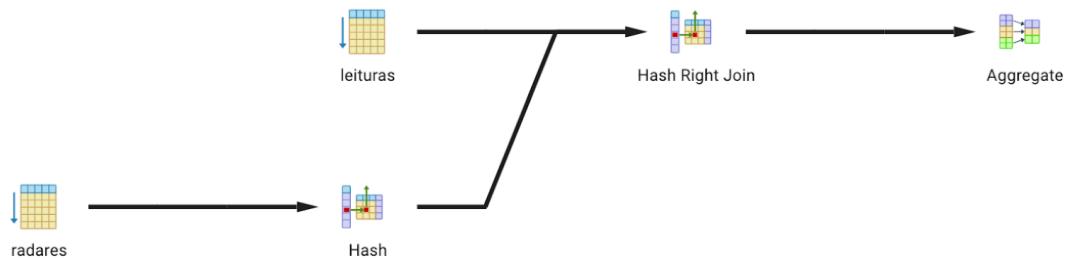
2. Velocidade média aferida em cada radar: tempo médio de 617.50 ms.

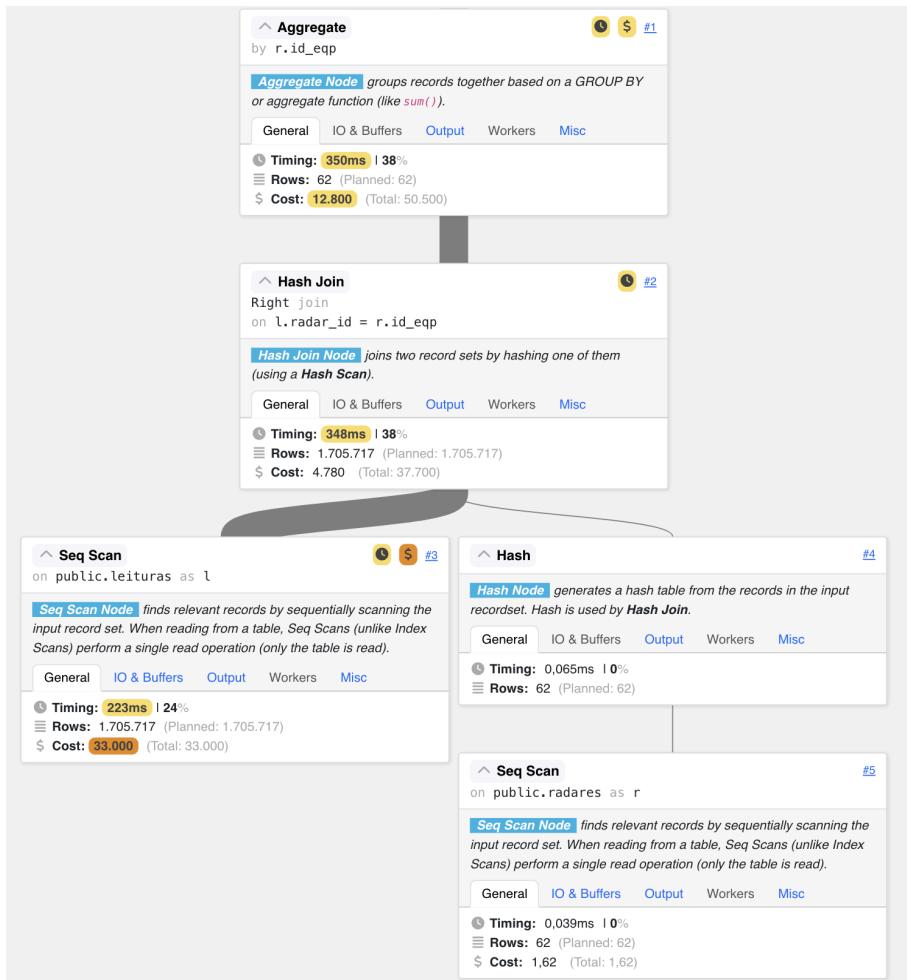


```
SELECT r.id_eqp, r.endereco, COALESCE(AVG(CASE WHEN l.velocidade_aferida <= 200 THEN
l.velocidade_aferida END), 0) as velocidade_media
FROM radares r
LEFT JOIN leituras l ON r.id_eqp = l.radar_id
GROUP BY r.id_eqp, r.endereco;
```

