

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO RIO
GRANDE DO SUL - CAMPUS RIO GRANDE
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE
SISTEMAS

KESSLER FELIPE MENEGILDO DE OLIVEIRA

**PostgreSQL *Rewrite Advisor*: Uma
ferramenta automatizada para otimização
de consultas SQL no PostgreSQL**

Trabalho de Conclusão apresentado como
requisito parcial para a obtenção do grau de
Tecnólogo em Análise e Desenvolvimento de
Sistemas

Prof. Rafael Betito
Orientador

Rio Grande, Dezembro de 2016

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

de Oliveira, Kessler Felipe Menegildo

PostgreSQL *Rewrite Advisor*: Uma ferramenta automatizada para otimização de consultas SQL no PostgreSQL / Kessler Felipe Menegildo de Oliveira. – Rio Grande: TADS/IFRS, 2016.

74 f.: il.

Trabalho de Conclusão de Curso (tecnólogo) – Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul - Campus Rio Grande. Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Rio Grande, BR-RS, 2016. Orientador: Rafael Betito.

1. Banco de dados, Otimização, SQL. I. Betito, Rafael. II. Título.

FOLHA DE APROVAÇÃO

Monografia sob o título *"PostgreSQL Rewrite Advisor: Uma ferramenta automatizada para otimização de consultas SQL no PostgreSQL"*, defendida por Kessler Felipe Menegildo de Oliveira e aprovada em 19 de dezembro de 2016, em Rio Grande, estado do Rio Grande do Sul, pela banca examinadora constituída pelos professores:

Prof. Rafael Betito
Orientador

Prof. Márcio Josué Ramos Torres
IFRS - Câmpus Rio Grande

Prof. Igor Ávila Pereira
IFRS - Câmpus Rio Grande

Prof. Eduardo Nunes Borges
FURG

"Julgue seu sucesso pelas coisas que você teve que renunciar para conseguir."
— DALAI LAMA

AGRADECIMENTOS

Agradeço primeiramente a minha mãe, Katia Menegildo, que em todos os momentos soube compreender e motivar-me durante toda a graduação.

Ao meu orientador, Rafael Betito, pelo grande apoio e paciência na orientação que tornaram possível a conclusão desta monografia.

Aos professores do curso que sempre estiveram dispostos a ajudar e compartilhar seu conhecimento.

A meus colegas de classe, Aline Nörnberg e Jacques Schmitz, que me apoiaram nos momentos difíceis e me motivaram a continuar.

A meus amigos Catia Vieira, Geovanni Pacheco, Kaoê Menna, Vinícius Patzdor e Tobi pelo apoio durante toda a graduação.

Enfim, a todas as pessoas que direta ou indiretamente colaboraram para a elaboração deste trabalho.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS	8
LISTA DE FIGURAS	9
LISTA DE TABELAS	11
RESUMO	12
1 INTRODUÇÃO	13
2 FUNDAMENTAÇÃO TEÓRICA	15
2.1 Problemas de Desempenho em SGBDs	15
2.2 Otimização de Consultas SQL	16
2.2.1 Índices	16
2.2.2 Coleta de Estatísticas	17
2.2.3 Visões Materializadas	18
2.2.4 Reescrita de Consulta	18
2.2.5 Estratégias de Reescrita	18
3 SISTEMAS EXISTENTES	22
3.1 SQL Tuning Advisor (STA)	22
3.2 Database Engine Tuning Advisor (DTA)	24
3.3 Postgres Advanced Server (PAS)	25
3.4 Quadro Comparativo	25
4 O POSTGRESQL REWRITE ADVISOR (PGRA)	26
4.1 Diagrama de Casos de Uso	26
4.1.1 Usuário	26
4.1.2 Banco de dados	27
4.2 Diagrama de Classes	27
4.2.1 A Classe Query	27
4.2.2 A Classe Table	29
4.2.3 A Classe Column	31
4.2.4 A Classe Database	31
4.2.5 A Classe PGRA	32
4.2.6 A Interface Refactor	33
4.2.7 A Classe Refactor_col_sub_aco	34
4.2.8 A Classe Refactor_where_sub_aco	35
4.2.9 A Classe PgQuery	36

4.2.10	A Classe JSON	36
4.2.11	A Classe Base	36
4.2.12	A Classe Rule	37
4.2.13	A Classe Formatter	37
4.3	Fluxograma de Funcionamento	38
4.4	Ferramentas Utilizadas	38
4.4.1	PostgreSQL	38
4.4.2	Ruby	38
4.4.3	Sinatra	39
4.4.4	Simple Sinatra MVC Template	39
4.4.5	pg_query	39
4.4.6	Materialize	41
5	RESULTADOS	42
5.1	O Protótipo	42
5.2	Casos de Testes	46
5.2.1	Primeiro caso de teste	46
5.2.2	Segundo caso de teste	50
5.2.3	Terceiro caso de teste	53
5.2.4	Quarto caso de teste	53
5.2.5	Quinto caso de teste	53
5.2.6	Sexto caso de teste	53
5.2.7	Quadro Comparativo	62
5.3	Restrições	62
6	CONCLUSÃO	63
	REFERÊNCIAS	65
	GLOSSÁRIO	67
APÊNDICE A	EXEMPLOS DE CONSULTAS SQL EQUIVALENTES COM DIFERENTES PERFORMANCES	68
A.1	Consultas Equivalentes	68
A.2	Estatísticas das Consultas no PostgreSQL	73
A.3	Comparação dos Tempos de Execução entre SGBDs	73

LISTA DE ABREVIATURAS E SIGLAS

DSL	Linguagem de Domínio Específico(<i>Domain-Specific Language</i>)
DTA	Database Engine Tuning Advisor
HTTP	Protocolo de transferência de hipertexto (<i>hypertext transfer protocol</i>)
MVC	Modelo-Visão-Controlador (<i>Model-View-Controller</i>)
PAS	Postgres Advanced Server
PGRA	PostgreSQL Rewrite Advisor
SGBD	Sistema de Gerenciamento de Banco de Dados (<i>Database Management System</i>)
SQL	Linguagem de Consulta Estruturada (<i>Structured Query Language</i>)
STA	SQL Tuning Advisor

LISTA DE FIGURAS

Figura 1.1:	Tipos de problemas de desempenho em consultas no SGBD. Fonte: (IKE, 2007)	13
Figura 2.1:	Processamento de uma consulta pelo SGBD.	16
Figura 2.2:	Comparação de consultas com índices.	17
Figura 2.3:	Comparação de consultas com expansão de OR.	19
Figura 2.4:	Comparação de consultas com desassociar subconsultas.	20
Figura 2.5:	Comparação de consultas com transformação estrela.	21
Figura 3.1:	Primeira consulta para os casos de testes do SQL Tuning Advisor. . .	23
Figura 3.2:	Captura de tela dos resultados do SQL Tuning Advisor para a primeira consulta.	23
Figura 3.3:	Segunda consulta para os casos de testes do SQL Tuning Advisor. . .	23
Figura 3.4:	Captura de tela dos resultados do SQL Tuning Advisor para a segunda consulta.	24
Figura 3.5:	Consulta do caso de teste do Database Engine Tuning Advisor. . . .	24
Figura 3.6:	Captura de tela dos resultados do Database Engine Tuning Advisor para a consulta.	25
Figura 4.1:	Diagrama de casos de Uso.	26
Figura 4.2:	Diagrama de classes simplificado.	28
Figura 4.3:	Diagrama da classe Query.	30
Figura 4.4:	Diagrama da classe Table.	30
Figura 4.5:	Diagrama da classe Column.	31
Figura 4.6:	Diagrama da classe Database.	32
Figura 4.7:	Diagrama da classe PGRA.	33
Figura 4.8:	Diagrama da interface Refactor.	33
Figura 4.9:	Consulta de exemplo para a classe Refactor_col_sub_aco.	34
Figura 4.10:	Pseudocódigo do identify da classe Refactor_col_sub_aco.	34
Figura 4.11:	Pseudocódigo do rewrite da classe Refactor_col_sub_aco.	34
Figura 4.12:	Consulta de exemplo para a classe Refactor_where_sub_aco.	35
Figura 4.13:	Pseudocódigo do identify da classe Refactor_where_sub_aco.	35
Figura 4.14:	Pseudocódigo do rewrite da classe Refactor_where_sub_aco.	35
Figura 4.15:	Diagrama da classe PgQuery.	36
Figura 4.16:	Diagrama da classe JSON.	36
Figura 4.17:	Diagrama da classe Base.	37
Figura 4.18:	Diagrama da classe Rule.	37
Figura 4.19:	Diagrama da classe Formatter.	37

Figura 4.20:	Fluxograma de funcionamento.	38
Figura 4.21:	Primeira consulta de exemplo do pg_query.	39
Figura 4.22:	Segunda consulta de exemplo do pg_query.	40
Figura 5.1:	Tela inicial do protótipo sem conexões cadastrada.	43
Figura 5.2:	Tela de cadastro de conexão do protótipo.	43
Figura 5.3:	Tela inicial do protótipo com conexões cadastrada.	43
Figura 5.4:	Visualizar uma conexão cadastrada no protótipo.	44
Figura 5.5:	Consulta de exemplo do protótipo PGRA.	44
Figura 5.6:	Tela para submeter uma consulta do protótipo.	44
Figura 5.7:	Tela de progresso de uma consulta do protótipo.	45
Figura 5.8:	Tela de resultados de uma consulta do protótipo.	45
Figura 5.9:	Diagrama Relacional.	46
Figura 5.10:	Consulta submetida do primeiro caso de teste.	47
Figura 5.11:	Primeiro caso de teste ao executar a linha 3 do pseudocódigo.	47
Figura 5.12:	Primeiro caso de teste ao executar a linha 6 do pseudocódigo.	47
Figura 5.13:	Primeiro caso de teste ao executar a linha 7 do pseudocódigo.	48
Figura 5.14:	Primeiro caso de teste ao executar a linha 8 do pseudocódigo.	48
Figura 5.15:	Primeiro caso de teste ao executar a linha 9 do pseudocódigo.	48
Figura 5.16:	Primeiro caso de teste ao executar a linha 10 do pseudocódigo.	49
Figura 5.17:	Tela de resultados do primeiro caso de teste.	49
Figura 5.18:	Consulta submetida do segundo caso de teste.	50
Figura 5.19:	Segundo caso de teste ao executar a linha 3 do pseudocódigo.	50
Figura 5.20:	Segundo caso de teste ao executar a linha 6 do pseudocódigo.	51
Figura 5.21:	Segundo caso de teste ao executar a linha 7 do pseudocódigo.	51
Figura 5.22:	Segundo caso de teste ao executar a linha 8 do pseudocódigo.	51
Figura 5.23:	Segundo caso de teste ao executar a linha 9 do pseudocódigo.	52
Figura 5.24:	Segundo caso de teste ao executar a linha 10 do pseudocódigo.	52
Figura 5.25:	Tela de resultados do segundo caso de teste.	52
Figura 5.26:	Comparação de consultas do terceiro caso de teste.	54
Figura 5.27:	Tela de resultados do terceiro caso de teste.	55
Figura 5.28:	Comparação de consultas do quarto caso de teste.	56
Figura 5.29:	Tela de resultados do quarto caso de teste.	57
Figura 5.30:	Consulta submetida do quinto caso de teste.	57
Figura 5.31:	Consultas do quinto caso de teste ao realizar a primeira reestruturação.	58
Figura 5.32:	Tela de resultados do quinto caso de teste ao realizar a primeira reestruturação.	59
Figura 5.33:	Consulta do quinto caso de teste ao realizar a segunda reestruturação.	59
Figura 5.34:	Tela de resultados do quinto caso de teste ao realizar a segunda reestruturação.	60
Figura 5.35:	Consulta submetida do sexto caso de teste.	60
Figura 5.36:	Tela de resultados do sexto caso de teste.	61

LISTA DE TABELAS

Tabela 2.1:	Tipo de estatísticas coletadas pelos SGBDs.	17
Tabela 3.1:	Ferramentas para aumento de desempenho de consultas SQL.	25
Tabela 5.1:	Volumetria para cada tabela.	46
Tabela 5.2:	Comparativo entre os tempos dos casos de testes.	62

RESUMO

Dentre 60% a 80% dos problemas no desempenho da recuperação de dados em SGBDs relacionais, são gerados pelo mau uso da SQL, quando o desenvolvedor escreve consultas sem preocupação com otimização. O mau uso da linguagem SQL pelos desenvolvedores não permite que o SGBD encontre o plano mais otimizado para uma consulta. Alguns SGBDs possuem ferramentas para sugestão de melhorias de aumento de desempenho como criação de índices, coleta de estatísticas, criação de visões materializadas e reescrita de consultas SQL. Reescrita de consultas SQL é analisar e transformar uma consulta em outra equivalente objetivando aumento de desempenho. Para o SGBD PostgreSQL, não foi encontrada no repositório oficial uma ferramenta para tratar com reescrita de consulta o problema do mau uso da linguagem SQL. Neste trabalho foram estudadas algumas estratégias de otimização por reescrita encontradas na bibliografia. Foi desenvolvida em Ruby uma ferramenta automatizada e interativa para melhoria do desempenho por reescrita de consultas SQL para o SGBD PostgreSQL. A ferramenta foi implementada de forma a permitir a adição de novos módulos de otimização por reescrita, desde que respeitem as regras definidas. Por fim, foram realizados testes de validação onde a ferramenta foi bem-sucedida identificando corretamente as situações de reescrita e reestruturando a consulta SQL para uma forma mais eficiente.

Palavras-chave: Banco de dados, Otimização, SQL.

1 INTRODUÇÃO

Um banco de dados pode ser definido como sendo uma coleção de dados relacionados entre si. Bancos de dados podem variar de tamanho e complexidade dependendo do caso. Um banco de dados pode ser criado e gerenciado manualmente ou através de um sistema computadorizado. (ELMASRI; NAVATHE, 2011)

Dentre os sistemas computadorizados mais comuns estão os sistemas gerenciadores de banco de dados (SGBDs). Um SGBD é um conjunto de programas para manutenção e recuperação de dados armazenados em um banco de dados. Estas ações são realizadas a partir da linguagem de consulta *Structured Query Language* (SQL). A SQL se baseia em comandos que utilizam álgebra relacional e cálculo relacional. (SILBERSCHATZ; KORTH; SUDARSHAN, 2006)

Para executar cada consulta SQL, um certo esforço computacional é necessário e, quanto maior o esforço computacional, maior o tempo gasto. Os problemas mais comuns relacionados à demora na resposta, são apresentados na figura 1.1. Destes problemas, o de maior ocorrência é o mau uso da linguagem SQL, ou seja, escrita da consulta sem preocupação com otimização. (IKE, 2007)

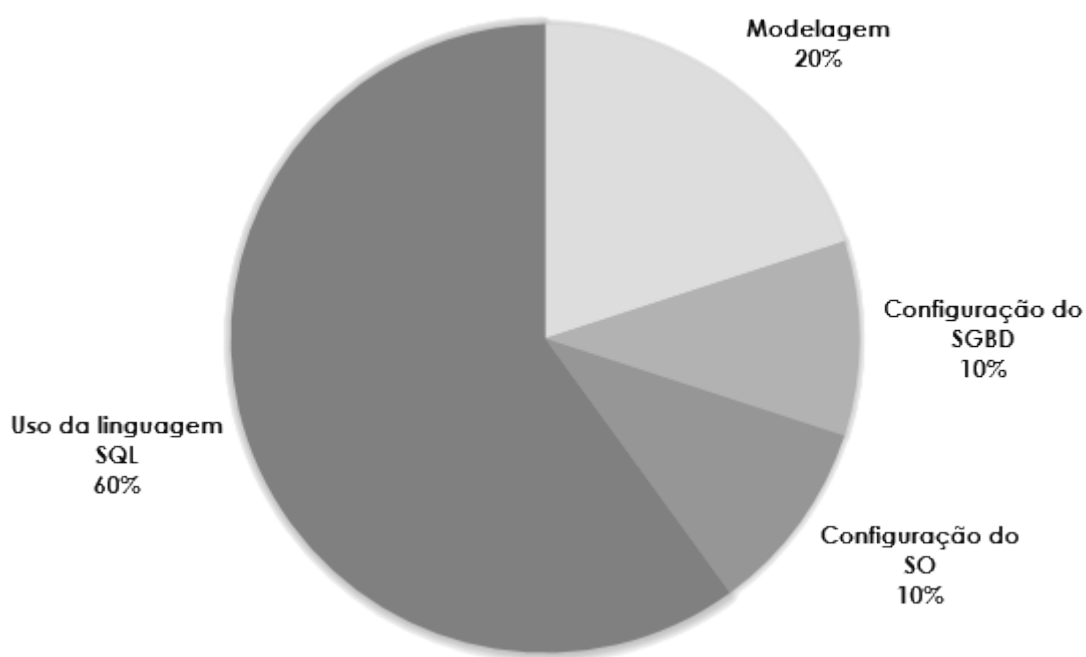


Figura 1.1: Tipos de problemas de desempenho em consultas no SGBD. Fonte: (IKE, 2007)

Na linguagem SQL, consultas diferentes que produzem exatamente o mesmo conjunto de dados, são denominadas equivalentes. Todavia, consultas equivalentes podem demandar esforços computacionais muito diferentes. (SILBERSCHATZ; KORTH; SUDARSHAN, 2006) O apêndice A apresenta 12 consultas equivalentes distintas com diferentes esforços computacionais.

Os SGBDs não esperam que o desenvolvedor escreva a consulta da forma mais eficiente para retornar um determinado conjunto de dados, pois é necessário profundo conhecimento da SQL. Para amenizar isto, alguns SGBDs realizam otimizações na tentativa de reduzir o esforço computacional necessário. Entretanto, o mau uso da linguagem SQL pelos desenvolvedores nem sempre permite que a forma mais otimizada seja alcançada. (SILBERSCHATZ; KORTH; SUDARSHAN, 2006)

Segundo MULLINS (1998), dentre 70% a 80% dos problemas no desempenho do banco de dados provém do mau uso da SQL. ANDRADE (2005) reforça, que 60% a 90% dos problemas de desempenho são causados por instruções SQL e índices.

