UNIVERSIDADE DO VALE DO ITAJAÍ
CENTRO DE CIÊNCIAS TECNOLÓGICAS DA TERRA E DO MAR
CURSO DE CIÊNCIA DA COMPUTAÇÃO

DESEMPENHO DE QUERIES SQL UTILIZANDO COMPLEXIDADE DE ALGORITMOS

por

Filipe Bernardi

Itajaí (SC), dezembro de 2012

UNIVERSIDADE DO VALE DO ITAJAÍ
CENTRO DE CIÊNCIAS TECNOLÓGICAS DA TERRA E DO MAR
CURSO DE CIÊNCIA DA COMPUTAÇÃO

DESEMPENHO DE QUERIES SQL UTILIZANDO COMPLEXIDADE DE ALGORITMOS

Área de Complexidade de Algoritmos

por

Filipe Bernardi

Relatório apresentado à Banca Examinadora do Trabalho Técnico-científico de Conclusão do Curso de Ciência da Computação para análise e aprovação.
Orientador: Rafael de Santiago, M.Sc.

Itajaí (SC), dezembro de 2012

Dedico este trabalho assim como meus esforços a meus pais: Ivacir José Bernardi e Adriana Regina Dalla Costa Bernardi.

AGRADECIMENTOS

Agradeço minha família. Aos meus pais Ivacir José Bernardi por ser o meu herói e Adriana Regina Dalla Costa Bernardi por ser a minha motivação e paixão pelo apoio, compreenssão e pelo carinho que recebo todos os dias. Agradeço também por acreditarem no meu potêncial e apoiarem em todas as decisões sem cobrança e sim confiança em todos os momentos que passamos juntos até minha gradução.

Ao meu orientador Rafael de Santigo, por acreditar no meu potencial e por fornecer conhecimento, cobrança e suporte no decorrer da realização do projeto, além de ser um grande amigo e excelente professor.

Agradeço aos bons amigos que conheci nessa jornada em especial ao Elcio Arthur, Paulo Krieger e Cristiano Elias pela troca de informações e a grande amizade criada nessa imensa jornada que passamos juntos. Posso garantir que foram poucos, mas esses com certeza eu não trocaria por nada.

A todos o meu sincero muito obrigado!

Se quiser mesmo ser feliz, vai ter que aprender a ignorar muita coisa.
Dr. House

RESUMO

BERNARDI, Filipe. **Desempenho de Queries SQL utilizando Complexidade de Algoritmos.** Itajaí, 2012. 73 f. Trabalho Técnico-científico de Conclusão de Curso (Graduação em Ciência da Computação) – Centro de Ciências Tecnológicas da Terra e do Mar, Universidade do Vale do Itajaí, Itajaí, 2012.

As soluções na área de tecnologia baseadas no conceito de banco de dados estão em constante aperfeiçoamento e possuem uma escalabilidade modular para adaptar-se ao crescimento dos sistemas corporativos. O desempenho de um banco de dados é a base para aperfeiçoar os processos e suportar uma base estável para receber grandes quantidades de dados. No contexto deste trabalho, a Complexidade de Algoritmos é aplicada em SGBDs para analisar as principais transações efetuadas pelas cláusulas de consulta com objetivo de mensurar uma melhora no processo de escolha de um plano considerado excelente. A ferramenta produto deste trabalho recebe determinada consulta como entrada e retorna a complexidade das consultas SQL. Depois de submetida à consulta, seu plano de execução é definido e utilizado pela ferramenta para definir a complexidade da consulta. A escolha do SGBD PostgreSQL foi definida baseada no conhecimento sobre o banco de dados, onde se é capaz de analisar sua estrutura de código facilitando a interpretação e análise. Os estudos realizados foram focados no Código-Fonte do SGBD selecionado, onde as cláusulas definidas no escopo do projeto puderam ser analisadas e suas complexidades extraídas. A partir destas informações, foi possível efetuar a modelagem e desenvolvimento da ferramenta e obter resultados favoráveis durante o processo de análise das consultas.

Palavras-chave: Complexidade de Algoritmos. Banco de Dados. Structured Query Language.