Consulta	Tempo(ms) - Tabelas com melhorias de particionamento, índices e estatísticas
1	602
2	632
3	629
4	620
5	645
6	615
7	630
8	610
9	610
10	594

Mapa e plano de execução:





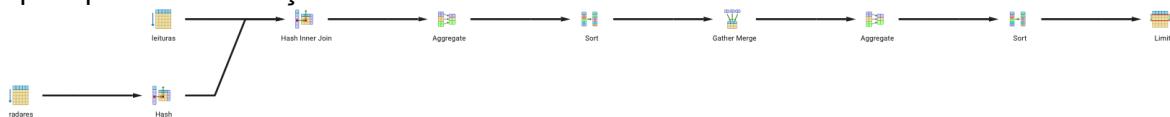
3. Os 5 primeiros radares que contém mais informações de leituras de veículos do tipo AUTOMÓVEL: tempo médio de 265.50 ms.

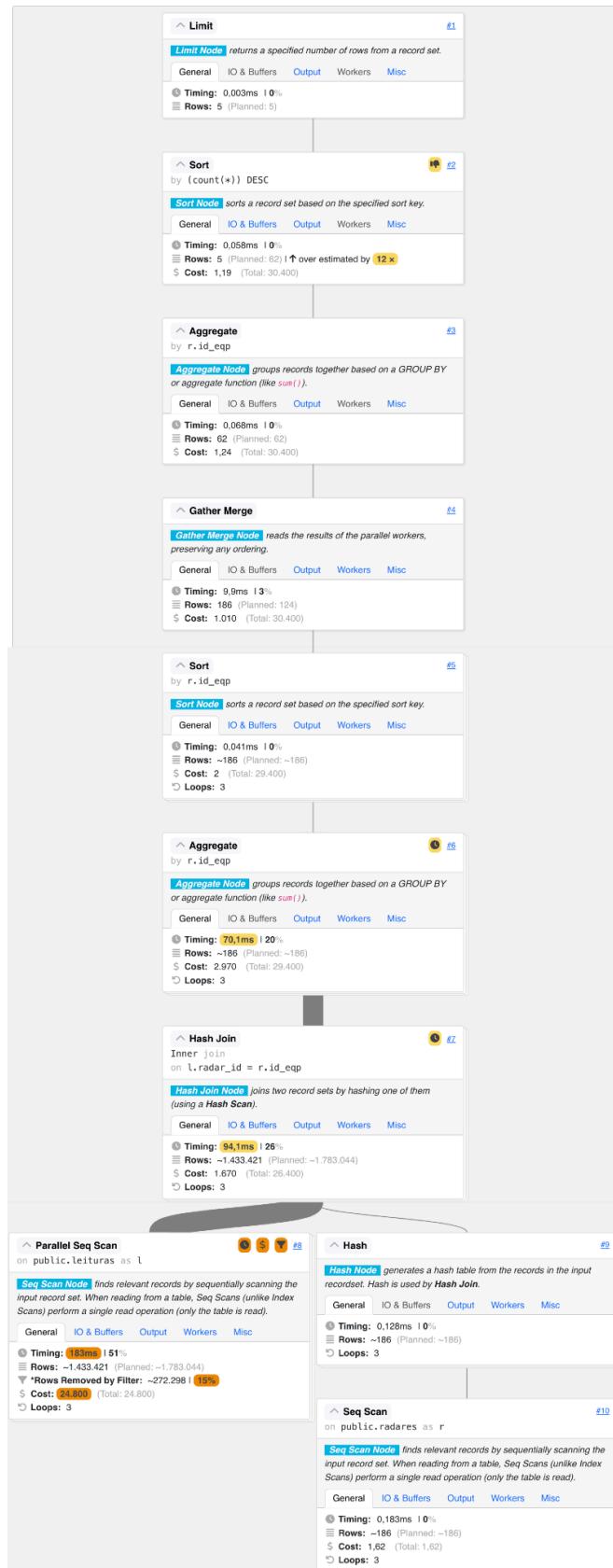
```

SELECT r.id_eqp, r.endereco, COUNT(*) as total_veiculos
FROM radares r
INNER JOIN leituras l ON r.id_eqp = l.radar_id
WHERE l.classificacao = 'AUTOMÓVEL'
GROUP BY r.id_eqp, r.endereco
ORDER BY total_veiculos DESC
LIMIT 5;
  