Apesar das otimizações feitas pelos SGBDs, nem sempre isto é suficiente para obter um real ganho de desempenho. Alguns SGBDs, como Oracle e Microsoft SQL Server, possuem ferramentas externas para uma análise mais profunda da consulta e sugestão de melhorias para aumento de desempenho, como a reescrita de consulta SQL. (CORADINI; CANTARELLI, 2012)

Analisar e transformar uma consulta em outra equivalente, de forma automática ou manual, objetivando aumento de desempenho é denominado *reescrita de consulta*. (SILBERSCHATZ; KORTH; SUDARSHAN, 2006)

Um SGBD para o qual não foi encontrada no repositório oficial uma ferramenta para tratar com reescrita de consulta o problema do mau uso da linguagem SQL é o PostgreSQL, um SGBD objeto-relacional de código aberto utilizado por grandes empresas. (The PostgreSQL Global Development Group, 2016a)

Este trabalho objetiva desenvolver uma ferramenta automatizada e interativa para melhoria do desempenho por reescrita de consultas SQL para o SGBD PostgreSQL.

Para alcançar isto, foram estabelecidos os objetivos específicos:

- Estudar problemas relacionados ao desempenho de consultas SQL;
- Estudar formas de otimização de consultas;
- Analisar programas existentes de otimização de consulta;
- Experimentar reescrita de consultas SQL que gerem aumento de desempenho;
- Implementar um protótipo de ferramenta para melhoria de desempenho de consultas para o SGBD PostgreSQL;
- Validar o protótipo utilizando casos de teste;

Este trabalho está organizado em seis capítulos. No segundo capítulo é apresentado a fundamentação teórica, destacando os problemas de desempenho em SGBDs e as formas de otimização de consultas SQL. No terceiro capítulo são abordados os sistemas existentes com foco em otimização de consultas SQL. No quarto capítulo é descrito como foi projetado o protótipo proposto. No quinto capítulo é apresentado o protótipo desenvolvido e os resultados obtidos. Por fim, no sexto capítulo são expostas as conclusões finais.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão abordados conceitos para o entendimento dos problemas de desempenho existentes em SGBDs e das técnicas de otimização de consultas SQL.

2.1 Problemas de Desempenho em SGBDs

Diversos aspectos podem interferir no desempenho de um SGBD. Consultas escritas de forma não otimizada, parâmetros de configuração do próprio SGBD, parâmetros de configuração do sistema operacional e especificação do *hardware* podem causar perda de desempenho. (SILBERSCHATZ; KORTH; SUDARSHAN, 2006) (IKE, 2007)

A configuração do sistema operacional, se feita de forma incorreta, afeta o desempenho de todo o SGBD. A política de escalonamento das CPUs, a quantidade de memória máxima permitida para cada processo, o tamanho das páginas de memória e o tipo de sistema de arquivos utilizado são alguns exemplos de parâmetros que influenciam diretamente o desempenho do SGBD. (LÓPEZ; DILL, 2012) (SANTOS CANEDO et al., 2013) (ROB; CORONEL, 2011)

Não ajustar os parâmetros do SGBD durante a instalação, também afeta o desempenho de todo SGBD. A quantidade máxima de conexões por clientes, a quantidade de memória utilizada, a forma de escrita de dados no disco e o tempo máximo de espera para cada requisição são alguns exemplos de parâmetros que também influenciam diretamente o desempenho do SGBD. (TRAMONTINA, 2008)

A modelagem de um banco de dados afeta diretamente o desempenho das consultas. O desrespeito às formas normais, a tipagem incorreta e a redundância de dados são alguns exemplos de problemas de modelagem que afetam o desempenho das consultas. Entretanto, mesmo bancos de dados que não possuem estes problemas, ainda podem apresentar problemas de desempenho nas consultas. Algumas consultas podem ser complexas e necessitar de índices não previstos no projeto, por não afetar a integridade referencial dos dados. (ELMASRI; NAVATHE, 2011)

Um outro problema que afeta o desempenho das consultas é o mau uso da linguagem SQL. O mau uso ocorre quando o desenvolvedor não tem o conhecimento ou a experiência necessária para escrever uma consulta da forma mais otimizada. Isto é ainda agravado pelo fato de que, na SQL, um mesmo conjunto de dados pode ser obtido com diversas consultas diferentes com esforços computacionais distintos. (ROB; CORONEL, 2011)

Para entender como consultas diferentes possuem esforços computacionais distintos, apesar de resultarem exatamente no mesmo conjunto de dados, é preciso entender como uma consulta é processada pelo SGBD. Primeiro, na etapa de tradução, a sintaxe da consulta SQL é validada e a consulta é traduzida para álgebra relacional. Em seguida, na etapa de otimização, para cada operação da expressão, é verificado quais algoritmos po-

dem ser utilizados. A cada algoritmo está associado um custo. É denominado plano de execução uma sequência de algoritmos que pode ser utilizada para avaliar uma consulta. A soma dos custos de cada algoritmo determina o custo total do plano. Encerrando a etapa de otimização, é escolhido o plano de execução de menor custo. Por último, na etapa de avaliação, o plano é processado, os dados são obtidos das bases e o resultado é enviado ao usuário. (SILBERSCHATZ; KORTH; SUDARSHAN, 2006) A figura 2.1 ilustra este processo.

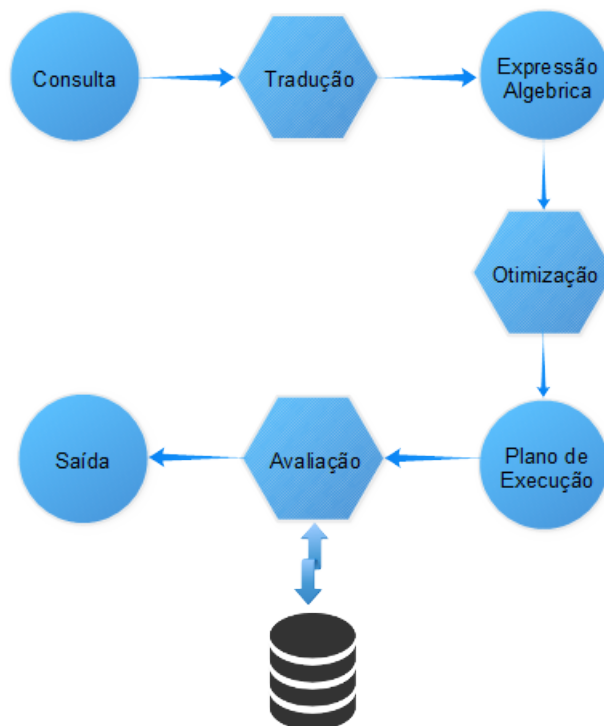


Figura 2.1: Processamento de uma consulta pelo SGBD.

Entretanto, mesmo com o otimizador escolhendo o plano com menor custo, a escolha está limitada aos planos possíveis da consulta submetida. Para encontrar planos melhores é necessário realizar uma reestruturação ou reescrita da consulta SQL. (ROB; CORONEL, 2011)

2.2 Otimização de Consultas SQL

Esta seção aborda as principais formas de otimização de consultas SQL.

2.2.1 Índices

O uso de índices acelera o acesso aos dados, melhorando o processo de busca. Um índice é um conjunto ordenado de valores chave associados a linhas em uma tabela e organizados de forma a diminuir a quantidade de avaliações necessárias para encontrar um dado. Índices de *hash*, árvores B, árvores B+ e mapas de bits são alguns exemplos de índices utilizados em SGBDs. (ROB; CORONEL, 2011) (SILBERSCHATZ; KORTH; SUDARSHAN, 2006)

Entretanto, a existência de índices nem sempre é uma garantia de melhoria de desempenho. Na figura 2.2, a consulta (a) possui desempenho pior do que a consulta (b)

apesar de **campo_b** e **campo_c** possuírem índices. Isto ocorre, pois, a presença do OR na primeira consulta força uma busca sequencial. (ELMASRI; NAVATHE, 2011)

01. SELECT campo_a	01. (SELECT campo_a
02. , campo_b	02. , campo_b
03. , campo_c	03. , campo_c
04. FROM table	04. FROM table
05. WHERE campo_b > 50	05. WHERE campo_b > 50)
06. OR campo_c < 5000	06. UNION
	07. (SELECT campo_a
	08. , campo_b
	09. , campo_c
	10. FROM table
	11. WHERE campo_c < 5000)

(a)
(b)

Figura 2.2: Comparação de consultas com índices.

2.2.2 Coleta de Estatísticas

Na etapa de otimização descrita na figura 2.1 a escolha por um dos diversos algoritmos para cada operação da expressão é baseada nas estatísticas coletadas sobre o banco. A tabela 2.1 mostra algumas estatísticas coletadas pelos SGBDs. Embora muitos SGBDs ofereçam suporte a coleta de estatísticas automaticamente, outros exigem que esta seja coletada manualmente, através do comando ANALYZE. (ROB; CORONEL, 2011)

Tabela 2.1: Tipo de estatísticas coletadas pelos SGBDs.

Objeto de banco de dados	Tipo de estatísticas
Tabelas	Número de linhas, número de blocos de disco utilizados, comprimento de linha, número de colunas em cada linha, número de valores distintos em cada coluna, valor máximo em cada coluna, valor mínimo em cada coluna e colunas que possuem índices.
Índices	Número e nome de colunas no índice, número de valores no índice, número de valores distintos no índice, histograma de valores do índice e número de páginas de disco utilizadas pelo índice.
Recursos do Ambiente	Tamanho físico e lógico de blocos de disco, localização e tamanho de arquivos de dados e número de expansões por arquivo de dados.

Porém, o otimizador pode não escolher o melhor algoritmo para uma operação se as estatísticas estiverem desatualizadas. Se foi detectada uma baixa volumetria na última coleta de estatísticas para uma tabela, uma consulta submetida após aumento significativo

da volumetria sem atualização das estatísticas, induz o otimizador a escolher um plano com menor desempenho. (ROB; CORONEL, 2011)

2.2.3 Visões Materializadas

Ao definir uma visão, o SGBD armazena a consulta que a define. Entretanto, ao se definir uma visão materializada, não apenas sua consulta é armazenada como também seu conteúdo é calculado e armazenado. Apesar da redundância introduzida, é obtido um melhor desempenho ao se buscar os dados já calculados na visão materializada. (SILBERSCHATZ; KORTH; SUDARSHAN, 2006)

O problema ao se utilizar visões materializadas é a queda de desempenho nas operações de inclusão, alteração e exclusão de dados nas tabelas envolvidas. (SILBERSCHATZ; KORTH; SUDARSHAN, 2006)

2.2.4 Reescrita de Consulta

Como já foi citado anteriormente, dentre 70% a 80% dos problemas de desempenho do banco de dados são gerados pelo mau uso da linguagem SQL. O mau uso ocorre quando o desenvolvedor não tem o conhecimento ou a experiência necessária para escrever uma consulta da forma mais otimizada, impedindo o otimizador de alcançar o plano com o melhor desempenho possível. (MULLINS, 1998) (ROB; CORONEL, 2011) (SILBERSCHATZ; KORTH; SUDARSHAN, 2006)

Analisar e transformar uma consulta em outra equivalente, de forma automática ou manual, objetivando aumento de desempenho é denominado reescrita de consulta. (SILBERSCHATZ; KORTH; SUDARSHAN, 2006)

Entretanto, mesmo com o otimizador escolhendo o plano com menor custo, a escolha está limitada aos planos possíveis da consulta submetida. Para encontrar planos melhores é necessário realizar uma reestruturação ou reescrita da consulta SQL. (ROB; CORONEL, 2011)

Ao analisar uma consulta o otimizador do SGBD escolhe o plano de execução de menor custo apenas dentre os planos possíveis da consulta submetida. Por essa razão, formular consultas melhores ajuda o SGBD a obter os resultados com mais eficiência. (ROB; CORONEL, 2011)

Entretanto, formular consultas equivalentes melhores é uma tarefa difícil, que requer grande conhecimento da linguagem SQL, do funcionamento do SGBD e da estrutura do banco de dados utilizado. (ROB; CORONEL, 2011)

Reestruturar uma consulta possibilita o aumento de desempenho pois permite a utilização de novos algoritmos e recursos do SGBD pelo otimizador.

Outro fator a ser considerado é que reescrita de consultas SQL é uma tarefa custosa realizada apenas por uma ferramenta disponível para o SGBD Oracle. (CORADINI; CANTARELLI, 2012)

2.2.5 Estratégias de Reescrita

A reescrita de uma consulta SQL às vezes é a maneira mais eficiente de se aumentar o desempenho. (CYRAN, 1999) Esta sessão pretende abordar algumas das técnicas de reescrita utilizadas para aumentar a performance de uma consulta.

2.2.5.1 Especificar as colunas

Embora seja comum a utilização do (*) para determinar as colunas que serão retornadas, mesmo sem precisar de todas elas, é sempre melhor especificar explicitamente quais são as colunas de interesse. (HABIMANA, 2015)

2.2.5.2 Evitar uso incorreto do HAVING

A condição HAVING deve envolver funções de agregação. Caso contrário, esta condição deve ser movida para o WHERE. (HABIMANA, 2015)

2.2.5.3 Evitar DISTINCT desnecessário

Não deve ser utilizado DISTINCT em colunas que são PRIMARY KEY ou UNIQUE. (HABIMANA, 2015)

2.2.5.4 Expandir OR

Consultas que no WHERE possuem operações OR podem ser substituídas por UNION ALL de subconsultas individuais. É recomendado realizar esse tipo de transformação para permitir a utilização de índices ou para evitar produtos cartesianos. (ASHDOWN; COLGAN; KYTE, 2015) (HABIMANA, 2015)

Na figura 2.3, a consulta (a) as colunas **promo_id** e **prod_id** possuem índices associados a elas, porém a estrutura da consulta não permite a utilização dos mesmos. Uma alternativa reestruturada pode ser vista na consulta (b). O pedaço AND COALESCE (NOT (prod_id = 136), true) é necessário para que não ocorra duplicação de dados que podem ser selecionados por ambas subconsultas. (ASHDOWN; COLGAN; KYTE, 2015)

```
01.  SELECT *
02.  FROM    sales
03.  WHERE   promo_id = 33
04.         OR prod_id = 136
```

(a)

```
01.  (SELECT *
02.  FROM    sales
03.  WHERE   prod_id = 136)
04.  UNION ALL
05.  (SELECT *
06.  FROM    sales
07.  WHERE   promo_id = 33
08.  AND COALESCE (NOT ( prod_id = 136 ), true))
```

(b)

Figura 2.3: Comparação de consultas com expansão de OR.

2.2.5.5 Mesclar Visões

Visões são consultas SQL armazenadas no SGBD que possuem um nome de referência. Ao se utilizar uma visão, o nome de referência é substituído em tempo de execução pela consulta SQL armazenada. Entretanto, isto pode resultar em redundância de tabelas, impedir a utilização de índices e limitar os planos possíveis. Uma alternativa para resolver alguns casos é a realização de uma mescla manual da consulta da visão com a consulta principal, tomando cuidado para remover tabelas redundantes. (ASHDOWN; COLGAN; KYTE, 2015)

2.2.5.6 Introduzir Predicado

Existem casos onde visões não podem ser mescladas. Por exemplo, quando a visão usa UNION. Neste caso, após a substituição manual do nome de referência pela consulta SQL armazenada é necessário adicionar a expressão do WHERE individualmente em cada subconsulta. (ASHDOWN; COLGAN; KYTE, 2015)

2.2.5.7 Desassociar Subconsultas

Subconsultas no WHERE são de dois tipos: acopladas e não acopladas. (HABIMANA, 2015)

Quando a subconsulta não for acoplada, é possível transformar a subconsulta em um JOIN. Isto só pode ser feito quando for garantido que a consulta reescrita retorna exatamente os mesmos dados da consulta original. Por exemplo, um caso onde isto não pode ser feito, é quando na subconsulta existem funções de agrupamento. (ASHDOWN; COLGAN; KYTE, 2015)

Utilizando esta estratégia, na figura 2.4, a consulta (a) pode ser reescrita para consulta (b). (ASHDOWN; COLGAN; KYTE, 2015)

```

01.  SELECT *
02.  FROM    sales
03.  WHERE   cust_id IN (SELECT cust_id
04.                      FROM    customers)

                                (a)

01.  SELECT sales.*
02.  FROM    sales
03.          ,customers
04.  WHERE   sales.cust_id = customers.cust_id

                                (b)

```

Figura 2.4: Comparação de consultas com desassociar subconsultas.

2.2.5.8 Utilizar Visões Materializadas

Sempre que possível, é recomendado reescrever uma consulta para que utilize as visões materializadas existentes. (ASHDOWN; COLGAN; KYTE, 2015)

2.2.5.9 Transformação Estrela

O propósito da transformação estrela é evitar que o SGBD seja sobrecarregado analisando linhas não relevantes à consulta. A transformação estrela altera uma consulta onde a tabela do JOIN também possui condições no WHERE. O JOIN substituído por um WHERE com IN e subconsulta na consulta original. (ASHDOWN; COLGAN; KYTE, 2015)

Seguindo esta estratégia, na figura 2.5, a consulta (a), é reestruturada para a consulta (b).

```
01.  SELECT SUM(s.amount_sold) sales_amount
02.  FROM    sales s
03.          INNER JOIN channels ch
04.          ON s.channel_id = ch.channel_id
05.  WHERE   ch.channel_desc = 'Internet'
```

(a)

```
01.  SELECT SUM(s.amount_sold) sales_amount
02.  FROM    sales s
03.  WHERE   s.channel_id IN (SELECT channel_id
04.                          FROM    channels
05.                          WHERE   channel_desc = 'Internet')
```

(b)

Figura 2.5: Comparação de consultas com transformação estrela.

3 SISTEMAS EXISTENTES

Neste capítulo serão descritos sistemas existentes que analisam uma consulta SQL e sugerem recomendações para melhorar o desempenho. As informações foram obtidas a partir da documentação oficial disponível e diretamente do uso das ferramentas.