ABSTRACT

The solutions in technology, based on the concept of database, are becoming increasingly sophisticated and feature a modular scalability to adapt to the growth of enterprise systems. The database performances are the basis for improved processes and support a stable base for receiving large amounts of data. In the context of this work, the complexity of algorithms is applied in DBMSs to analyze the main transactions performed by the query clauses in order to measure an improvement in the process of choosing a plan considered excellent. The tool receives product of this work given query as input and returns the complexity of SQL queries. After having been subjected to consultation, its execution plan is defined and used by the tool to define the complexity of the query. The choice of PostgreSQL DBMS was defined based on knowledge of the database, where you can analyze your code structure facilitating the interpretation and analysis. The studies were focused on Source Code selected DBMS, where the terms defined in the project scope could be analyzed and extracted its complexities. From this information, it was possible to perform the modeling and development of the tool and obtain favorable results during the analysis process of consultations.

Keywords: *Complexity of Algorithms. Database. Sructure Query Language.*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE EQUAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
BSD	Berkeley Software Distribution
CGI	Common Gateway Interface
DML	Data Manipulation Language
GPL	General Public License
IBM	Internacional Business Machines
ISAM	Indexed Sequential Access Method
JSON	JavaScript Object Notation
OEM	Original Equipment Manufacturer
SGBD	Sistema Gerenciador de Banco De Dados
SQL	Structured Query Language
TTC	Trabalho Técnico-Científico de Conclusão de Curso
UNIVALI	Universidade do Vale do Itajaí
UML	Unified Modeling Language
WWW	World Wide Web

LISTA DE SÍMBOLOS

π	Projeção
σ	Seleção
| X |	Junção
∩	Intersecção
∪	União
×	Produto Cartesiano
O	Ômicron
Θ	Tetha
Ω	Ômega
∑	Somatório
μ	Mi

SUMÁRIO

1 INTRODUÇÃO

A grande quantidade de informações geradas diariamente por empresas demanda a cada dia um aumento na capacidade de armazenamento dos dados em sistemas computacionais. O grande objetivo do armazenamento em sistemas computacionais é possibilitar a recuperação das informações de forma precisa, ágil e segura. Atualmente os Sistemas Gerenciadores de Bancos de Dados (SGBDs) atendem todos os requisitos de precisão, velocidade e segurança no retorno dos dados e estão evoluindo a cada dia, motivando entusiastas e estudantes a investir no estudo da área de Banco de Dados.

O tempo necessário para a recuperação de informações depende de diversos fatores físicos (*hardware*) e lógicos (*software*). Dentre os fatores físicos é possível mencionar a velocidade e a capacidade do disco no qual os dados estão armazenados, a quantidade de memória disponível e a capacidade de processamento do computador. Os fatores lógicos envolvem o método utilizado na recuperação dos dados e o projeto das estruturas que representam os dados no banco de dados. Este estudo tem como foco os métodos de recuperação de dados.

A manipulação de dados nos bancos de dados envolve recuperação, inserção, modificação e eliminação de informações. A Linguagem de Manipulação de Dados (DML) é a linguagem que viabiliza o acesso e a manipulação dos dados. DMLs procedurais exigem que o usuário especifique quais dados se deseja e a forma como esses dados serão obtidos. DMLs não procedurais exigem que o usuário informe quais dados necessite sem especificar como obtê-los (GUTTOSKI, 2006, p.1).

Uma consulta no banco de dados é uma requisição para retorno das informações contidas no banco de dados. A parte da DML responsável pela recuperação de informações é conhecida como linguagem de consultas ou *queries*.

Com análise e estudo sobre desempenho e otimização de consultas no banco de dados, objetiva-se a redução do tempo de resposta das consultas provenientes do servidor. O método de aprimoramento de consultas SQL implica no planejamento, implementação e implantação. Deve-se considerar no decorrer do projeto o desempenho antes da codificação (GUTTOSKI, 2006, p.1).

O custo de processamento de uma consulta é determinado pelo acesso ao disco, que é lento comparado ao acesso à memória principal. É possível definir várias estratégias para processar uma determinada consulta, principalmente se a consulta for complexa. A diferença

entre uma estratégia boa e uma ruim em termos de números de acesso ao disco é significativa e pode alcançar uma grande magnitude (SILBERSCHATZ, 1999).