```

Consulta	Tempo(ms) - Tabelas com melhorias de particionamento, índices e estatísticas
1	320
2	245
3	267
4	236
5	234
6	243
7	273
8	302
9	262
10	307

Mapa e plano de execução:





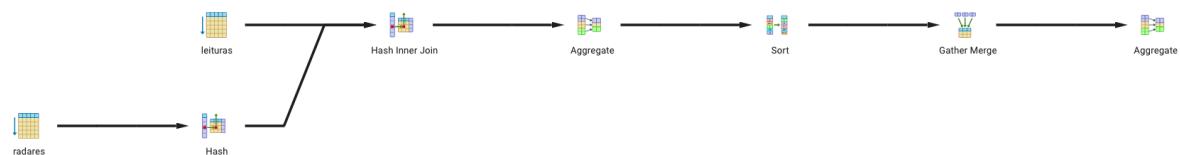
4. Média de tamanho de veículos em cada radar: tempo médio de 300 ms.

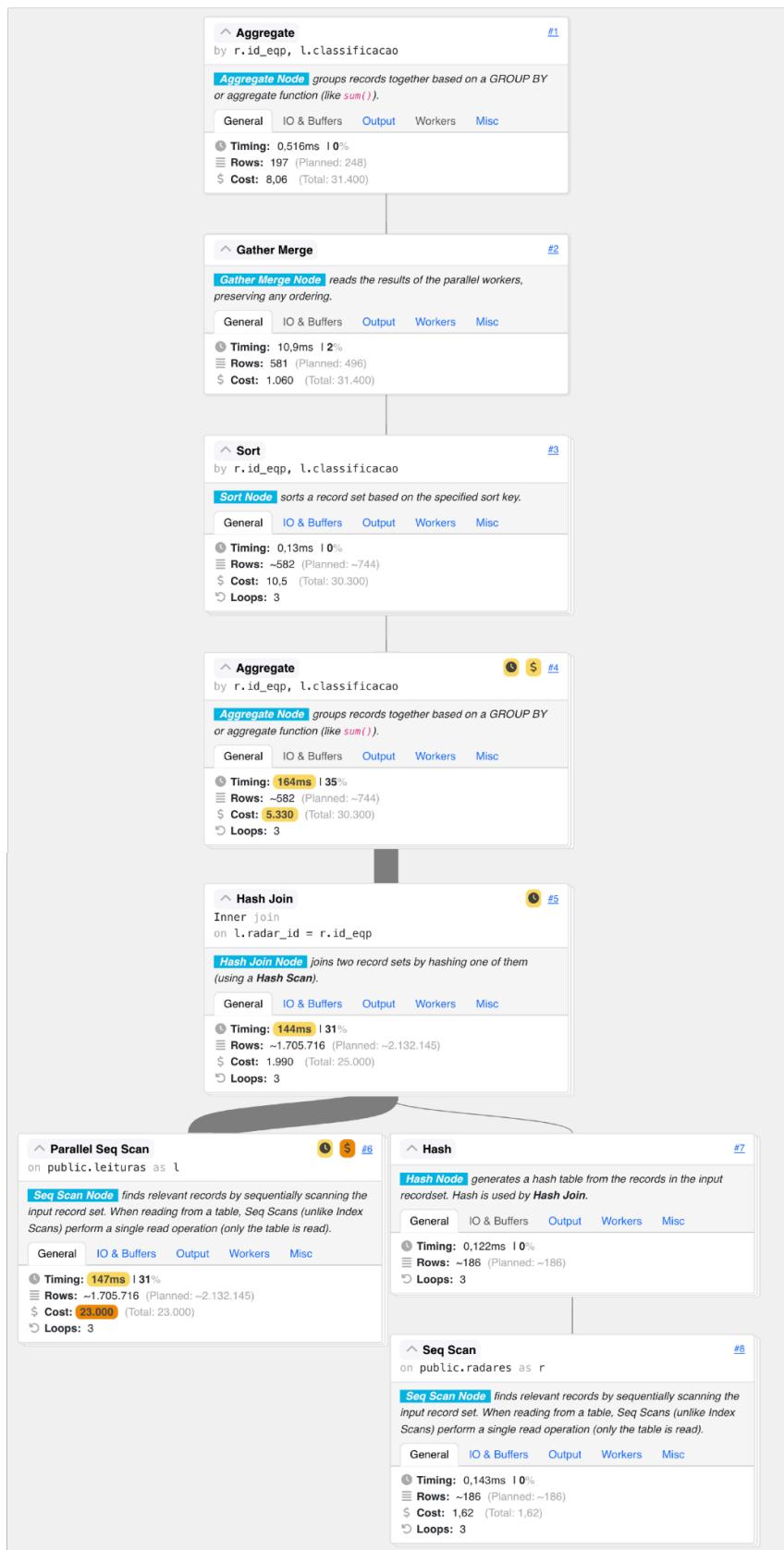


```
SELECT r.id_eqp, r.endereco, l.classificacao, AVG(l.tamanho) as
media_tamanho
FROM radares r
INNER JOIN leituras l ON r.id_eqp = l.radar_id
GROUP BY r.id_eqp, r.endereco, l.classificacao;
```

Consulta	Tempo(ms) - Tabelas com melhorias de particionamento, índices e estatísticas
1	297
2	300
3	311
4	319
5	290
6	305
7	300
8	289
9	281
10	307

Mapa e plano de execução:





5. Média de tamanho dos veículos em todos os radares por classificação, exceto veículos do tipo MOTO: tempo médio de 212 ms.