3.1 SQL Tuning Advisor (STA)

SQL Tuning Advisor é uma ferramenta destinada ao SGBD Oracle, incluso no pacote Oracle Database Tuning Pack, que realiza coleta de estatísticas, criação de índices e reescrita de consulta SQL. O pacote Oracle Database Tuning Pack é um pacote padrão do SGBD Oracle Database. (ASHDOWN; COLGAN; KYTE, 2015)

A sugestão de reescrita da SQL ocorre somente em casos extremos, pois o Oracle possui um processo de transformação interno que já realiza a reestruturação da consulta em casos onde é possível. Este processo é denominado *query transformation*. (ASHDOWN; COLGAN; KYTE, 2015)

Neste teste, foram utilizados o SGBD Oracle Database 12c - Versão 12.1.0 e a ferramenta de gerenciamento SQLDeveloper - Versão 4.1.3. É a ferramenta SQLDeveloper que disponibiliza acesso ao SQL Tuning Advisor. Para um usuário acessar o SQL Tuning Advisor, é necessário que possua a permissão de `ADVISOR`. O SQL Tuning Advisor pode ser executado na ferramenta SQLDeveloper ao clicar no botão específico ou através do atalho `Ctrl + F12`.

A figura 3.2 apresenta, na área destacada pelo retângulo, as melhorias sugeridas para a consulta apresentada na figura 3.1.

Nas descobertas “**A tabela "TUNING_ADVISOR"."DEPT" não foi analisada**” e “**A tabela "TUNING_ADVISOR"."EMP" não foi analisada**” a melhoria sugerida foi a de coletar estatísticas para as tabelas **dept** e **emp**. A ferramenta percebeu a falta de estatísticas, sinalizou a necessidade delas para melhorar os planos e indicou o comando a ser executado para esta tarefa.

Na descoberta “**Foi encontrada uma operação de produto cartesiano na linha de ID 2 do plano de execução**” a melhoria sugerida foi a de reestruturação da consulta. Como o processo de *query transformation* não conseguiu reestruturar de forma automática, apenas foi sinalizado a necessidade da reescrita sem sugerir como deve ser feito.

A figura 3.4 apresenta, na área destacada em vermelho, as melhorias sugeridas para a consulta apresentada na figura 3.3.

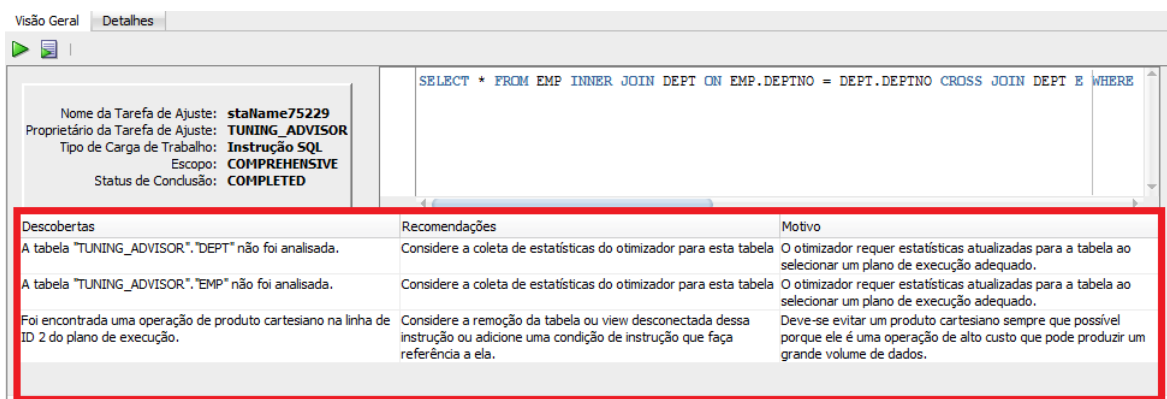
Assim como na figura 3.2, nas descobertas “**A tabela "TUNING_ADVISOR"."DEPT" não foi analisada**” e “**A tabela "TUNING_ADVISOR"."EMP" não foi analisada**” a melhoria sugerida foi a de coletar estatísticas para as tabelas **dept** e **emp**. A ferramenta percebeu a falta de estatísticas, sinalizou a necessidade delas para melhorar os planos e

```

01.  SELECT *
02.  FROM    emp
03.          INNER JOIN dept
04.          ON emp.deptno = dept.deptno
05.          CROSS JOIN dept e
06.  WHERE   (NVL(empno, 0) = 10)
07.          AND (dept.deptno = 10)
08.          AND (job LIKE 'MA_%'
09.              OR job LIKE 'CL_%'
10.              OR job = 'EAD')
11.  ORDER  BY emp.job
12.          ,emp.deptno

```

Figura 3.1: Primeira consulta para os casos de testes do SQL Tuning Advisor.



Descobertas	Recomendações	Motivo
A tabela "TUNING_ADVISOR", "DEPT" não foi analisada.	Considere a coleta de estatísticas do otimizador para esta tabela	O otimizador requer estatísticas atualizadas para a tabela ao selecionar um plano de execução adequado.
A tabela "TUNING_ADVISOR", "EMP" não foi analisada.	Considere a coleta de estatísticas do otimizador para esta tabela	O otimizador requer estatísticas atualizadas para a tabela ao selecionar um plano de execução adequado.
Foi encontrada uma operação de produto cartesiano na linha de ID 2 do plano de execução.	Considere a remoção da tabela ou view desconectada dessa instrução ou adicione uma condição de instrução que faça referência a ela.	Deve-se evitar um produto cartesiano sempre que possível porque ele é uma operação de alto custo que pode produzir um grande volume de dados.

Figura 3.2: Captura de tela dos resultados do SQL Tuning Advisor para a primeira consulta.

```

01.  SELECT *
02.  FROM    emp a
03.          INNER JOIN dept b
04.          ON b.deptno = a.deptno
05.  WHERE   empno = 10

```

Figura 3.3: Segunda consulta para os casos de testes do SQL Tuning Advisor.

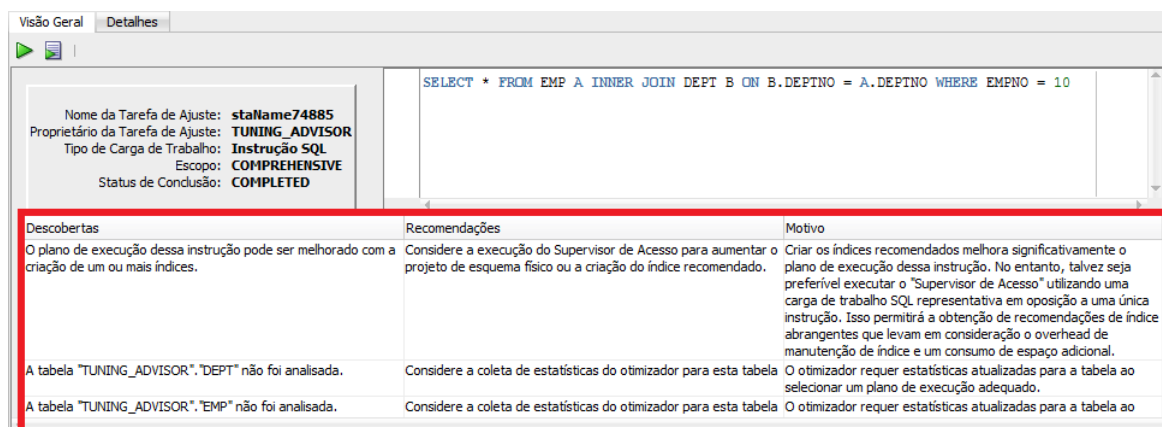


Figura 3.4: Captura de tela dos resultados do SQL Tuning Advisor para a segunda consulta.

indicou o comando a ser executado para esta tarefa.

Na descoberta **“O plano de execução dessa instrução pode ser melhorado com a criação de um ou mais índices”** a melhoria sugerida foi a de criação de índices. A ferramenta sinalizou a criação de um ou mais índices e indicou o comando a ser executado para esta tarefa.

3.2 Database Engine Tuning Advisor (DTA)

Database Engine Tuning Advisor é uma ferramenta destinada ao SGBD Microsoft SQL Server, capaz de sugerir criação de índices, criação de estatísticas e visões materializadas. A coleta de dados pode ser feita através de uma consulta SQL ou através da ferramenta SQL Server Profile. (SILBERSCHATZ; KORTH; SUDARSHAN, 2006)

Neste teste, foram utilizados o SGBD Microsoft SQL Server - Versão 12.0.4 e a ferramenta de gerenciamento SQL Server Management Studio - Versão 12.0.2 que disponibiliza acesso ao Database Engine Tuning Advisor.

O Database Engine Tuning Advisor pode ser executado na ferramenta SQL Server Management Studio ao clicar com o botão direito sobre a área de escrita da consulta e selecionar a opção específica.

A figura 3.6 apresenta, na área destacada em vermelho, as melhorias sugeridas para a consulta apresentada na figura 3.5. Como mostra a figura, foi estimado um aumento de desempenho de 96% com a aplicação das sugestões.

```

01.  SELECT *
02.  FROM    salesorderheader soh
03.          INNER JOIN salesorderdetail sod
04.              ON soh.salesorderid = sod.salesorderid
05.  WHERE   orderdate = '2005-12-19T00:00:00.000'

```

Figura 3.5: Consulta do caso de teste do Database Engine Tuning Advisor.

Nos objetos [dbo].[SalesOrderDetail] e [dbo].[SalesOrderHeader] as recomendações que iniciam com **_dta_index** são de sugestão de criação de índices. A ferramenta

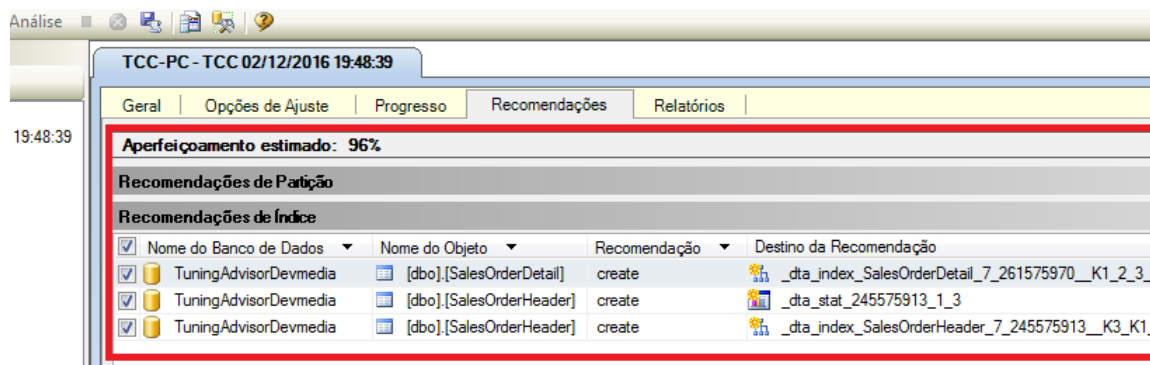


Figura 3.6: Captura de tela dos resultados do Database Engine Tuning Advisor para a consulta.

sinalizou a criação de um índice e indicou o comando a ser executado para esta tarefa.

No objeto **[dbo].[SalesOrderHeader]** a recomendação que inicia com **_dta_stat** é de criação de estatísticas para a tabela **salesorderheader**. A ferramenta percebeu a falta de estatísticas, sinalizou a necessidade delas para melhorar os planos e indicou o comando a ser executado para esta tarefa.

3.3 Postgres Advanced Server (PAS)

Postgres Advanced Server é uma ferramenta destinada ao SGBD PostgreSQL. Conta com diversas funcionalidades para o monitoramento e administração do SGBD, dentre elas possui a capacidade de analisar consultas SQL e sugerir a criação de índices. (EnterpriseDB Corporation, 2016)

Como esta ferramenta é paga, não foi possível a realização de testes para verificar seu funcionamento. Porém, de acordo com o manual a consulta é submetida ao utilitário **pg_advise_index** que analisa o desempenho da consulta, sugere a criação de índices indicando o comando a ser utilizado e estima o benefício pela utilização destes índices. (EnterpriseDB Corporation, 2016)

Mais informações podem ser encontradas em EnterpriseDB Corporation (2016).

3.4 Quadro Comparativo

A tabela 3.1 apresenta um quadro comparativo com as principais características das ferramentas analisadas.

Tabela 3.1: Ferramentas para aumento de desempenho de consultas SQL.

	STA	DTA	PAS
Criação de índices	Sim	Sim	Sim
Coleta de estatísticas	Sim	Sim	Não
Criação de visões materializadas	Não	Sim	Não
Reescrita da consulta	Sim	Não	Não

É importante ressaltar que apenas a ferramenta STA trabalha com a reescrita de consultas SQL.

4 O POSTGRESQL REWRITE ADVISOR (PGRA)

Para realizar a reescrita de consultas SQL de forma automatizada, foi desenvolvido um protótipo utilizando a linguagem Ruby chamado PostgreSQL Rewrite Advisor.

4.1 Diagrama de Casos de Uso

A figura 4.1 ilustra o diagrama de casos de uso do PGRA. Os atores envolvidos na utilização do protótipo são o usuário e o banco de dados. O usuário pode efetuar a manutenção de conexões de SGBD cadastradas, conectar em um SGBD já cadastrado, submeter consultas para otimização e visualizar os resultados após a otimização. O banco de dados apenas retorna metadados que serão utilizados pelo PGRA.

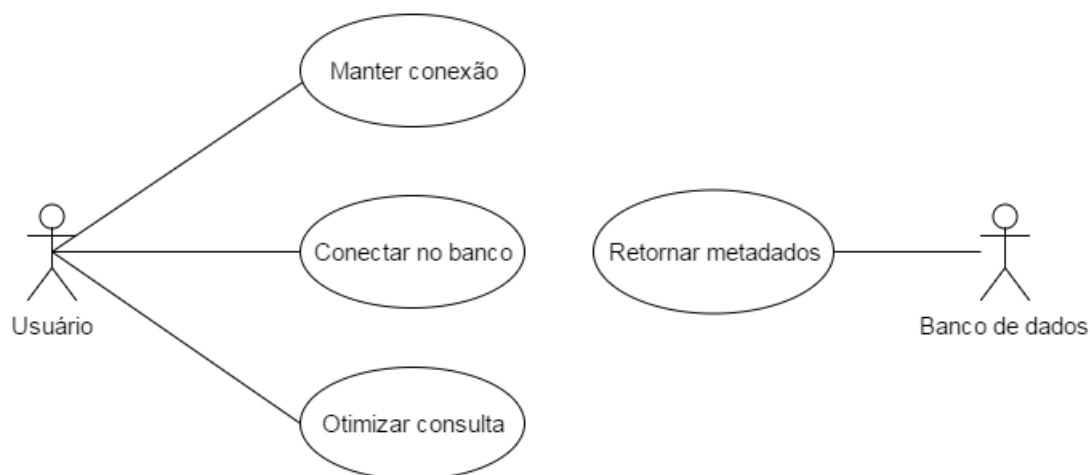


Figura 4.1: Diagrama de casos de Uso.

4.1.1 Usuário

Manter conexão nesta ação o ator pode efetuar a manutenção de conexões para o SGBD PostgreSQL. Para o cadastro de uma conexão é necessário informar nome, banco de dados, esquema, servidor, usuário e senha para acesso ao banco. Os dados da conexão só são armazenados se a conexão é válida. Após o cadastro é possível editar e remover uma conexão, desde que a conexão não esteja ativa. Ao alterar uma conexão esta deve continuar sendo válida.

Conectar no banco nesta ação o ator efetiva uma das conexões já cadastradas. Ao realizar isto, a opção de editar e remover a conexão fica indisponível. É possível alternar

entre as conexões cadastradas ou desconectar. A opção de submeter uma consulta para a otimização fica disponível.

Otimizar consulta nesta ação o ator submete uma consulta SQL para ser otimizada pelo PGRA. Isto só é possível com uma conexão ativa. Ao submeter uma consulta, primeiro é realizada uma análise que valida a consulta no banco de dados ativo. Caso negativo, o protótipo volta para a tela inicial e um erro é apresentado. Em seguida, é gerada uma árvore sintática com base na consulta submetida. Esta árvore passa por uma busca transversal para sintetização dos dados sobre a consulta. Com base nos dados sintetizados, são coletados metadados sobre as tabelas envolvidas. Depois, a consulta passa por um processo de identificação das possibilidades de otimização. Na sequência, é realizado um processo de reescrita para cada possibilidade identificada. Por fim são mostradas para o usuário a consulta original e as consultas geradas pelo PGRA. Junto com cada consulta gerada, é também apresentado o ganho percentual em relação à consulta original.

4.1.2 Banco de dados

Retornar metadados nesta ação o ator retornará os metadados solicitados pelo PGRA de uma consulta. Entre os metadados estão os dados resultantes da consulta, a quantidade de linhas retornadas, o EXPLAIN da consulta e informações sobre as tabelas envolvidas.

4.2 Diagrama de Classes

A figura 4.2 ilustra o diagrama de classes simplificado, sem atributos e métodos, referente ao protótipo criado. O protótipo possui as classes `Query`, `Column`, `Table`, `Database` e `PGRA`. A interface `Refactor` contém o contrato que as classes de otimização por reescrita devem obedecer. Atualmente, existem duas classes concretas de reescrita, `Refactor_col_sub_aco` e `Refactor_where_sub_aco`. O diagrama apresenta também as classes provenientes de terceiros e distribuídas como bibliotecas ou frameworks, `PgQuery`, `JSON`, `Base`, `Rule` e `Formatter`. As ligações entre as classes `Query`, `Table` e `Column` indicam associação por composição. A ligação entre as classes `Rule` e `Formatter` indicam associação por agregação. As demais ligações indicam apenas associação.

4.2.1 A Classe Query

A figura 4.3 apresenta a classe `Query` que é a classe contém a estrutura principal do protótipo. É uma classe concreta que tem visibilidade *public*. Todos os atributos são *private*, mas possuem métodos acessores para acesso externo a classes. Os atributos da classe `Query` são:

text é um atributo do tipo *String* que armazena o texto de uma consulta.

tree é um atributo do tipo *Hash* que armazena a árvore sintática de uma consulta. É obtida através do `PgQuery`.

explain é um atributo do tipo *Hash* que armazena o retorno do comando EXPLAIN. É obtido consultando o banco.