A complexidade de algoritmos reflete o esforço computacional requerido para executá-lo. Esse esforço mede a quantidade de trabalho em termos de tempo de execução. É possível medir experimentalmente a quantidade de trabalho de um sistema computacional ao decorrer da sua execução. Uma alternativa útil é fazer análises matemáticas do algoritmo de maneira geral. Essa análise, além de independente da implementação permite, muitas vezes, antecipar o cálculo da complexidade para a fase de projeto do algoritmo. Esse gênero de análise é útil para avaliar as dificuldades intrínsecas da resolução computacional de um problema. (TOSCANI; VELOSO, 2008).

No contexto de desempenho, existem várias heurísticas que podem ser utilizadas para mensurar a quantidade de tempo que algum sistema computacional consumirá no decorrer de sua execução. Uma das medidas mais aceitas sob um algoritmo relaciona a complexidade computacional, na qual uma função representa a quantidade de operações que serão executadas dadas uma determinada configuração de entrada (ARORA; BARAK, 2009).

O objetivo deste trabalho foi realizar uma pesquisa sobre a complexidade de consultas de um determinado SGBD. Esta pesquisa gerou como produto uma ferramenta que, recebendo a consulta SQL, indica qual a complexidade da mesma. Para mensurar a complexidade, um estudo foi realizado sobre tipos de algoritmos utilizados em diferentes configurações de consulta. É importante destacar que em uma análise de complexidade o volume de entrada (quantidade de dados no banco) não é considerado, mas sim, a eficiência do algoritmo sobre uma entrada de tamanho N.

No contexto deste trabalho, foi utilizada a perspectiva pessimista para as análises, com a intenção de identificar qual a função de complexidade utilizando ordens assintóticas de complexidade.

1.1 PROBLEMATIZAÇÃO

1.1.1 Formulação do Problema

A análise de Complexidade de Algoritmos é um campo de pesquisa muito amplo. Atualmente Sistemas Gerenciadores de Bancos de Dados possuem um plano de decisão capaz de analisar essa complexidade e optar pelo melhor plano de execução.

No entanto os Sistemas Gerenciadores de Banco de Dados retornam seu tempo de execução, sem representar a real complexidade e como a consulta foi formalizada e estruturada antes de durante o processo de execução.

1.1.2 Solução Proposta

Dentre as etapas de desenvolvimento de sistemas, este trabalho foca no aprimoramento das complexidades que estão implícitas em cada cláusula SQL de um Sistema Gerenciador de Banco de Dados, que serão desmembradas no decorrer do projeto e agora serão analisadas de acordo com a estrutura montada manualmente pelo usuário.

Este trabalho desenvolveu uma ferramenta computacional que, a partir de determinada consulta SQL dada como entrada é capaz de analisar a consulta e retornar sua complexidade (esforço computacional). As cláusulas analisadas serão: SELECT, FROM, WHERE, GROUP BY e ORDER BY. Estas cláusulas fazem parte do grupo de cláusulas SQL que é responsável pela extração, ordernação e agrupamento das informações provindas do SGBD Postgres, foco do projeto.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Especificar como é realizada a análise de complexidade de algoritmos utilizados em consultas SQL em um sistema de gerenciador de banco de dados e desenvolver uma ferramenta para indicar a complexidade dada uma determinada consulta.

1.2.2 Objetivos Específicos

- Selecionar o SGBD que permita visualizar como o processo das consultas é efetuado;
- Pontuar as principais características de consultas SQL com métricas de complexidade de algoritmo;
- Validar a ferramenta desenvolvida de acordo com os resultados gerados para as consultas de entrada de dados;

- Desenvolver a ferramenta para indicar a complexidade de uma determinada consulta SQL, recebida como entrada, utilizando ordens assintóticas que é definida pelo crescimento da complexidade para grandes entradas de dados.

1.3 Metodologia

A metodologia empregada nesse trabalho é disseminada em quatro etapas consideradas fundamentais: (i) Pesquisa, (ii) Fundamentação Teórica, (iii) Desenvolvimento dos algoritmos e (iv) Documentação.

Para etapa de pesquisa (i), foram realizadas pesquisas sobre todos os assuntos relacionados e correlacionados como: SGBDs open-source, álgebra relacional, complexidade pessimista, complexidade média, desempenho de algoritmos, busca sequencial e teorema mestre. Todo embasamento teórico tem como objetivo a produção da ferramenta proposta e definição do processo.