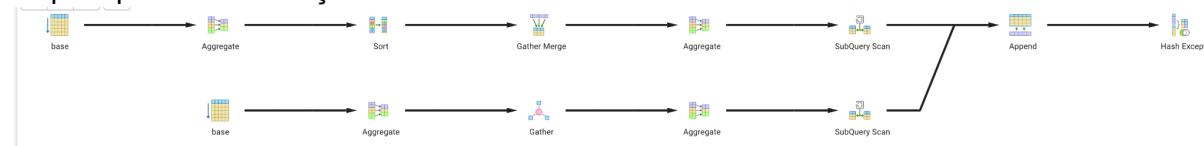
```

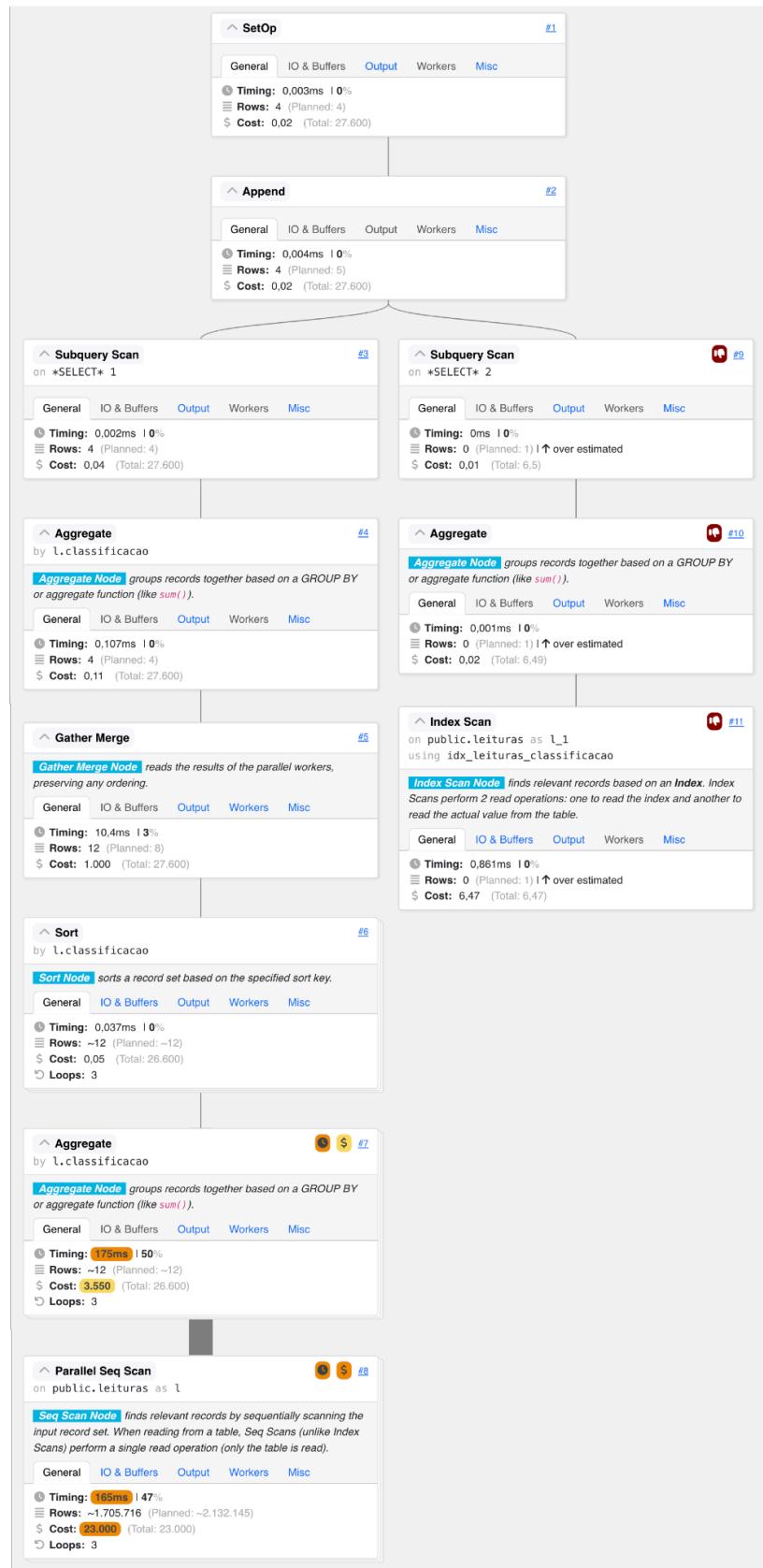
1 SELECT l.classificacao, AVG(l.tamanho) as media_tamanho
2 FROM leituras l
3 GROUP BY l.classificacao
4 EXCEPT
5 SELECT l.classificacao, AVG(l.tamanho) as media_tamanho
6 FROM leituras l
7 WHERE l.classificacao = 'moto'
8 GROUP BY l.classificacao;

```

Consulta	Tempo(ms) - Tabelas com melhorias de particionamento, índices e estatísticas
1	312
2	290
3	175
4	288
5	203
6	225
7	176
8	185
9	221
10	177

Mapa e plano de execução:





Aplicando um Otimizador

Por fim, o grupo aplicou as seguintes orientações dadas pelo otimizador PgTune. O resultado se encontra nas tabelas abaixo.