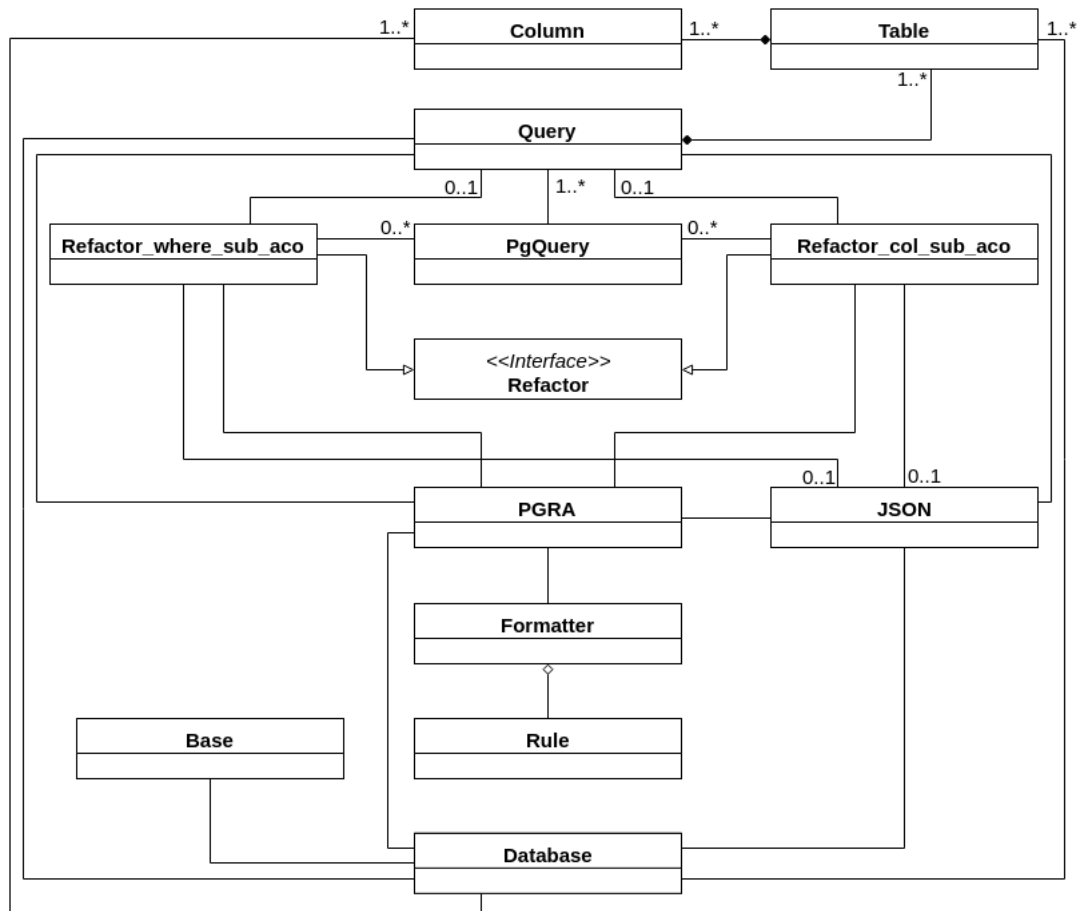


Figura 4.2: Diagrama de classes simplificado.

work é um atributo do tipo *boolean* que armazena se uma consulta foi executada com sucesso. É obtido consultando o banco.

cost é um atributo do tipo *float* que armazena o custo do comando EXPLAIN. É obtido através do atributo **explain**.

count é um atributo do tipo *int* que armazena a volumetria do resultado de uma consulta. É obtido consultando o banco.

type é um atributo do tipo *String* que armazena o tipo de comando SQL submetido. É obtido através do atributo **tree**.

tables é um atributo do tipo coleção de *Table* que armazena informações referentes às tabelas envolvidas em uma consulta. É obtido através do atributo **tree** e consultando o banco.

select é um atributo do tipo coleção de *Hash* que armazena a lista de campos que devem ser retornados. É obtido através do atributo **tree** e **tables**.

where é um atributo do tipo *Hash* que armazena a expressão WHERE de uma consulta. É obtido através do atributo **tree**.

subqueries é um atributo do tipo coleção de *Hash* que armazena as subconsultas de uma consulta. É obtido através do atributo **tree**. Cada subconsulta é tratada novamente

como um objeto `Query`.

Os métodos da classe `Query` são:

eval_text retorna o texto da consulta removendo espaços desnecessários.

eval_tree instancia o objeto `PgQuery` e retorna o valor do atributo **tree**.

eval_explain executa o comando `EXPLAIN` no banco e retorna seu resultado.

eval_work executa uma consulta no banco e retorna se foi bem-sucedido ou não.

eval_cost analisa o atributo **explain** e retorna o seu custo.

eval_count transforma uma consulta substituindo a lista de campos de retorno por `COUNT`, executa a consulta no banco e retorna a volumetria.

eval_type analisa o atributo **tree** e retorna o tipo de comando SQL que uma consulta executa.

eval_tables analisa no atributo **tree** as tabelas envolvidas em uma consulta, coleta informações sobre suas colunas no banco e retorna uma coleção de `Table`.

eval_select analisa nos atributos **tree** e **tables** as colunas que uma consulta utiliza e retorna metadados sobre cada coluna.

eval_where analisa no atributo **tree** o `WHERE` de uma consulta e retorna uma árvore sintática.

eval_subquerys analisa no atributo **tree** as subconsultas que uma consulta possui e retorna metadados sobre cada subconsulta. Cada subconsulta é recursivamente tratada como um objeto `Query`.

4.2.2 A Classe `Table`

A figura 4.4 apresenta a classe `Table` que possui associação de composição com a classe `Query`. É uma classe concreta que tem visibilidade *public*. Todos os atributos são *private*, mas possuem *getters* para acesso externo a classe. Os atributos da classe `Table` são:

name é um atributo do tipo `String` que armazena o nome de uma tabela.

alias é um atributo do tipo `String` que armazena o apelido de uma tabela.

columns é um atributo do tipo coleção de `Column` que armazena as colunas que uma tabela possui. É obtido através do método *static* **eval_columns** da classe `Column`.

Os métodos da classe `Table` são:

eval_name retorna o nome da tabela removendo espaços desnecessários.

eval_alias retorna o apelido da tabela removendo espaços desnecessários.

Query
+ text: String + tree: Hash + explain: Hash + work: boolean + cost: float + count: int + type: String + tables: Table[] + select: Hash[] + where: Hash + subquerys: Hash[]
+ eval_text(String): String + eval_tree(String): Hash + eval_explain(String): Hash + eval_work(String, boolean): boolean + eval_cost(Hash): float + eval_count(Hash): int + eval_type(Hash): String + eval_tables(Hash): Table[] + eval_select(Hash): Hash[] + eval_where(Hash): Hash + eval_subquerys(Hash): Hash[]

Figura 4.3: Diagrama da classe Query.

Table
+ name: String + alias: String + columns: Column[]
+ eval_name(String): String + eval_alias(String): String

Figura 4.4: Diagrama da classe Table.

4.2.3 A Classe Column

A figura 4.5 apresenta a classe `Column` que possui associação de composição com a classe `Table`. É uma classe concreta que tem visibilidade *public*. Todos os atributos são *private*, mas possuem *getters* para acesso externo a classe. Os atributos da classe `Column` são:

name é um atributo do tipo *String* que armazena o nome da coluna.

is_nullable é um atributo do tipo *String* que armazena **YES** se uma coluna pode ser nula ou **NO** caso contrário.

data_type é um atributo do tipo *String* que armazena o tipo de dado associado a coluna.

O método da classe `Column` é:

eval_columns é um método *static* que consulta uma tabela no banco e retorna metadados sobre cada coluna.

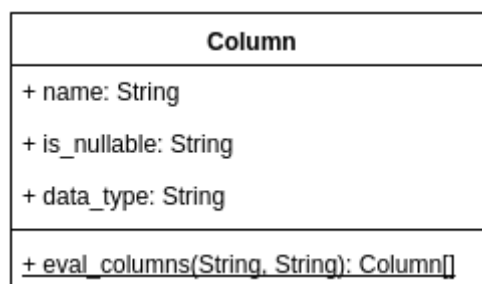


Figura 4.5: Diagrama da classe `Column`.

4.2.4 A Classe Database

A figura 4.6 apresenta a classe `Database` que tem métodos *static* utilitários para realizar a integração entre a aplicação e o banco de dados. É uma classe concreta *singleton* que tem visibilidade *public* e não possui atributos. Os métodos da classe `Database` são:

logout desconecta da conexão ativa.

get_connection_id retorna o identificador único da conexão ativa.

set_connection_id grava o identificador único da conexão ativa.

get_connection retorna as configurações de uma conexão cadastrada.

get_connections retorna as configurações de todas as conexões cadastradas.

eval_last_id cria e retorna um identificador único com base nos cadastrados.

query executa e retorna o resultado de um comando SQL na conexão ativa.

connect efetiva uma conexão.

valid_connect? retorna se uma conexão é válida.

connected? retorna se existe uma conexão ativa.

add cadastra uma conexão.

delete remove uma conexão cadastrada.

edit edita uma conexão cadastrada.

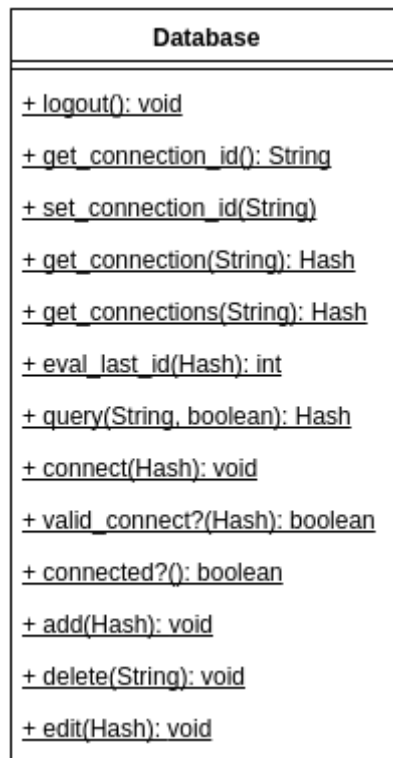


Figura 4.6: Diagrama da classe Database.

4.2.5 A Classe PGRA

A figura 4.7 apresenta a classe PGRA que tem métodos utilitários do protótipo PGRA. É uma classe concreta *singleton* que tem visibilidade *public* e não possui atributos. Todos os métodos são *static* com exceção do método **identify_any**, este é um método *private*. Os métodos da classe PGRA são:

save salva dados utilizados para comunicação intermodulos em formato *JSON*.

load retorna dados salvos pelo comando **save** em formato *Hash*.

add_error cadastra um erro que ocorreu, para ser mostrado posteriormente.

list_errors retorna a coleção de erros cadastrados.

clean_errors limpa a coleção de erros cadastrados.

path_exist? verifica se um *Hash* possui uma determinada *Key*.

identify executa de forma iterativa todos os métodos **identify** dos módulos de reescrita.

rewrite executa de forma iterativa os métodos **rewrite** dos módulos de reescrita se o **identify** correspondente retornou verdadeiro. Retorna uma coleção de *Query*.

get_query retorna do resultado do **rewrite** uma consulta SQL formatada.

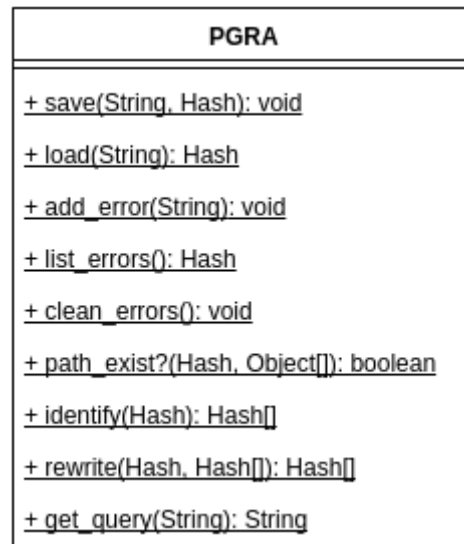


Figura 4.7: Diagrama da classe PGRA.

4.2.6 A Interface Refactor

A figura 4.8 apresenta a interface *Refactor* que possui o contrato que cada módulo de reescrita deve obedecer. É uma classe abstrata *singleton* que tem visibilidade *public*, possui métodos *static* e não possui atributos. Os métodos da interface *Refactor* são:

get_name retorna o nome do módulo de reescrita.

get_description retorna uma descrição sobre o módulo de reescrita.

identify retorna se uma consulta pode ser reescrita segundo o critério implementado pelo módulo.

rewrite retorna uma consulta reestruturada pelo módulo de reescrita.

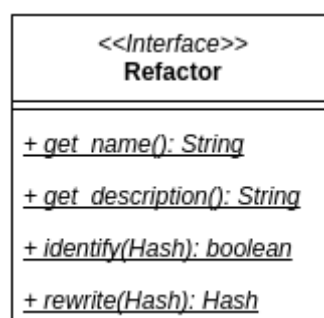


Figura 4.8: Diagrama da interface Refactor.

4.2.7 A Classe Refactor_col_sub_aco

A classe `Refactor_col_sub_aco` é uma classe concreta que especializa a interface `Refactor`. Tem o objetivo de reestruturar consultas que possuam uma subconsulta com condição de acoplamento na lista de retorno da consulta. A consulta apresentada na figura 4.9 é um exemplo deste caso. A lógica da implementação dos métodos **identify** e **rewrite** é apresentada nos pseudocódigos 4.10 e 4.11. Uma explicação detalhada do funcionamento do pseudocódigo será apresentada na subseção 5.2.1.

```

01.  SELECT coenti
02.      , (SELECT noenti
03.          FROM    ses_cias sc
04.          WHERE   sb.coenti = sc.coenti) noenti
05.      , valor
06.  FROM    ses_balanco sb

```

Figura 4.9: Consulta de exemplo para a classe `Refactor_col_sub_aco`.

```

1 Se existe subconsulta na consulta externa então
2   Para cada subconsulta faça
3     Se subconsulta está na lista de retorno da
      consulta externa então
4       Para cada elemento do WHERE da subconsulta faça
5         Se elemento for um campo da consulta externa
6           Retorna verdadeiro
7 Retorna falso

```

Figura 4.10: Pseudocódigo do `identify` da classe `Refactor_col_sub_aco`.

```

1 Para cada coluna do SELECT na consulta externa
2   Se a coluna é uma subconsulta e precisa ser reescrita
3     Adiciona a subconsulta com um apelido de tabela
      único ao FROM da consulta externa usando um LEFT JOIN
4     Para cada elemento do WHERE da subconsulta adicionada
5       Se o elemento pertence à subconsulta
6         Adicionar este elemento à lista de retorno
          da subconsulta
7     Move todas as condições do WHERE da subconsulta
      adicionada para o ON do LEFT JOIN criado
8     Ajusta apelidos das tabelas no ON do LEFT JOIN
9     Substitui a subconsulta na lista de retorno da consulta
      externa pelo seu resultado
10    Ajusta apelidos das tabelas na lista de retorno
11    Retorna a consulta reescrita

```

Figura 4.11: Pseudocódigo do `rewrite` da classe `Refactor_col_sub_aco`.

4.2.8 A Classe Refactor_where_sub_aco

A classe `Refactor_where_sub_aco` é uma classe concreta que especializa a interface `Refactor`. Tem o objetivo de reestruturar consultas que possuam uma subconsulta com condição de acoplamento no `WHERE` da consulta. A consulta apresentada na figura 4.12 é um exemplo deste caso. A lógica da implementação dos métodos **identify** e **rewrite** é apresentada nos pseudocódigos 4.13 e 4.14. Uma explicação detalhada do funcionamento do pseudocódigo será apresentada na subseção 5.2.2.

```

01.  SELECT coenti
02.      ,valor
03.  FROM    ses_balanco sb
04.  WHERE   sb.coenti = (SELECT sc.coenti
05.                        FROM    ses_cias sc
06.                        WHERE   sc.coenti = sb.coenti)

```

Figura 4.12: Consulta de exemplo para a classe `Refactor_where_sub_aco`.

```

1 Se existe subconsulta na consulta externa então
2   Para cada subconsulta faça
3     Se subconsulta está no WHERE da consulta
      externa então
4       Para cada elemento do WHERE da subconsulta faça
5         Se elemento for um campo da consulta externa
6           Retorna verdadeiro
7 Retorna falso

```

Figura 4.13: Pseudocódigo do `identify` da classe `Refactor_where_sub_aco`.

```

1 Para cada elemento do WHERE na consulta externa
2   Se o elemento é uma subconsulta e precisa ser reescrita
3     Adiciona a subconsulta com um apelido de tabela
      único ao FROM da consulta externa usando um INNER JOIN
4     Para cada elemento do WHERE da subconsulta adicionada
5       Se o elemento pertence à subconsulta
6         Adicionar este elemento à lista de retorno
          da subconsulta
7     Move todas as condições do WHERE da subconsulta
      adicionada para o ON do INNER JOIN criado
8     Ajusta apelidos das tabelas no ON do INNER JOIN
9     Substitui a subconsulta no WHERE da consulta
      externa pelo seu resultado
10    Ajusta apelidos das tabelas na lista de retorno
      e no WHERE
11    Retorna a consulta reescrita

```

Figura 4.14: Pseudocódigo do `rewrite` da classe `Refactor_where_sub_aco`.

4.2.9 A Classe PgQuery

A figura 4.15 apresenta a parte da classe `PgQuery` de terceiros utilizada no protótipo PGRA. É uma classe concreta que tem visibilidade *public*. Os métodos da classe `PgQuery` utilizados são:

parse é um método *static* que retorna um objeto do tipo **PgQuery**.

deparse é um método *public* que retorna uma consulta a partir da árvore sintática.