Na segunda etapa (ii) Fundamentação Teórica foi realizado um levantamento teórico a respeito do comportamento das consultas SQL e a definição da metodologia a ser aplicada, assim como, contemplar a busca por algoritmos que auxiliem na melhor forma de se obter a informação provinda do SGBD.

Na terceira etapa (iii) desenvolvimento dos algoritmos foi realizado um estudo referente às ferramentas e linguagens de programação fornecendo meios de desenvolver a aplicação.

Na quarta etapa (iv) documentação foi realizado o processo de redação de toda documentação deste trabalho de conclusão de curso que acompanhou todo processo de pesquisa e fundamentação.

1.4 Estrutura do trabalho

Este trabalho de conclusão de curso está estruturado em 4 capítulos. Introdução, Fundamentação teórica, Projeto e considerações finais.

O Capítulo 1, Introdução, apresenta uma visão global do trabalho, permitindo analisar à contextualização do projeto.

O Capítulo 2, Fundamentação Teórica, apresenta uma revisão bibliográfica sobre os assuntos que compõem esse trabalho: Banco de Dados, Complexidade de Algoritmos, álgebra

relacional e teorema mestre. Além dessa revisão, em paralelo, foram pesquisados sobre os algoritmos de programação paralela e a metodologia de pesquisas SQL.

O Capítulo 3, Projeto, aborda a ferramenta proposta a ser desenvolvida, suas especificações e os Diagramas UML (Unified Modeling Language). Neste capítulo também são descritos casos de uso, testes e requisitos do sistema.

O Capítulo 4, Considerações Finais, conclui o término do documento apresentado e apresenta técnicas que serão utilizadas posteriormente para compor o corpo do projeto.

2 FUNDAMENTAÇÃO TEÓRICA

Para compreensão de todas as informações necessárias para o desenvolvimento do projeto, foram realizadas buscas e estudos em livros, artigos e documentos digitais para alcançar os objetivos que abrangem o escopo do trabalho. Dentre todas as informações levantandas, destacam-se: conceitos sobre banco de dados, métodos de otimização de SGBD e análise de complexidade de algoritmos. As informações e estudos realizados podem ser observados nesta seção.

2.1 SGBD

A história do SQL inicia em um laboratório da IBM em San Jose, Califórnia (Estados Unidos da América), onde a linguagem SQL foi desenvolvida em 1970. A IBM, junto com outros fornecedores de banco de dados relacionais, desejava uma linguagem padronizada para acessar e manipular dados em um banco de dados relacional. O pesquisador da IBM Dr. E.F.Codd levou ao desenvolvimento de um produto modelo de dado relacional chamado SEQUEL ou Linguagem de Consulta Estruturada, que, com o passar dos anos, transformou-se em SQL, (Structured Query Language) uma linguagem declarativa e baseada na Álgebra Relacional, que está apoiada em duas vertentes da matemática conhecidas como teoria dos conjuntos e a lógica de predicado de primeira ordem (KLINE; KLINE, 2010).

A idéia original do SQL só previa seu uso de forma interativa. Após sofrer alguns acréscimos, ela passou também a ter capacidade de ser utilizada em linguagens hospedeiras, tais como COBOL, FORTRAN e C. (MACHADO; ABREU, 1996).

Atualmente, a linguagem SQL assume um papel importante nos sistemas de gerenciamento de banco de dados, podendo assumir vários enfoques como:

1. Linguagem interativa de consulta (*query Adhoc*)**:** por meio de SQL, os usuários podem montar consultas sem a necessidade da criação de um programa, podendo utilizar interface gráfica ou ferramentas de montagem de relatório;
2. Linguagem de programação para acesso a banco de dados**:** comandos SQL embutidos em programas de aplicação que acessam os dados armazenados;
3. Linguagem de administração de banco de dados: responsável pela administração do banco de dados (Administrador do banco de dados) podem utilizar comandos SQL para realizar suas tarefas.

Em essência, um sistema de bancos de dados é um *"sistema computadorizado de armazenamento de registros"* (DATE, 2000). O banco de dados pode ser visto como o equivalente eletrônico de um armário de arquivamento. Os usuários do sistema poderão executar diversas operações sobre tais arquivos – por exemplo:

- Acrescentar novos registros ao banco de dados;
- Inserir novos dados em registros existentes;
- Recuperar dados de registros existentes;
- Alterar dados em registros existentes;
- Eliminar dados de registros existentes;
- Remover registros existentes do banco de dados.