1. Total de infrações por tipo de veículo e por radar: tempo médio de 248.50 ms.

Consulta	Tempo(ms) - Otimizador aplicado
1	257
2	249
3	240
4	238
5	258
6	261
7	182
8	248
9	257
10	234

2. Velocidade média aferida em cada radar: tempo médio de 612.50 ms.

Consulta	Tempo(ms) - Otimizador aplicado
1	612
2	613
3	603
4	602
5	606
6	614
7	607
8	625

Consulta	Tempo(ms) - Otimizador aplicado
9	622
10	632

3. Os 5 primeiros radares que contém mais informações de leituras de veículos do tipo AUTOMÓVEL: tempo médio de 217 ms.

Consulta	Tempo(ms) - Otimizador aplicado
1	241
2	237
3	221
4	233
5	195
6	209
7	220
8	191
9	214
10	201

4. Média de tamanho de veículos em cada radar: tempo médio de 262 ms.

Consulta	Tempo(ms) - Otimizador aplicado
1	281
2	264
3	286
4	260
5	212
6	206
7	278

Consulta	Tempo(ms) - Otimizador aplicado
8	260
9	278
10	237

5. Média de tamanho dos veículos em todos os radares por classificação, exceto veículos do tipo MOTO: tempo médio de 173.5 ms.

Consulta	Tempo(ms) - Otimizador aplicado
1	175
2	164
3	152
4	142
5	255
6	215
7	208
8	150
9	172
10	195

Análises e conclusões

Foram considerados três cenários distintos: consultas sem otimizações, consultas com otimizações manuais e consultas com a aplicação de um otimizador externo. Após a execução de todas as consultas nesses três cenários, os resultados obtidos, baseados no tempo médio de execução de cada consulta, estão sintetizados na tabela abaixo:

Query	Tempo médio (ms) sem melhorias	Tempo médio (ms) com melhorias	Tempo(ms) - Otimizador aplicado	Ganho (ms)
1	720	256	248.5	471.5
2	723	617.5	612.5	110.5
3	493	264.5	217	276

Query	Tempo médio (ms) sem melhorias	Tempo médio (ms) com melhorias	Tempo(ms) - Otimizador aplicado	Ganho (ms)
4	643	300	262	381
5	368	212	173.5	194.5

As consultas sem otimizações apresentaram tempos médios de resposta relativamente altos, refletindo um desempenho sub ótimo do banco de dados PostgreSQL em seu estado padrão. Por exemplo, a Query 1 levou em média 720 ms para ser executada, enquanto a Query 2 teve um tempo médio de 723 ms.

Já com a aplicação de melhorias manuais nas consultas, observou-se uma significativa redução nos tempos médios de resposta. A Query 1, por exemplo, reduziu seu tempo médio de 720 ms para 256 ms, um ganho de 464 ms. A Query 4 também apresentou uma melhoria expressiva, com uma redução de 343 ms no tempo de resposta.

Por fim, a utilização de um otimizador externo, como o pgTune, proporcionou ganhos adicionais em relação às melhorias manuais. Na maioria das consultas, o tempo de resposta foi ainda mais reduzido. A Query 1, por exemplo, foi executada em 248.5 ms após a aplicação do otimizador externo, resultando em um ganho total de 471.5 ms em comparação com o cenário sem otimizações. A Query 3 apresentou um ganho de 276 ms, passando de 493 ms para 217 ms.

Esses resultados demonstram que a aplicação de técnicas de otimização, tanto manuais quanto automatizadas, pode trazer melhorias significativas no desempenho de consultas em um banco de dados PostgreSQL. Em particular, o uso de um otimizador externo mostrou-se altamente eficaz em maximizar esses ganhos, contribuindo para um ambiente de banco de dados mais eficiente e responsivo.