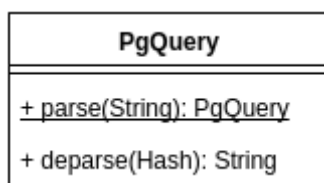


Figura 4.15: Diagrama da classe `PgQuery`.

4.2.10 A Classe JSON

A figura 4.16 apresenta a parte da classe `JSON` de terceiros utilizada no protótipo PGRA. É uma classe concreta que tem visibilidade *public*. Os métodos da classe `JSON` utilizados são:

parse é um método *static* que retorna um *Hash* a partir de um *JSON*.

generate é um método *static* que retorna um *JSON* a partir de um *Hash*.

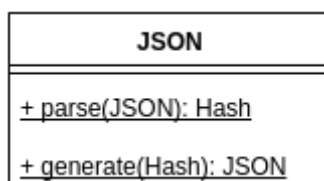


Figura 4.16: Diagrama da classe `JSON`.

4.2.11 A Classe Base

A figura 4.17 apresenta a parte da classe `Base` de terceiros utilizada no protótipo PGRA. É uma classe concreta que tem visibilidade *public*. O método da classe `Base` utilizado é:

connection é um método *static* que retorna uma instancia da classe `PostgreSQLAdapter`.

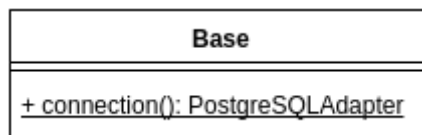


Figura 4.17: Diagrama da classe Base.

4.2.12 A Classe Rule

A figura 4.18 apresenta a parte da classe `Rule` de terceiros utilizada no protótipo PGRA. É uma classe concreta que tem visibilidade *public*. Os atributos da classe `Rule` alterados são:

keyword é um atributo do tipo *int* que define a formatação dos comandos SQL na consulta. Estes podem variar entre maiúsculos, minúsculos e não alterar.

function_names é um atributo do tipo coleção de *String* que define quais funções SQL serão reconhecidas pelo formatador.

indent_string é um atributo do tipo *String* que define a quantidade de espaços na tabulação.

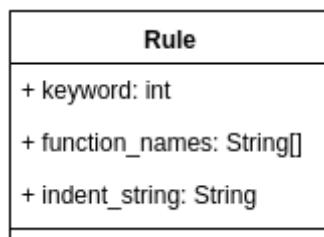


Figura 4.18: Diagrama da classe Rule.

4.2.13 A Classe Formatter

A figura 4.19 apresenta a parte da classe `Formatter` de terceiros utilizada no protótipo PGRA. Possui associação de agregação com a classe `Rule`. É uma classe concreta que tem visibilidade *public*. Necessita de um objeto `Rule`, que define as regras a serem seguidas na formatação, para sua criação. O método da classe `Formatter` utilizado é:

format é um método *public* que retorna o texto de uma consulta formatado.

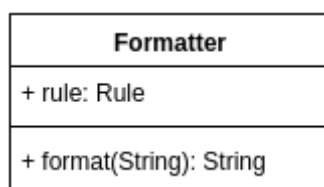


Figura 4.19: Diagrama da classe Formatter.

4.3 Fluxograma de Funcionamento

A figura 4.20 ilustra o fluxograma de funcionamento do protótipo PGRA. As setas representam o fluxo de execução, os quadrados com cantos arredondados representam as tarefas e o losango uma decisão. Os terminadores possuem seus próprios símbolos. Nesta figura, o primeiro passo para a utilização do protótipo é efetivar uma conexão cadastrada. Após, o protótipo recebe uma consulta, que é validada no SGBD antes de prosseguir. Em seguida é gerada uma árvore sintática da consulta, utilizando o **pg_query**, e realizado o acesso ao bando de dados para calcular a volumetria, obter o **EXPLAIN** e os metadados das tabelas envolvidas. Todas estas informações serão utilizadas pelos métodos de reescrita. Os módulos de reescrita devem estar em uma pasta específica para ser reconhecidos e utilizados pelo PGRA. Depois, são executados, de forma independente, os métodos de identificação de cada módulo de reescrita encontrado. Na sequência, são executados, de forma independente, os métodos de reescrita caso a respectiva identificação tenha sido positiva. Por fim, todas as formas reescritas são apresentadas ao usuário.

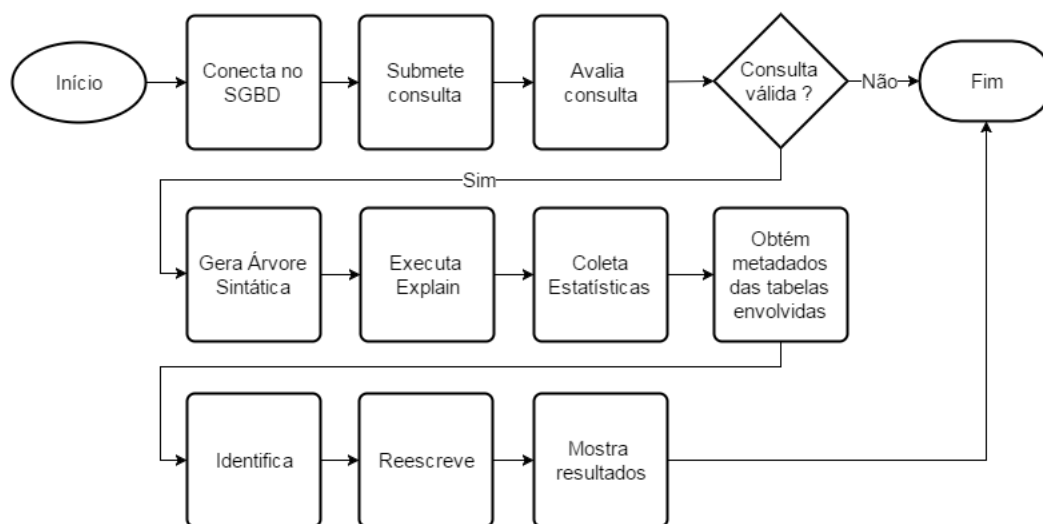


Figura 4.20: Fluxograma de funcionamento.

4.4 Ferramentas Utilizadas

4.4.1 PostgreSQL

O PostgreSQL é um SGBD objeto-relacional *open source*. Possui mais de 15 anos de desenvolvimento e suporta os principais sistemas operacionais tais como GNU/Linux, Unix e MS Windows. Além disso, possui diversas *features*, que incluem compatibilidade ACID, transações, triggers, procedures e interfaces nativas de programação com diversas linguagens. Por ser um banco de grande porte *open source* diversas companhias adotaram o uso dele, dentre as categorias pode-se destacar e-commerce, educação, jogos entre outras. (The PostgreSQL Global Development Group, 2016b)

4.4.2 Ruby

Linguagem de programação lançada em 1995 por Yukihiro "Matz" Matsumoto. Equilibra programação funcional em conjunto de programação imperativa para o seu desenvolvimento. Ruby é uma linguagem interpretada, orientada a objetos, dinamicamente tipada

e que possui uma licença BDS, a qual garante direitos de utilização, cópia, modificação e distribuição, livre de custos. A versão estável atual é a 2.3.1. (Ruby Community, 2016)

4.4.3 Sinatra

Sinatra é uma *Domain-Specific Language* (DSL) para a criação rápida de aplicações *WEB*, desenvolvidas em Ruby com o mínimo esforço. Foi desenvolvida em 2007 por Blake Mizerany. Sinatra não é um *framework*, pois não impõe nenhuma estrutura a ser seguida. Ele apenas auxilia aplicações Ruby, expondo elas na web, sendo possível acessá-las através de requisições *HTTP*. (JONES, 2013)

4.4.4 Simple Sinatra MVC Template

É um *framework* MVC desenvolvido por Kath Giron Pe, disponível para *download* via GitHub, o modelo inclui diversas extensões Ruby a qual auxiliam na criação do projeto e possui uma estrutura predefinida de organização. (PE, 2016)

4.4.5 pg_query

Extensão Ruby, com a função de realizar *parser* de consultas SQL destinadas ao SGBD PostgreSQL e retornar uma árvore sintática. Esta extensão foi desenvolvida por Lukas Fittl e se encontra atualmente na versão 0.11.2. (FITTL, 2015) Por falta de detalhes na documentação foi realizado testes para verificar o retorno da extensão. Utilizando a consulta apresentada na figura 4.21 foi gerada a seguinte árvore sintática. Por esta ser muito extensa, este é um exemplo simplificado com as principais informações.

```
01.  SELECT *
02.  FROM  tabela1 t1
03.  WHERE t1.campo1 = 'PGRA'
04.        AND t1.campo2 > 10
```

Figura 4.21: Primeira consulta de exemplo do pg_query.

```
[{"SelectStmt":{"targetList":[{"ResTarget":{"val":{"ColumnRef":{"fields":["A_Star"]}}}]}],
"fromClause":[{"RangeVar":{"relname":"tabela1",
"alias":{"Alias":{"aliasname":"t1"}}}]}],
"whereClause":{"BoolExpr":{"boolop":0,
```

```

"args":[
  {"A_Expr":{
    "name":[{"String":{"str":"="}}],
    "lexpr":{"ColumnRef":{"fields":[
      {"String":{"str":"t1"}},
      {"String":{"str":"campo1"}}
    ]}},
    "rexpr":{"A_Const":{"val":{"
      "String":{"str":"PGRA"}
    }}}
  }},
  {"A_Expr":{
    "name":[{"String":{"str":">"}},
    "lexpr":{"ColumnRef":{"fields":[
      {"String":{"str":"t1"}},
      {"String":{"str":"campo2"}}
    ]}},
    "rexpr":{"A_Const":{"val":{"
      "Integer":{"ival":10}
    }}}
  }}
]
}
}
}]

```

Na realização dos testes foi verificado que a extensão utiliza recursividade para representar subconsultas. Utilizando a consulta apresentada na figura 4.22 foi gerada a seguinte árvore sintática. Por esta ser muito extensa, este é um exemplo simplificado com as principais informações.

```

01.  SELECT *
02.  FROM  tabela1 t1
03.  WHERE t1.campo1 IN (SELECT t2.campo1
04.                      FROM  tabela2 t2)

```

Figura 4.22: Segunda consulta de exemplo do pg_query.

```

[{"SelectStmt":{
  "targetList":[
    {"ResTarget":{"val":{"ColumnRef":{"fields":[
      {"A_Star":{}}
    ]}}}}
  ],
  "fromClause":[

```


5 RESULTADOS

Neste capítulo serão apresentados os resultados obtidos.

5.1 O Protótipo

Neste trabalho foi implementado em Ruby um protótipo de ferramenta para melhoria de desempenho de consultas para o SGBD PostgreSQL.

A figura 5.1 mostra a tela inicial do protótipo, sem conexões cadastradas. No canto inferior direito está o botão para realizar o cadastro de conexões.

Ao clicar no botão para realizar cadastro de conexões da tela inicial, é apresentado o formulário para o cadastro, conforme a figura 5.2. São informados, obrigatoriamente: nome, banco de dados, esquema, servidor, usuário e senha para acesso ao banco. É possível verificar se a conexão é válida através do botão **TESTAR CONEXÃO**. O botão **ADICIONAR** cadastra a conexão caso seja válida e retorna para a tela inicial. A figura 5.3 mostra a nova tela inicial, mostrando a conexão PGRA adicionada.

Na tela inicial, é possível efetuar a manutenção de uma conexão cadastrada. A figura 5.4 mostra os dados da conexão PGRA e os botões para alterar e excluir. A alteração utiliza o mesmo formulário de cadastro de conexão. A conexão deve continuar válida.

Na tela inicial, ao clicar no botão de conectar de uma conexão, é mostrada a tela 5.6. Nesta tela é possível submeter uma consulta para análise. Na figura, foi utilizada a consulta apresentada na figura 5.5. Ao clicar no botão **ANALISAR** é exibida ao usuário a tela de progresso, mostrando os passos a medida que vão sendo realizados. A figura 5.7 apresenta um exemplo de tela de progresso.

Concluído o processo de análise, é apresentada a tela de resultados. A figura 5.8 apresenta o resultado para a consulta apresentada na figura 5.5. Na tela de resultados, é exibida uma tabela na qual as linhas são consultas equivalentes ordenadas pelo seu desempenho. Para cada consulta é apresentado qual reescrita foi aplicada, o custo obtido através do EXPLAIN, o ganho percentual de desempenho em relação à consulta original, um botão para visualizar o comando SQL reescrito e um botão para continuar a análise a partir desta consulta.

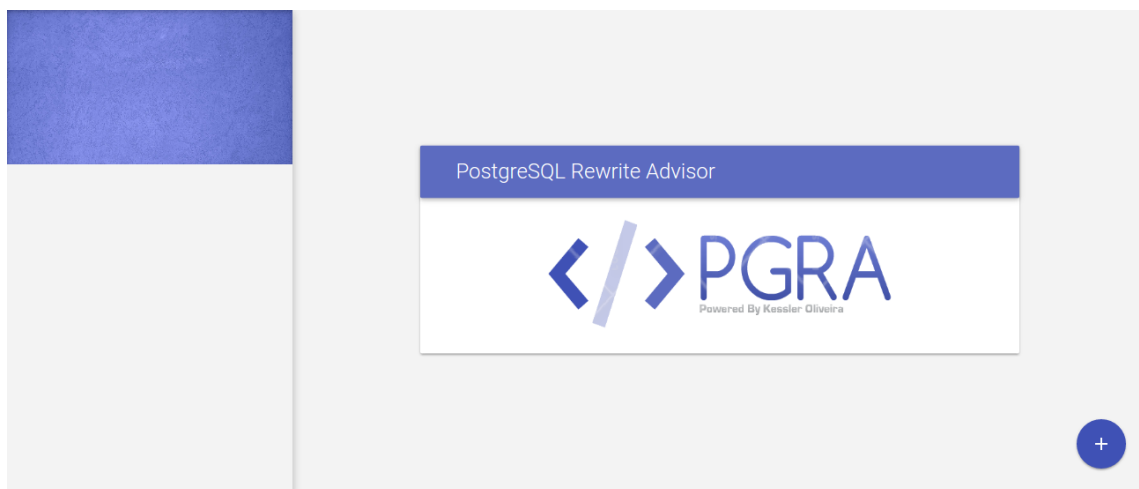


Figura 5.1: Tela inicial do protótipo sem conexões cadastrada.

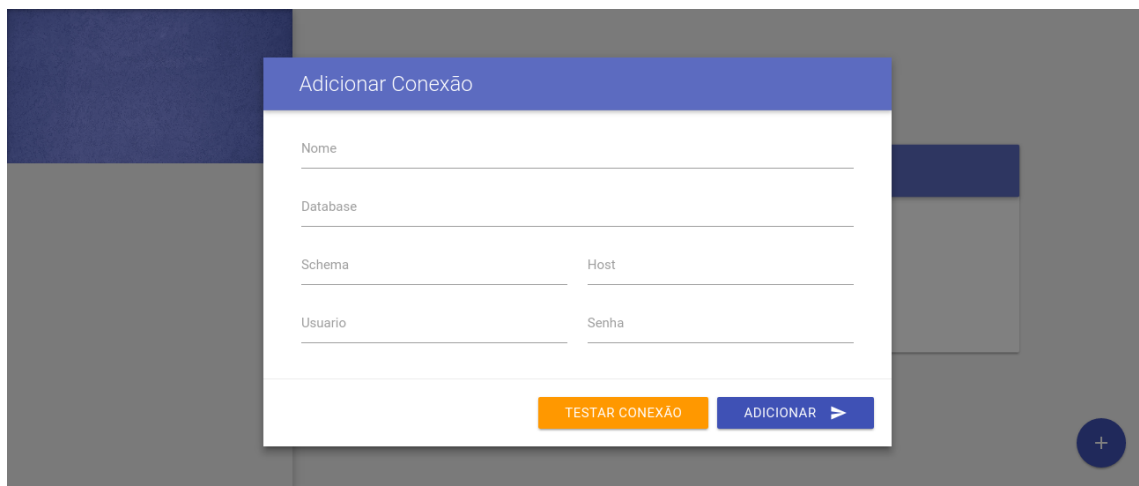


Figura 5.2: Tela de cadastro de conexão do protótipo.

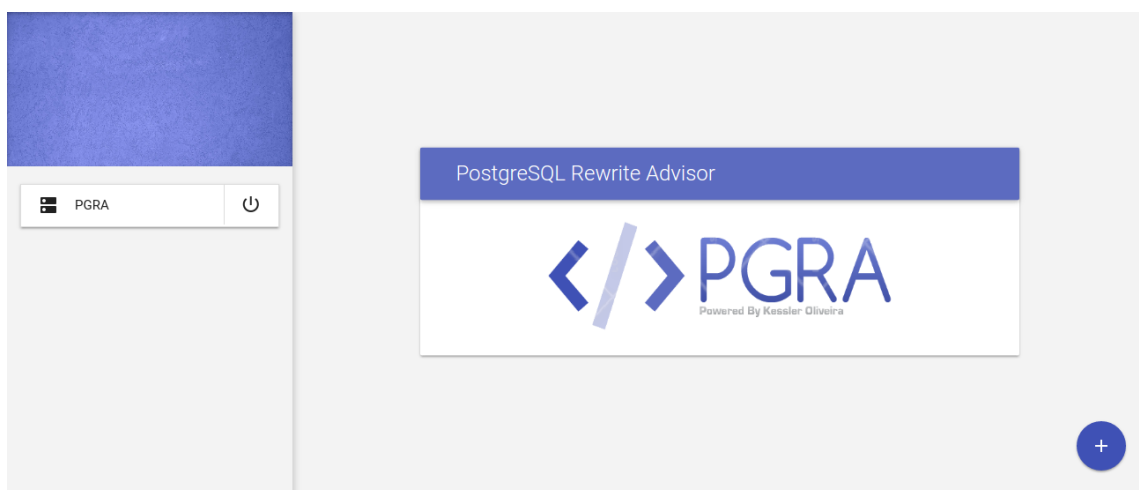


Figura 5.3: Tela inicial do protótipo com conexões cadastrada.