O banco de dados cujo propósito geral é armazenar e permitir ao usuário buscar e atualizar informações quando solicitado. As informações em questão podem ser qualquer contexto desde que tenha significado para o indivíduo ou a organização a que o sistema deve servir – em outras palavras, tudo o que é necessário para auxiliar no processo geral de tomada de decisões de negócio desse indivíduo ou dessa organização.

Algumas implicações adicionais da utilização da abordagem do usuário de um banco de dados que beneficiam as organizações (DATE, 2000):

- Uso de padrões entre os usuários de um banco de dados em uma grande organização facilita a comunicação e cooperação;
- A flexibilidade permite a alteração da estrutura do banco de dados quando necessário alterar os requisitos de forma evolutiva sem afetar os dados armazenados e aplicações existentes;
- A disponibilidade da atualização das informações é essencial para aplicações de processamento de transações;
- A economia de escala permite a consolidação dos dados e das aplicações, desta forma reduzindo a perda por superposição entre as atividades de processamento de dados possibilitando a redução dos investimentos em equipamentos de informática, reduzindo o custo total da operação e gerenciamento (DATE, 2000).

O Sistema Gerenciador de Banco de Dados (SGBDs), conhecido como relacional proporcionam um ambiente produtivo para manipulação de informações. A partir de uma linguagem de alto nível, tais sistemas permitem que seus usuários descrevam consultas de uma maneira simples sem definir com isso, detalhes relacionados ao seu processamento. Tais detalhes são de responsabilidade do próprio SGBD, o qual deve escolher, através de um sofisticado processo de otimização e planejamento, uma alternativa eficiente para a obtenção dessas informações. A otimização de junções é uma das mais importantes e complexas dentre todas as fases que compõem este processo (DATE 2000). Guttoski (2006, p. VI) afirma que a definição da melhor ordem de junções somente pode ser realizada em condições relativamente simples, através do uso de algoritmos de busca exaustiva fortemente baseada na programação dinâmica.

Na Figura 1 é esboçado o Sistema de Gerenciamento de Bancos de Dados (SGBD). A figura representa os quatro componentes principais:

1. Usuários: podem ser divididos em três grupos: programadores de aplicações, usuários finais e administradores de banco de dados;
2. Hardware: consistem em volumes de armazenamento secundário, processadores de hardware e memória principal;
3. Dados: os dados estão integrados e compartilhados no SGBD;
4. Software: a função geral fornecida pelo SGBD é de isolar os usuários do banco de dados dos detalhes no nível de hardware e abstrair detalhes não relevantes.