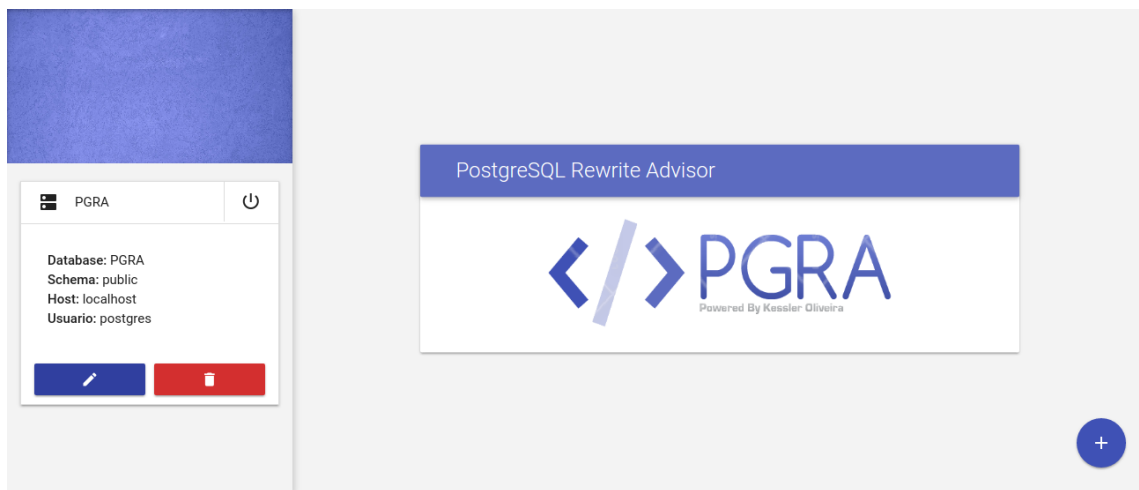


Figura 5.4: Visualizar uma conexão cadastrada no protótipo.

```

01.  SELECT (SELECT y.id
02.          FROM teste2 y
03.          WHERE x.id = y.id)
04.  FROM teste x

```

Figura 5.5: Consulta de exemplo do protótipo PGRA.

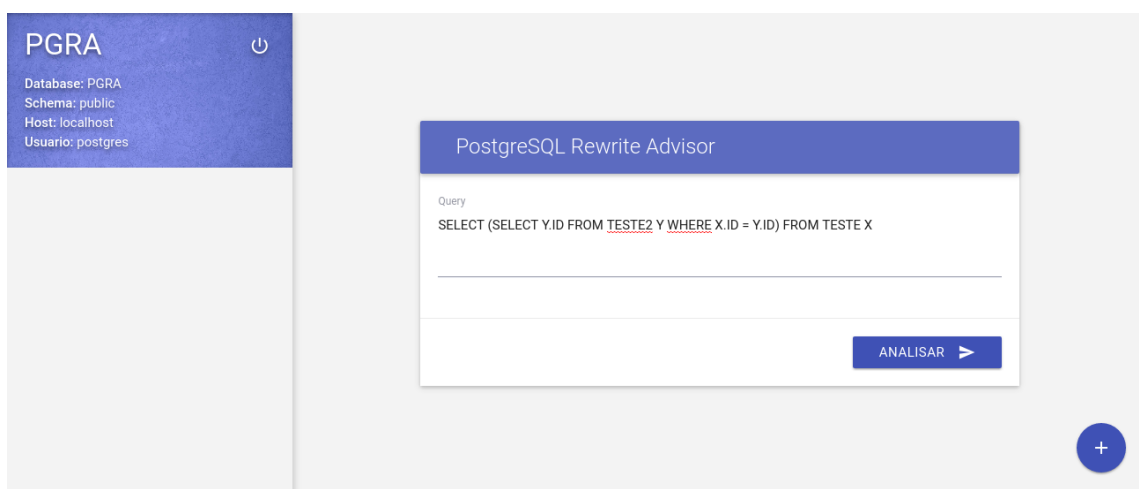


Figura 5.6: Tela para submeter uma consulta do protótipo.

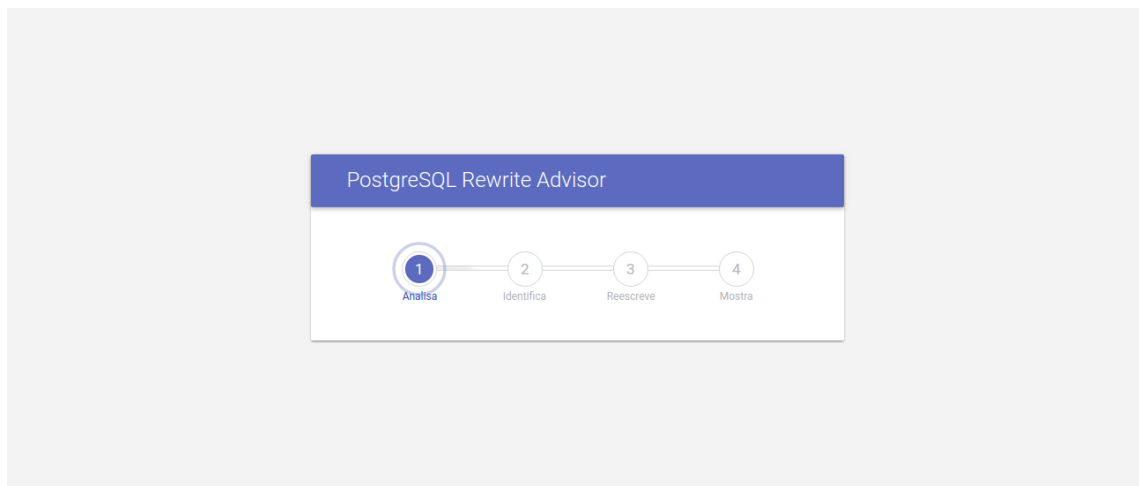


Figura 5.7: Tela de progresso de uma consulta do protótipo.

PGRA
 Database: PGRA
 Schema: public
 Host: localhost
 Usuario: postgres

PostgreSQL Rewrite Advisor

Rank	Reescrita	Descrição	Custo	Aumento de Performance	Query	Reescrever Novamente
1	Subconsulta acoplada na lista de parâmetros	Refatora subconsultas acopladas que estejam na lista de parâmetros de retorno, apenas para o primeiro nível.	201.88	99.44%		
2	Original	-	35735.5	0%		

INICIO

+

Figura 5.8: Tela de resultados de uma consulta do protótipo.

5.2 Casos de Testes

Nesta seção serão apresentados os resultados dos testes realizados.

O banco de dados utilizado foi derivado da base de dados do Sistema de estatísticas da SUSEP. O diagrama relacional do banco utilizado é apresentado na figura 5.9. A quantidade de linhas em cada tabela é apresentada na tabela 5.1.

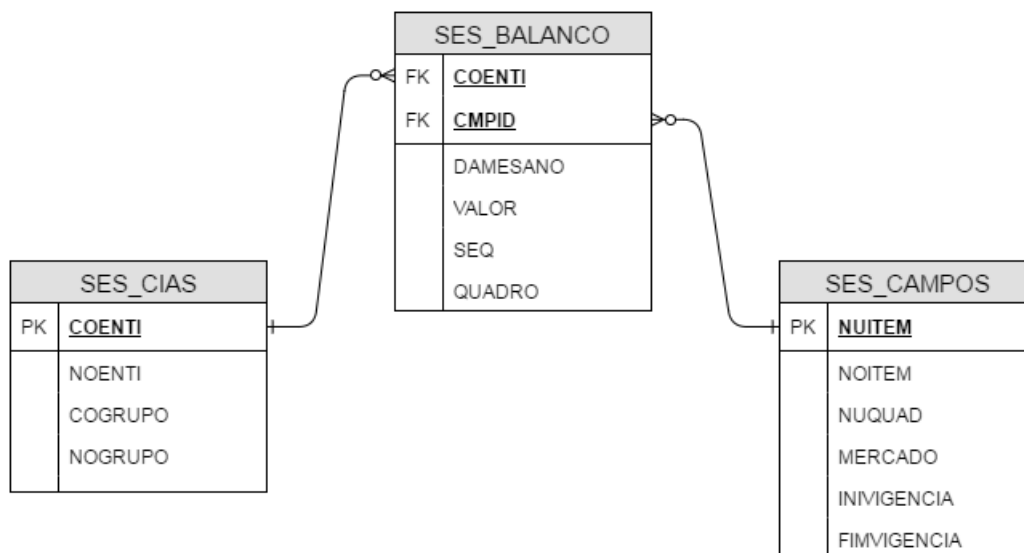


Figura 5.9: Diagrama Relacional.

Tabela 5.1: Volumetria para cada tabela.

Tabela	Linhas
SES_BALANCO	999.978
SES_CAMPOS	1.895
SES_CIAS	615

5.2.1 Primeiro caso de teste

O primeiro caso de teste apresenta o padrão de consulta SQL, básico, que deve ser reconhecido e reescrito pelo módulo da classe `Refactor_col_sub_aco`. Ao submeter a consulta apresentada na figura 5.10 ocorrem as seguintes reestruturações após executar cada linha, conforme o pseudocódigo da figura 4.11.

Ao executar a linha 3 a consulta é transformada conforme a figura 5.11.

Ao executar a linha 6 a consulta é transformada conforme a figura 5.12.

Ao executar a linha 7 a consulta é transformada conforme a figura 5.13.

Ao executar a linha 8 a consulta é transformada conforme a figura 5.14.

Ao executar a linha 9 a consulta é transformada conforme a figura 5.15.

Ao executar a linha 10 a consulta é transformada conforme a figura 5.16.

O resultado gerado pelo primeiro caso de reescrita é apresentado na figura 5.17.

```

01.  SELECT coenti
02.      ,(SELECT noenti
03.          FROM    ses_cias sc
04.          WHERE   sb.coenti = sc.coenti) noenti
05.      ,valor
06.  FROM    ses_balanco sb

```

Figura 5.10: Consulta submetida do primeiro caso de teste.

```

01.  SELECT coenti
02.      ,(SELECT noenti
03.          FROM    ses_cias sc
04.          WHERE   sb.coenti = sc.coenti) noenti
05.      ,valor
06.  FROM    ses_balanco sb
07.      LEFT JOIN (SELECT noenti
08.                  FROM    ses_cias sc
09.                  WHERE   sb.coenti = sc.coenti) rw_tmp_1

```

Figura 5.11: Primeiro caso de teste ao executar a linha 3 do pseudocódigo.

```

01.  SELECT coenti
02.      ,(SELECT noenti
03.          FROM    ses_cias sc
04.          WHERE   sb.coenti = sc.coenti) noenti
05.      ,valor
06.  FROM    ses_balanco sb
07.      LEFT JOIN (SELECT noenti
08.                  ,sc.coenti rw_coenti
09.                  FROM    ses_cias sc
10.                  WHERE   sb.coenti = sc.coenti) rw_tmp_1

```

Figura 5.12: Primeiro caso de teste ao executar a linha 6 do pseudocódigo.

```

01.  SELECT coenti
02.      ,(SELECT noenti
03.          FROM    ses_cias sc
04.          WHERE   sb.coenti = sc.coenti) noenti
05.      ,valor
06.  FROM    ses_balanco sb
07.      LEFT JOIN (SELECT noenti
08.                  ,sc.coenti rw_coenti
09.                  FROM    ses_cias sc) rw_tmp_1
10.          ON sb.coenti = sc.coenti

```

Figura 5.13: Primeiro caso de teste ao executar a linha 7 do pseudocódigo.

```

01.  SELECT coenti
02.      ,(SELECT noenti
03.          FROM    ses_cias sc
04.          WHERE   sb.coenti = sc.coenti) noenti
05.      ,valor
06.  FROM    ses_balanco sb
07.      LEFT JOIN (SELECT noenti
08.                  ,sc.coenti rw_coenti
09.                  FROM    ses_cias sc) rw_tmp_1
10.          ON sb.coenti = rw_tmp_1.rw_coenti

```

Figura 5.14: Primeiro caso de teste ao executar a linha 8 do pseudocódigo.

```

01.  SELECT coenti
02.      ,noenti noenti
03.      ,valor
04.  FROM    ses_balanco sb
05.      LEFT JOIN (SELECT noenti
06.                  ,sc.coenti rw_coenti
07.                  FROM    ses_cias sc) rw_tmp_1
08.          ON sb.coenti = rw_tmp_1.rw_coenti

```

Figura 5.15: Primeiro caso de teste ao executar a linha 9 do pseudocódigo.


```

01.  SELECT coenti
02.      ,rw_tmp_1.noenti noenti
03.      ,valor
04.  FROM    ses_balanco sb
05.      LEFT JOIN (SELECT noenti
06.                  ,sc.coenti rw_coenti
07.                  FROM    ses_cias sc) rw_tmp_1
08.      ON sb.coenti = rw_tmp_1.rw_coenti

```

Figura 5.16: Primeiro caso de teste ao executar a linha 10 do pseudocódigo.

The screenshot shows the PostgreSQL Rewrite Advisor interface. On the left, a sidebar displays 'PGRA' with database details: Database: PGRA, Schema: public, Host: localhost, and Usuario: postgres. The main panel is titled 'PostgreSQL Rewrite Advisor' and contains a table with the following data:

Rank	Reescrita	Descrição	Custo	Aumento de Performance	Query	Reescrever Novamente
1	Subconsulta acoplada na lista de parâmetros	Refatora subconsultas acopladas que estejam na lista de parâmetros de retorno, apenas para o primeiro nível.	31418.24	99.88%		
2	Original	-	26717938.8	0%		

At the bottom right of the table, there is a blue button labeled 'INICIO' with a circular arrow icon next to it. A blue circular button with a white plus sign is located at the bottom right of the interface.

Figura 5.17: Tela de resultados do primeiro caso de teste.

5.2.2 Segundo caso de teste

O segundo caso de teste apresenta o padrão de consulta SQL, básico, que deve ser reconhecido e reescrito pelo módulo da classe `Refactor_where_sub_aco`. Ao submeter a consulta apresentada na figura 5.18 ocorrem as seguintes reestruturações após executar cada linha, conforme o pseudocódigo da figura 4.14.

Ao executar a linha 3 a consulta é transformada conforme a figura 5.19.

Ao executar a linha 6 a consulta é transformada conforme a figura 5.20.

Ao executar a linha 7 a consulta é transformada conforme a figura 5.21.

Ao executar a linha 8 a consulta é transformada conforme a figura 5.22.

Ao executar a linha 9 a consulta é transformada conforme a figura 5.23.

Ao executar a linha 10 a consulta é transformada conforme a figura 5.24.

O resultado gerado pelo segundo caso de reescrita é apresentado na figura 5.25.

```

01.  SELECT coenti,
02.      valor
03.  FROM    ses_balanco sb
04.  WHERE   sb.coenti = (SELECT sc.coenti
05.                      FROM    ses_cias sc
06.                      WHERE   sc.coenti = sb.coenti)

```

Figura 5.18: Consulta submetida do segundo caso de teste.

```

01.  SELECT coenti,
02.      valor
03.  FROM    ses_balanco sb
04.      JOIN (SELECT sc.coenti
05.            FROM    ses_cias sc
06.            WHERE   sc.coenti = sb.coenti) rw_tmp_1
07.  WHERE   sb.coenti = (SELECT sc.coenti
08.                      FROM    ses_cias sc
09.                      WHERE   sc.coenti = sb.coenti)

```

Figura 5.19: Segundo caso de teste ao executar a linha 3 do pseudocódigo.

```

01.  SELECT coenti,
02.      valor
03.  FROM  ses_balanco sb
04.      JOIN (SELECT sc.coenti,
05.                  sc.coenti rw_coenti
06.                  FROM  ses_cias sc
07.                  WHERE  sc.coenti = sb.coenti) rw_tmp_1
08.  WHERE sb.coenti = (SELECT sc.coenti
09.                    FROM  ses_cias sc
10.                    WHERE  sc.coenti = sb.coenti)

```

Figura 5.20: Segundo caso de teste ao executar a linha 6 do pseudocódigo.

```

01.  SELECT coenti,
02.      valor
03.  FROM  ses_balanco sb
04.      JOIN (SELECT sc.coenti,
05.                  sc.coenti rw_coenti
06.                  FROM  ses_cias sc) rw_tmp_1
07.      ON sc.coenti = sb.coenti
08.  WHERE sb.coenti = (SELECT sc.coenti
09.                    FROM  ses_cias sc
10.                    WHERE  sc.coenti = sb.coenti)

```

Figura 5.21: Segundo caso de teste ao executar a linha 7 do pseudocódigo.

```

01.  SELECT coenti,
02.      valor
03.  FROM  ses_balanco sb
04.      JOIN (SELECT sc.coenti,
05.                  sc.coenti rw_coenti
06.                  FROM  ses_cias sc) rw_tmp_1
07.      ON rw_tmp_1.rw_coenti = sb.coenti
08.  WHERE sb.coenti = (SELECT sc.coenti
09.                    FROM  ses_cias sc
10.                    WHERE  sc.coenti = sb.coenti)

```

Figura 5.22: Segundo caso de teste ao executar a linha 8 do pseudocódigo.

```

01.  SELECT coenti,
02.      valor
03.  FROM    ses_balanco sb
04.      JOIN (SELECT sc.coenti,
05.                  sc.coenti rw_coenti
06.                  FROM    ses_cias sc) rw_tmp_1
07.      ON rw_tmp_1.rw_coenti = sb.coenti
08.  WHERE  sb.coenti = sc.coenti

```

Figura 5.23: Segundo caso de teste ao executar a linha 9 do pseudocódigo.

```

01.  SELECT sb.coenti,
02.      sb.valor
03.  FROM    ses_balanco sb
04.      JOIN (SELECT sc.coenti,
05.                  sc.coenti rw_coenti
06.                  FROM    ses_cias sc) rw_tmp_1
07.      ON rw_tmp_1.rw_coenti = sb.coenti
08.  WHERE  sb.coenti = rw_tmp_1.coenti

```

Figura 5.24: Segundo caso de teste ao executar a linha 10 do pseudocódigo.

PGRA

Database: PGRA
Schema: public
Host: localhost
Usuário: postgres

PostgreSQL Rewrite Advisor

Rank	Reescrita	Descrição	Custo	Aumento de Performance	Query	Reescrever Novamente
1	Subconsulta acoplada na cláusula WHERE	Refatora subconsultas acopladas que estejam nas cláusula WHERE, apenas para o primeiro nível.	31418.24	99.88%		
2	Original	-	26720440.0	0%		

INICIO ↻

+

Figura 5.25: Tela de resultados do segundo caso de teste.

5.2.3 Terceiro caso de teste

O terceiro caso de teste apresenta o padrão de consulta SQL, complexo, que deve ser reconhecido e reescrito pelo módulo da classe `Refactor_col_sub_aco`. Ao submeter a consulta (a) apresentada na figura 5.26 o protótipo realiza a reestruturação, gerando a consulta (b). O resultado gerado pelo terceiro caso de reescrita é apresentado na figura 5.27.

5.2.4 Quarto caso de teste

O quarto caso de teste apresenta o padrão de consulta SQL, complexo, que deve ser reconhecido e reescrito pelo módulo da classe `Refactor_where_sub_aco`. Ao submeter a consulta (a) apresentada na figura 5.28 o protótipo realiza a reestruturação, gerando a consulta (b). O resultado gerado pelo quarto caso de reescrita é apresentado na figura 5.29.

5.2.5 Quinto caso de teste

O quinto caso de teste apresenta o padrão de consulta SQL, composto, que deve ser reconhecido e reescrito pelos módulos das classes `Refactor_col_sub_aco` e `Refactor_where_sub_aco`. Ao submeter a consulta da figura 5.30 o protótipo realiza a reestruturação, gerando as consultas (a) e (b) apresentadas na figura 5.31.

Porém, ainda é possível realizar reescrita, como pode ser visto na figura 5.32, sendo assim foi efetuado a reescrita novamente para a primeira consulta do ranque, resultado na consulta apresentada na figura 5.33. O resultado final gerado pelo quinto caso de reescrita é apresentado na figura 5.34.

5.2.6 Sexto caso de teste

O sexto caso de teste apresenta o padrão de consulta SQL, porem este se encontra no segundo nível, impossibilitando a identificação de reescrita. Ao submeter a consulta da figura 5.35 o protótipo não consegue identificar o padrão, retornando apenas a consulta original. O resultado gerado pelo sexto caso de reescrita é apresentado na figura 5.36.

```

01.  SELECT coenti
02.      , (SELECT noenti
03.          FROM   ses_cias sc
04.          WHERE  sb.coenti = sc.coenti) noenti
05.      , (SELECT cogrupa
06.          FROM   ses_cias sc
07.          WHERE  sb.coenti = sc.coenti) cogrupa
08.      , (SELECT nogrupa
09.          FROM   ses_cias sc
10.          WHERE  sb.coenti = sc.coenti) nogrupa
11.      , valor
12.  FROM   ses_balanco sb

```

(a)

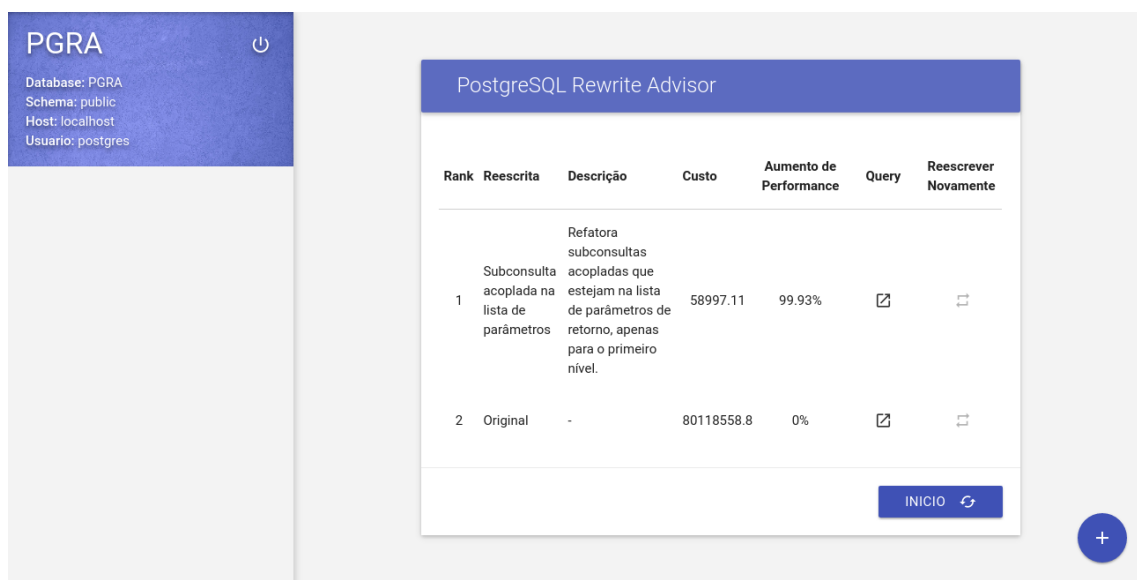
```

01.  SELECT coenti
02.      ,rw_tmp_1.noenti noenti
03.      ,rw_tmp_2.cogrupa cogrupa
04.      ,rw_tmp_3.nogrupa nogrupa
05.      , valor
06.  FROM   ses_balanco sb
07.      LEFT JOIN (SELECT noenti
08.                  ,sc.coenti rw_coenti
09.                  FROM   ses_cias sc) rw_tmp_1
10.      ON sb.coenti = rw_tmp_1.rw_coenti
11.      LEFT JOIN (SELECT cogrupa
12.                  ,sc.coenti rw_coenti
13.                  FROM   ses_cias sc) rw_tmp_2
14.      ON sb.coenti = rw_tmp_2.rw_coenti
15.      LEFT JOIN (SELECT nogrupa
16.                  ,sc.coenti rw_coenti
17.                  FROM   ses_cias sc) rw_tmp_3
18.      ON sb.coenti = rw_tmp_3.rw_coenti





```

(b)

Figura 5.26: Comparação de consultas do terceiro caso de teste.



The screenshot displays the PGRA (PostgreSQL Rewrite Advisor) interface. On the left, a sidebar shows the database context: PGRA, Database: PGRA, Schema: public, Host: localhost, and Usuario: postgres. The main area is titled 'PostgreSQL Rewrite Advisor' and contains a table with two rows of suggestions. The first row (Rank 1) describes a rewrite for a subquery, showing a significant cost reduction and performance improvement. The second row (Rank 2) shows the original query for comparison. At the bottom right of the table, there is an 'INICIO' button and a refresh icon. A circular '+' button is also visible in the bottom right corner of the interface.

Rank	Reescrita	Descrição	Custo	Aumento de Performance	Query	Reescrever Novamente
1	Subconsulta acoplada na lista de parâmetros	Refatora subconsultas acopladas que estejam na lista de parâmetros de retorno, apenas para o primeiro nível.	58997.11	99.93%		
2	Original	-	80118558.8	0%		


INICIO 

Figura 5.27: Tela de resultados do terceiro caso de teste.

```

01.  SELECT coenti
02.      ,valor
03.  FROM    ses_balanco sb
04.  WHERE   sb.coenti = (SELECT sc.coenti
05.                      FROM    ses_cias sc
06.                      WHERE   sc.coenti = sb.coenti)
07.      AND sb.coenti = (SELECT so.nuitem
08.                      FROM    ses_campos so
09.                      WHERE   so.nuitem = sb.coenti)
10.      AND sb.valor > 1000

```

(a)

```

01.  SELECT sb.coenti
02.      ,sb.valor
03.  FROM    ses_balanco sb
04.      JOIN (SELECT sc.coenti
05.              ,sc.coenti rw_coenti
06.              FROM    ses_cias sc) rw_tmp_1
07.      ON rw_tmp_1.rw_coenti = sb.coenti
08.      JOIN (SELECT so.nuitem
09.              ,so.nuitem rw_nuitem
10.              FROM    ses_campos so) rw_tmp_2
11.      ON rw_tmp_2.rw_nuitem = sb.coenti
12.  WHERE   sb.coenti = rw_tmp_1.coenti
13.      AND sb.coenti = rw_tmp_2.nuitem
14.      AND sb.valor > 1000

```

(b)

Figura 5.28: Comparação de consultas do quarto caso de teste.

PGRA
 Database: PGRA
 Schema: public
 Host: localhost
 Usuario: postgres

PostgreSQL Rewrite Advisor

Rank	Reescrita	Descrição	Custo	Aumento de Performance	Query	Reescrever Novamente
1	Subconsulta acoplada na cláusula WHERE	Refatora subconsultas acopladas que estejam nas cláusula WHERE, apenas para o primeiro nível.	24183.62	99.98%		
2	Original	-	99135182.4	0%		

INICIO

Figura 5.29: Tela de resultados do quarto caso de teste.

```

01.  SELECT coenti
02.      , (SELECT noenti
03.          FROM  ses_cias sc
04.          WHERE  sb.coenti = sc.coenti) noenti
05.      , valor
06.  FROM  ses_balanco sb
07.  WHERE  sb.coenti = (SELECT sc.coenti
08.                      FROM  ses_cias sc
09.                      WHERE  sc.coenti = sb.coenti)

```

Figura 5.30: Consulta submetida do quinto caso de teste.

```

01.  SELECT sb.coenti
02.      , (SELECT noenti
03.          FROM    ses_cias sc
04.          WHERE   sb.coenti = sc.coenti) noenti
05.      ,sb.valor
06.  FROM    ses_balanco sb
07.  JOIN (SELECT sc.coenti
08.          ,sc.coenti rw_coenti
09.          FROM    ses_cias sc) rw_tmp_1
10.      ON rw_tmp_1.rw_coenti = sb.coenti
11.  WHERE   sb.coenti = rw_tmp_1.coenti

```

(a)

```

01.  SELECT coenti
02.      ,rw_tmp_1.noenti noenti
03.      ,valor
04.  FROM    ses_balanco sb
05.  LEFT JOIN (SELECT noenti
06.              ,sc.coenti rw_coenti
07.              FROM    ses_cias sc) rw_tmp_1
08.      ON sb.coenti = rw_tmp_1.rw_coenti
09.  WHERE   sb.coenti = (SELECT sc.coenti
10.                      FROM    ses_cias sc
11.                      WHERE   sc.coenti = sb.coenti)

```

(b)

Figura 5.31: Consultas do quinto caso de teste ao realizar a primeira reestruturação.

PGRA

Database: PGRA

Schema: public

Host: localhost

Usuario: postgres

PostgreSQL Rewrite Advisor

Rank	Reescrita	Descrição	Custo	Aumento de Performance	Query	Reescrever Novamente
1	Subconsulta acoplada na cláusula WHERE	Refatora subconsultas acopladas que estejam nas cláusula WHERE, apenas para o primeiro nível.	26731728.24	0.46%		
2	Subconsulta acoplada na lista de parâmetros	Refatora subconsultas acopladas que estejam na lista de parâmetros de retorno, apenas para o primeiro nível.	26766610.14	0.33%		
3	Original	-	26853930.88	0%		

+

Figura 5.32: Tela de resultados do quinto caso de teste ao realizar a primeira reestruturação.

```

01.  SELECT sb.coenti
02.          ,rw_tmp_2.noenti noenti
03.          ,sb.valor
04.  FROM    ses_balanco sb
05.          JOIN (SELECT sc.coenti
06.                  ,sc.coenti rw_coenti
07.                  FROM    ses_cias sc) rw_tmp_1
08.          ON rw_tmp_1.rw_coenti = sb.coenti
09.  LEFT JOIN (SELECT noenti
10.              ,sc.coenti rw_coenti
11.              FROM    ses_cias sc) rw_tmp_2
12.          ON sb.coenti = rw_tmp_2.rw_coenti
13.  WHERE   sb.coenti = rw_tmp_1.coenti

```

Figura 5.33: Consulta do quinto caso de teste ao realizar a segunda reestruturação.

The screenshot shows the PGRA (PostgreSQL Rewrite Advisor) interface. On the left, a sidebar displays the database context: PGRA, Database: PGRA, Schema: public, Host: localhost, and Usuario: postgres. The main area is titled 'PostgreSQL Rewrite Advisor' and contains a table with the following columns: Rank, Reescrita, Descrição, Custo, Aumento de Performance, Query, and Reescrever Novamente. Two rows are visible: Rank 1 shows a rewritten query with a cost of 45207.68 and a 99.83% performance increase; Rank 2 shows the original query with a cost of 26731728.24 and 0% performance increase. Both rows have checkboxes for 'Query' and 'Reescrever Novamente'. At the bottom right of the table is a button labeled 'INICIO' with a refresh icon. A blue circular button with a '+' sign is located at the bottom right of the interface.

Rank	Reescrita	Descrição	Custo	Aumento de Performance	Query	Reescrever Novamente
1	Subconsulta acoplada na lista de parâmetros	Refatora subconsultas acopladas que estejam na lista de parâmetros de retorno, apenas para o primeiro nível.	45207.68	99.83%	<input checked="" type="checkbox"/>	<input type="checkbox"/>
2	Original	-	26731728.24	0%	<input checked="" type="checkbox"/>	<input type="checkbox"/>

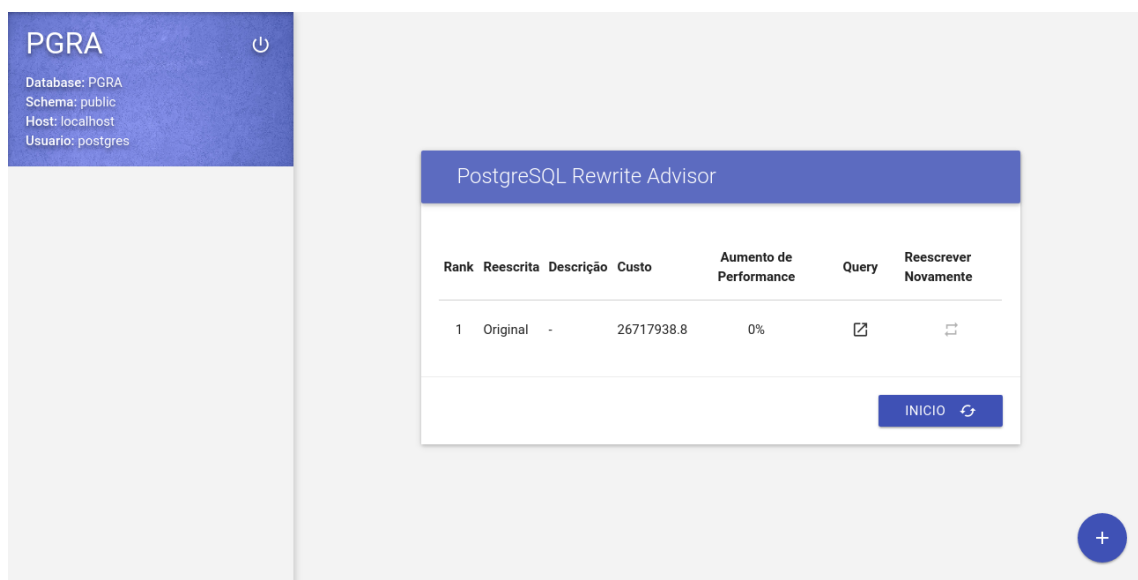
Figura 5.34: Tela de resultados do quinto caso de teste ao realizar a segunda reestruturação.

```



01.  SELECT coenti
02.      ,noenti
03.      ,valor
04.  FROM  (SELECT coenti
05.          , (SELECT noenti
06.              FROM    ses_cias sc
07.              WHERE   sb.coenti = sc.coenti) noenti
08.          ,valor
09.      FROM    ses_balanco sb) sb

```

Figura 5.35: Consulta submetida do sexto caso de teste.



The screenshot displays the PGRA (PostgreSQL Rewrite Advisor) web interface. On the left, a blue sidebar contains the title 'PGRA' and connection details: 'Database: PGRA', 'Schema: public', 'Host: localhost', and 'Usuario: postgres'. The main area features a 'PostgreSQL Rewrite Advisor' window with a table of results. The table has seven columns: Rank, Reescrita, Descrição, Custo, Aumento de Performance, Query, and Reescrever Novamente. A single row is shown with Rank 1, Reescrita 'Original', Descrição '-', and Custo 26717938.8. The 'Aumento de Performance' is 0%. There are icons for the Query and Reescrever Novamente columns. At the bottom right of the table is a blue button labeled 'INICIO' with a refresh icon. A blue circular button with a '+' sign is located in the bottom right corner of the interface.

Rank	Reescrita	Descrição	Custo	Aumento de Performance	Query	Reescrever Novamente
1	Original	-	26717938.8	0%		


INICIO 

Figura 5.36: Tela de resultados do sexto caso de teste.

5.2.7 Quadro Comparativo

A tabela 5.2 apresenta um quadro comparativo entre os tempos dos casos de testes. A coluna **Original** mostra o tempo médio, em milissegundos que a consulta original levou para retornar os dados. A coluna **Reescrita** mostra o tempo médio, em milissegundos que a consulta reescrita levou para retornar os dados. Foi realizado uma bateria de 5 execuções para cada consulta. A coluna **Volumetria** mostra a quantidade de linhas retornadas e a coluna **Performance** mostra o ganho percentual entre o desempenho da consulta original e a consulta reescrita.

Tabela 5.2: Comparativo entre os tempos dos casos de testes.

Caso de Teste	Original	Reescrita	Volumetria	Performance
1	82.738,00	70.947,20	999.978	14,251%
2	83.232,00	29.673,00	999.978	64,349%
3	239.804,00	82.308,60	999.978	65,677%
4	119.395,20	1.657,60	46.302	98,612%
5	158.299,00	72.265,20	999.978	54,349%
6	82.263,40	-	999.978	-

5.3 Restrições

Devido à complexidade do código para realizar a identificação e a reescrita nas classes `Refactor_col_sub_aco` e `Refactor_where_sub_aco`, foram identificadas as seguintes restrições:

- Suporta apenas comandos `SELECT`.
- Reconhece apenas campos, subconsultas e constantes na lista de retorno.
- Reconhece apenas expressões simples no `WHERE`.
- Realiza a reescrita apenas no primeiro nível da consulta.
- As subconsultas devem conter apenas tabelas reais no `FROM`.
- O prefixo `rw_` dentro das subconsultas é reservado para o uso do PGRA.
- Não suporta funções.
- Não suporta `LIMIT`.
- Não suporta visões.