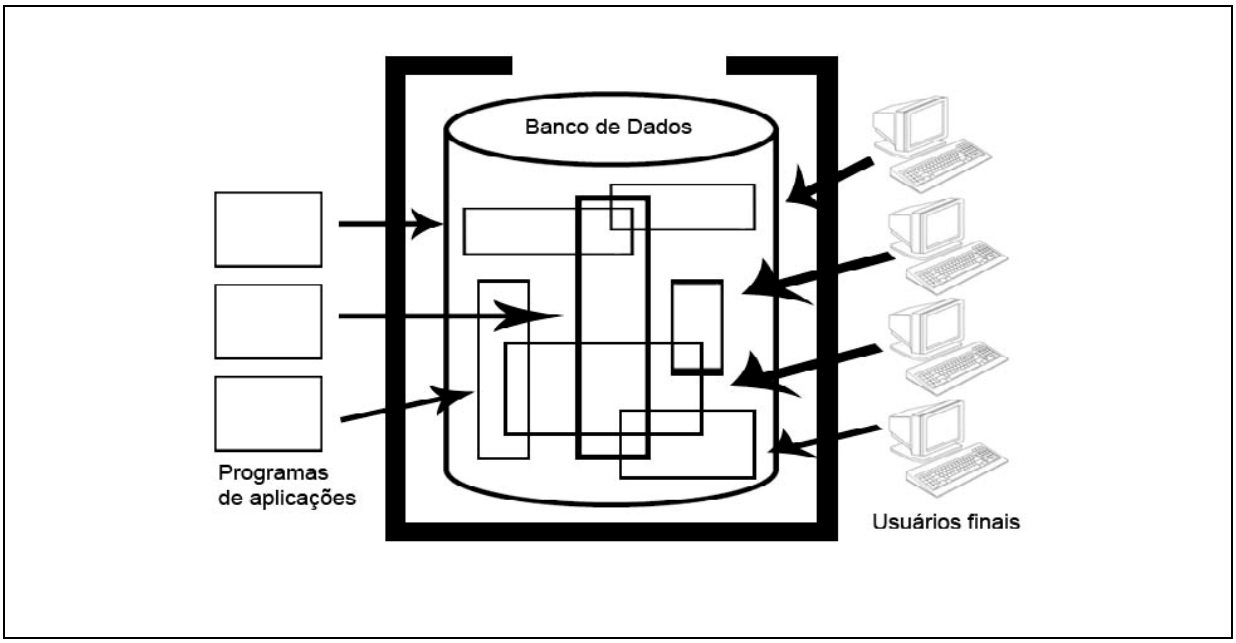


Figura 1. Representação simplificada de um sistema de banco de dados
Fonte*:* Adaptado de Date (2003).

2.1.1 Comparativo entre SGBDs e opção do Postgres

A decisão por definir qual SGBD seguir como foco da pesquisa foi realizada, avaliando todos os pontos positivos e negativos de cada SGBD. A opção pela escolha de um banco de dados de código aberto foi tomada no início do projeto, devido à facilidade de se obter documentação, pesquisas e uma comunidade ativa onde houvesse a possibilidade de compartilhar conhecimento e troca e informações com a comunidade desenvolvedora.

Nesta pesquisa foram abordados dois SGBDs conhecidos pela comunidade de desenvolvimento OPEN SOURCE, que são o MySQL e o PostgresSQL. Ambos SGBDs possuem vantagens e características em comum, porém o foco da pesquisa depende dos recursos oferecidos e da documentação.

A pesquisa sobre a definição do SGBD a ser utilizado não levou em consideração resultados de *Benchmark* (comparação de desempenho), pois o foco em hardware e cargas de trabalho utilizadas serão abordados no decorrer da pesquisa, sob perspectiva discutida na introdução deste documento serão analisados os algoritmos utilizados na pesquisa, não o conjunto carga de trabalho de hardware.

Nos tópicos a seguir é possível analisar um apanhado geral das características de cada SGBD, explicação sobre seus termos, foco e compreender o porquê escolher o SGBD PostgreSQL.

2.1.2 MySQL

O MySQL é um SGBD relacional, de licença dupla (sendo uma delas OPEN-SOURCE), projetado inicialmente para trabalhar com aplicações de pequeno e médio porte, mas no cenário atual atende aplicações de grande porte. Possui características que um banco de dados de grande porte precisa, sendo reconhecido por algumas entidades como o "banco de dados *open-source*" (MILANI,2006).

MySQL teve origem quando os desenvolvedores David Armark, Allan Larsson e Michael “Monty” Widenius, na década de 90, precisaram de uma *interface* SQL compatível com as rotinas ISAM que utilizavam em suas aplicações e tabelas. Em um primeiro momento, tentaram utilizar a API Mysql, escreveram em C e C++ uma nova API que deu origem ao MySQL (MILANI,2006).

O resultado gerado pela criação da nova API desenvolvida começou a ser difundido e seus criadores fundaram a empresa responsável por sua manutenção que é a MySQLAB.

O MySQL é desenvolvido e distribuído por meio de duas licenças que dependerá do tipo de uso da ferramenta. Na maioria dos casos, seu uso é livre. (MILANI,2006).

A primeira, software livre, é baseada nas cláusulas da GNU-GPL (General Public License), a qual estabelece o que se pode ou não com a ferramenta e demais recursos. Além do programa, o seu Código-fonte também é disponibilizado para que qualquer pessoa possa adaptá-lo às suas necessidades; contudo, todas essas situações serão tratadas e detalhadas na licença GNU-GPL. (*ibidem*).

A licença do tipo GNU-GPL baseia-se nos seguintes principios.