6 CONCLUSÃO

Este trabalho foi motivado pela constatação de que dentre 60% a 80% dos problemas no desempenho do banco de dados são gerados pelo mau uso da SQL, ou seja, quando o desenvolvedor escreve consultas sem preocupação com otimização. O mau uso da linguagem SQL pelos desenvolvedores não permite que o SGBD encontre o plano mais otimizado para uma consulta.

Analisar e transformar uma consulta em outra equivalente, de forma automática ou manual, objetivando aumento de desempenho é denominado reescrita de consulta.

Os SGBDs Oracle, Microsoft SQL Server e PostgreSQL possuem ferramentas externas para sugestão de melhorias de aumento de desempenho. Entretanto, para PostgreSQL, não foi encontrada no repositório oficial uma ferramenta para tratar com reescrita de consulta o problema do mau uso da linguagem SQL.

Este trabalho objetivou o estudo e o desenvolvimento de uma ferramenta automatizada e interativa para melhoria do desempenho por reescrita de consultas SQL para o SGBD PostgreSQL.

Inicialmente foi realizado um estudo dos problemas relacionado ao desempenho de consultas SQL e as formas de otimização de consultas. Foi levantado que diversos aspectos podem interferir no desempenho de um SGBD. Consultas escritas de forma não otimizada, parâmetros de configuração do próprio SGBD, parâmetros de configuração do sistema operacional e especificação do *hardware* podem causar perda de desempenho. A utilização de índices, a coleta de estatísticas, a utilização de visões materializadas e a reescrita de consultas SQL são algumas formas de melhorar o desempenho de uma consulta. Para a reescrita, foram descritas algumas estratégias encontradas na bibliografia.

Foram analisados programas existentes de otimização de consulta em vários SGBDs. SQL Tuning Advisor para o Oracle, Database Engine Tuning Advisor para o Microsoft SQL Server e Postgres Advanced Server para o PostgreSQL. Como o foco do trabalho é na reescrita, vale salientar que apenas a ferramenta SQL Tuning Advisor oferece esta possibilidade.

Foram experimentadas consultas SQL reescrita de forma equivalente, com o intuito de verificar se a reescrita de uma consulta SQL influencia no seu desempenho. Foram utilizadas 12 formas equivalentes para uma mesma consulta, usando os SGBDs PostgreSQL, Microsoft SQL Server e Oracle. No final foi verificado que ocorreu uma variação no tempo em que cada consulta necessitava para retornar os dados.

Por último, foi implementado em Ruby e validado um protótipo de ferramenta para melhoria de desempenho de consultas utilizando reescrita para o SGBD PostgreSQL. O protótipo foi pensando para ser modular e aceitar a adição de novos módulos de otimização por reescrita, desde que respeitem o contrato da interface `Refactor`.

Foram realizados seis testes abrangendo consultas com diversas características. O pro-

tótipo foi bem-sucedido identificando corretamente as situações de reescrita e reestruturando a consulta conforme as regras estipuladas em cada módulo. Porém, foram encontradas algumas restrições.

Apesar de funcional, ainda falta muito para que esta ferramenta possa ser utilizada em produção. Ficam para trabalhos futuros a solução de problemas não detectados, o tratamento das restrições, a adição de outras formas de melhoria de desempenho de consultas e, principalmente, a adição de novos módulos de reescrita são as melhorias que devem ser feitas para transformar este protótipo em uma ferramenta profissional.

Todo o código fonte está licenciado sobre a licença Mozilla Public License Version 2.0 e disponível no GitHub no endereço: <https://github.com/kessler-oliveira/PGRA>.

Por fim, esta ferramenta além de auxiliar os desenvolvedores na otimização de consulta, tem o intuito de sinalizar a importância e impacto que esta ação possui no desempenho dos sistemas.

REFERÊNCIAS

- ANDRADE, L. D. **Otimização de Consultas de Aplicações T-SQL em Ambiente SQL Server 2000**. 2005. Trabalho de Conclusão de Curso — Universidade Federal da Bahia, Instituto de Matemática, Departamento de Ciência da Computação, Salvador - Bahia.
- ASHDOWN, L.; COLGAN, M.; KYTE, T. **Oracle Database SQL Tuning Guide, 12c Release 1 (12.1)**. Redwood Shores - Califórnia: Oracle Corporation, 2015.
- CORADINI, T. P.; CANTARELLI, G. S. Avaliação de desempenho de ferramentas para tuning em banco de dados. **Disciplinarum Sciential Naturais e Tecnológicas**, [S.l.], v.13, n.2, p.201–211, 2012.
- CYRAN, M. **Oracle 8i: designing and tuning for performance**, release 2 (8.1.6). Redwood Shores - Califórnia: Oracle Corporation, 1999.
- ELMASRI, R.; NAVATHE, S. B. **Sistema de banco de dados**. 6.ed. São Paulo: Pearson, 2011.
- EnterpriseDB Corporation. **EDB Postgres Advanced Server Guide v9.5**. Bedford - Massachusetts: EnterpriseDB Corporation, 2016.
- FITTL, L. **pg_query**. San Francisco - Califórnia: pganalyze Team, 2015. Disponível em: <[http:// www.rubydoc.info/ gems/ pg_query/ 0.11.2](http://www.rubydoc.info/gems/pg_query/0.11.2)>. Acesso em: 26 Outubro de 2016.
- HABIMANA, J. Query Optimization Techniques - Tips For Writing Efficient And Faster SQL Queries. **International Journal of Scientific & Technology Research**, IJSTR, v.4, 2015.
- IKE, F. PostgreSQL Tuning: elefante mais rápido que um leopardo. In: PG CONFERENCE BRASIL 2007, 2007. **Anais...** [S.l.: s.n.], 2007.
- JONES, D. **Jump Start Sinatra**. Estados Unidos da América: SitePoint PtyLtd, 2013.
- LÓPEZ, L. C.; DILL, S. Sintonia em Banco de Dados sob Sistemas de Arquivos livres. **Anais SULCOMP**, [S.l.], v.1, 2012.
- Materialize Team. **Materialize**. Disponível em: <[http:// materializecss.com/](http://materializecss.com/)>. Acesso em: 26 Outubro de 2016.
- MULLINS, C. S. **SQL Analysis and Review**. Disponível em: <[http:// www.craigsmullins.com/ sql_a-r.htm](http://www.craigsmullins.com/sql_a-r.htm)>. Acesso em: 18 Junho de 2016.

PE, K. G. **Simple Sinatra MVC Template**. Disponível em: <[https:// github.com/ katgironpe/ simple-sinatra-mvc](https://github.com/katgironpe/simple-sinatra-mvc)>. Acesso em: 26 Outubro de 2016.

ROB, P.; CORONEL, C. **Carlos. Sistemas de Banco de Dados - Projeto, Implementação e Administração**. 8.ed. São Paulo: Cengage Learning, 2011.

Ruby Community. **Ruby Lang**. Disponível em: <<https:// www.ruby-lang.org/ en/>>. Acesso em: 26 Outubro de 2016.

SANTOS CANEDO, F. dos; BRUSCHI, G. C.; SILVA, L. A. da; OLIVEIRA TEIXEIRA, V. de. Gerenciamento e alta disponibilidade em armazenamento de banco de dados. **Ca- derno de Estudos Tecnológicos**, [S.l.], v.1, n.1, 2013.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistema de banco de dados**. 5.ed. Rio de Janeiro: Elsevier, 2006.

The PostgreSQL Global Development Group. **PostgreSQL**: postgresql featured users. Disponível em: <<https:// www.postgresql.org/ about/ users/>>. Acesso em: 18 Novembro de 2016.

The PostgreSQL Global Development Group. **PostgreSQL 9.5.3 Documentation**. Rio Grande: The PostgreSQL Global Development Group, 2016.

TRAMONTINA, G. B. Database Tuning: configurando o interbase e o postgresql. **Campinas**.< [http://www. ic. unicamp. br/~ geovane/mo410-091/Ch20- ConfigInterbasePosgres-art. pdf](http://www.ic.unicamp.br/~geovane/mo410-091/Ch20-ConfigInterbasePosgres-art.pdf), [S.l.], 2008.

GLOSSÁRIO

download é o ato de fazer cópia de uma informação que se encontra num computador remoto.

features funcionalidades ou características.

framework em desenvolvimento de software, é uma abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica.

front-end é responsável por coletar a entrada do usuário em várias formas e processá-la para adequá-la a uma especificação.

hardware conjunto dos componentes físicos (material eletrônico, placas, monitor, equipamentos periféricos etc.) de um computador.

open source significa código aberto. Isso diz respeito ao código-fonte de um software.

parser é o processo de analisar uma sequência de entrada para determinar sua estrutura gramatical segundo uma determinada gramática formal.

WEB nome pelo qual a rede mundial de computadores internet se tornou conhecida a partir de 1991.

APÊNDICE A EXEMPLOS DE CONSULTAS SQL EQUIVALENTES COM DIFERENTES PERFORMANCES

A.1 Consultas Equivalentes

A figura A.1 apresenta a consulta equivalente 01.

A figura A.2 apresenta a consulta equivalente 02.

A figura A.3 apresenta a consulta equivalente 03.

A figura A.4 apresenta a consulta equivalente 04.

A figura A.5 apresenta a consulta equivalente 05.

A figura A.6 apresenta a consulta equivalente 06.

A figura A.7 apresenta a consulta equivalente 07.

A figura A.8 apresenta a consulta equivalente 08.

A figura A.9 apresenta a consulta equivalente 09.

A figura A.10 apresenta a consulta equivalente 10.

A figura A.11 apresenta a consulta equivalente 11.

A figura A.12 apresenta a consulta equivalente 12.

```

01.  SELECT sc.coenti
02.          ,sc.noenti
03.          ,sc.cogrupo
04.          ,sc.nogrupa
05.  FROM    ses_cias sc
06.  WHERE   (SELECT COUNT(1)
07.          FROM    ses_balanco sb
08.          WHERE   sc.coenti = sb.coenti) = 0
09.  ORDER  BY sc.coenti

```

Figura A.1: Consulta equivalente 01.

```

01.  SELECT tmp.coenti
02.          ,tmp.noenti
03.          ,tmp.cogrupo
04.          ,tmp.nogrupa
05.  FROM    (SELECT sc.coenti
06.          ,sc.noenti
07.          ,sc.cogrupo
08.          ,sc.nogrupa
09.          , (SELECT COUNT(1)
10.          FROM    ses_balanco sb
11.          WHERE   sc.coenti = sb.coenti) quant
12.          FROM    ses_cias sc) tmp
13.  WHERE   tmp.quant = 0
14.  ORDER  BY tmp.coenti

```

Figura A.2: Consulta equivalente 02.

```

01.  SELECT sc.coenti
02.          ,sc.noenti
03.          ,sc.cogrupo
04.          ,sc.nogrupa
05.  FROM    ses_cias sc
06.  WHERE   sc.coenti NOT IN (SELECT sb.coenti
07.          FROM    ses_balanco sb)
08.  ORDER  BY sc.coenti

```

Figura A.3: Consulta equivalente 03.

```

01.  SELECT sc.coenti
02.          ,sc.noenti
03.          ,sc.cogruppo
04.          ,sc.nogruppo
05.  FROM    ses_cias sc
06.  WHERE   sc.coenti NOT IN (SELECT DISTINCT sb.coenti
07.                              FROM    ses_balanco sb)
08.  ORDER  BY sc.coenti

```

Figura A.4: Consulta equivalente 04.

```

01.  WITH tmp
02.      AS (SELECT DISTINCT sb.coenti
03.          FROM    ses_balanco sb)
04.
05.  SELECT sc.coenti
06.          ,sc.noenti
07.          ,sc.cogruppo
08.          ,sc.nogruppo
09.  FROM    ses_cias sc
10.  WHERE   sc.coenti NOT IN (SELECT tmp.coenti
11.                              FROM    tmp)
12.  ORDER  BY sc.coenti

```

Figura A.5: Consulta equivalente 05.

```

01.  SELECT sc.coenti
02.          ,sc.noenti
03.          ,sc.cogruppo
04.          ,sc.nogruppo
05.  FROM    ses_cias sc
06.      LEFT JOIN ses_balanco sb
07.          ON sc.coenti = sb.coenti
08.  WHERE   sb.coenti IS NULL
09.  ORDER  BY sc.coenti

```

Figura A.6: Consulta equivalente 06.

```

01.  SELECT sc.coenti
02.          ,sc.noenti
03.          ,sc.cogrupo
04.          ,sc.nogrupa
05.  FROM    ses_cias sc
06.          LEFT JOIN (SELECT DISTINCT sb.coenti coenti
07.                      FROM    ses_balanco sb) tmp
08.          ON sc.coenti = tmp.coenti
09.  WHERE   tmp.coenti IS NULL
10.  ORDER  BY sc.coenti

```

Figura A.7: Consulta equivalente 07.

```

01.  WITH tmp
02.      AS (SELECT DISTINCT sb.coenti coenti
03.          FROM    ses_balanco sb)
04.
05.  SELECT sc.coenti
06.          ,sc.noenti
07.          ,sc.cogrupo
08.          ,sc.nogrupa
09.  FROM    ses_cias sc
10.          LEFT JOIN tmp
11.          ON sc.coenti = tmp.coenti
12.  WHERE   tmp.coenti IS NULL
13.  ORDER  BY sc.coenti

```

Figura A.8: Consulta equivalente 08.

```

01.  SELECT sc.coenti
02.      ,sc.noenti
03.      ,sc.cogruppo
04.      ,sc.nogruppo
05.  FROM  ses_cias sc
06.  WHERE sc.coenti IN ((SELECT sc.coenti
07.                      FROM  ses_cias sc)
08.                     EXCEPT
09.                     (SELECT sb.coenti
10.                      FROM  ses_balanco sb))
11.  ORDER BY sc.coenti

```

Figura A.9: Consulta equivalente 09.

```

01.  SELECT sc.coenti
02.      ,sc.noenti
03.      ,sc.cogruppo
04.      ,sc.nogruppo
05.  FROM  ses_cias sc
06.  WHERE sc.coenti IN ((SELECT sc.coenti
07.                      FROM  ses_cias sc)
08.                     EXCEPT
09.                     (SELECT DISTINCT sb.coenti
10.                      FROM  ses_balanco sb))
11.  ORDER BY sc.coenti

```

Figura A.10: Consulta equivalente 10.

```

01.  SELECT sc.coenti
02.      ,sc.noenti
03.      ,sc.cogruppo
04.      ,sc.nogruppo
05.  FROM  ses_cias sc
06.  WHERE NOT EXISTS (SELECT 1
07.                   FROM  ses_balanco sb
08.                   WHERE  sb.coenti = sc.coenti)
09.  ORDER BY sc.coenti

```

Figura A.11: Consulta equivalente 11.


```

01.  SELECT sc.coenti
02.      ,sc.noenti
03.      ,sc.cogrupo
04.      ,sc.nogrupo
05.  FROM    ses_cias sc
06.  WHERE   NOT EXISTS (SELECT 1
07.                      FROM    (SELECT DISTINCT sb.coenti coenti
08.                                FROM    ses_balanco sb) tmp
09.                      WHERE   tmp.coenti = sc.coenti)
10.  ORDER  BY sc.coenti

```

Figura A.12: Consulta equivalente 12.

A.2 Estatísticas das Consultas no PostgreSQL

O banco de dados utilizado foi derivado da base de dados do Sistema de estatísticas da SUSEP. O diagrama relacional do banco utilizado é apresentado na figura 5.9. A quantidade de linhas em cada tabela é apresentada na tabela 5.1.

A tabela A.1 apresenta as estatísticas de desempenho para cada uma das consultas equivalentes. A coluna **Custo** foi obtida do comando EXPLAIN. A coluna **Tempo** mostra o tempo médio, em milissegundos, de execução de uma bateria 5 execuções para cada consulta. A coluna **Performance** mostra o ganho percentual entre o desempenho da pior consulta, a consulta 2, com o de cada consulta.

Tabela A.1: Estatísticas de desempenho das consultas.

Número	Custo	Tempo	Performance
1	12.383.067,93	103.601,40	0,840%
2	12.383.067,93	104.479,20	0,000%
3	8.926.941,30	77.791,60	25,543%
4	20.153,68	305,80	99,707%
5	20.158,10	315,60	99,698%
6	38.086,33	535,80	99,487%
7	20.183,78	315,60	99,698%
8	20.171,88	310,80	99,703%
9	30.191,29	361,00	99,654%
10	20.196,48	318,60	99,695%
11	38.086,33	529,20	99,493%
12	20.183,78	309,20	99,704%

A.3 Comparação dos Tempos de Execução entre SGBDs

A tabela A.2 apresenta uma comparação entre os tempos de execução de cada consulta nos SGBDs PostgreSQL, Microsoft SQL Server e Oracle. Foi utilizado o tempo médio de execução em milissegundos para comparação pois os custos do plano de execução

em cada SGBD são calculados de forma diferente. Os melhores tempos em cada SGBD estão destacados. Conforme citado na seção 3.1, o SGBD Oracle realiza automaticamente o processo de *query transformation*. É por este motivo que foi observada uma pequena variação entre os tempos das consultas no SGBD Oracle.

Tabela A.2: Comparação de tempo médio entre SGBDs.

Número	PostgreSQL	Microsoft SQL Server	Oracle
1	103.601,40	32.686,20	1.225,20
2	104.479,20	1.458,00	2.022,80
3	77.791,60	33.431,60	1.218,60
4	305,80	33.036,60	1.392,00
5	315,60	33.065,60	1.350,00
6	535,80	1.307,00	1.188,20
7	315,60	1.387,40	1.643,80
8	310,80	1.365,60	1.220,60
9	361,00	32.852,80	1.286,60
10	318,60	32.854,60	1.280,80
11	529,20	32.632,20	1.422,40
12	309,20	32.756,40	1.385,60