- Permite utilizar o software para qualquer propósito;
- Permite a livre distribuição do software;
- Permite que seu funcionamento seja estudado a partir de seu código-fonte;
- Permite que seu código-fonte seja alterado para evoluir a ferramenta, desde que seu novo código-fonte continue sendo livre seguindo mesma licença (GNU-GPL).

A licença comercial é utilizada em algumas situações sobre como embutir o MySQL dentro das aplicações comerciais (OEMs, ou seja, fora dos termos da GNU-GPL), obter suporte diferenciado, ou obter pacotes com mais ferramentas. (*ibidem*).

O MySQL possui foco em facilidade de administração e baixo consumo de recursos do hardware. Tornou-se popular graças à Internet, pois os bancos tradicionais tinham tempos

de conexão extremamente elevados, inadequados para aplicações CGI. Para atingir seus objetivos, não implementava funções com grande *overhead* como integridade referencial, aplicar, desfazer e níveis de isolamento de transação. Nas novas versões é possível utilizar todas essas funções que foram implementas e aprimoradas (OLIVEIRA, 2010).

É possível acompanhar o processo de desenvolvimento do MySQL, acessar os foruns, listas de discussão e canais pelo site oficial “mysql.org”.

2.1.3 PostgreSQL

Em 1986, na universidade da Califória em Berkeley o professor especialista em tecnologia de banco de dados chamado Michael Stonebraker iniciou a construção de um novo e otimizado sistema de banco de dados. Apesar de já ter obtido grandes resultados com seu projeto anterior, o INGRES, Stonebraker percebeu que o código do INGRES tornou-se demasiadamente grande e pesado que, ou invés de tentar implementar a nova versão do projeto INGRES, ele deveria reconstruir um novo sistema do zero, cujo resultado foi o que ele denominou de POSTGRES. (GILMORE; TREAT, 2006).

Ao longo dos próximos oito anos (após 1986), o POSTGRES cresceu em popularidade, especialmente entre a comunidade de pesquisa. O projeto POSTGRES foi oficialmente encerrado na versão 4.2. Graças a sua liberação sob licença BSD que permite a acesso do código objeto ou o próprio executável. Mesmo sem a liberação do Código-fonte, não foi o fim do projeto de desenvolvimento do banco de dados. (GILMORE; TREAT, 2006).

O projeto desenvolvido foi disponibilizado nos principais canais de compartilhamento de informações como Internet, e em 1994, Andrew Yu e Jolly Ched adicionaram novas funcionalidades como o interpretador SQL que substituiu o anterior sistema de linguagem QUEL e posteriormente lançado como uma nova versão Postgres95. O nome PostgreSQL homenageava o projeto POSTGRES original, mas com novas funcionalidades e novos recursos que haviam sido implementados. O processo de versionamento foi criado de maneira incremental com a versão original do projeto. (GILMORE; TREAT, 2006).

O POSTGRES atualmente é um dos mais populares projetos Open-Source. Como muitos dos projetos que sairam do BIND (BSD Unix, sendmail) estão entre suas contribuições. No cenário atual, o POSTGRES possui uma gama de aplicativos, sites e uma parte da espinha dorsal da Internet. (*ibidem*).

O PostgreSQL não utiliza licença GNU, mas sim, a licença BSD (Barkeley Software Distribution). Originada com o sistema operacional FreeBSD, a licença BSD obteve

reconhecimento e vários outros software também a utilizam graças à sua liberdade frente a outras licenças de software. Isto torna o código muito mais acessível para diversos tipos de utilização, incluindo a livre utilização da ferramenta até mesmo para fins comerciais. (MILANI, 2008).

É interessante salientar que todas as empresas envolvidas com o PostgreSQL, nenhuma delas tem qualquer tipo de propriedade do código ou controlar a direção de desenvolvimento do PostgreSQL. Tudo isso é executado por voluntários da comunidade e os desenvolvedores, que coletivamente controlam o que é adicionado ao sistema central. (GILMORE; TREAT, 2006).

Para banco de dados muito grandes, complexos e que exige confiabilidade e escalabilidade é recomendado utilizar o PostgreSQL quando se trata de banco de dado Open-Source.

Por exemplo, aplicações com fortes componentes transacionais e que necessitam de tipos de dados especializados, como Sistemas de Informações Geográficas e repositórios de meta-dados. (OLIVEIRA, 2010).

2.1.4 MYSQL X POSTGRESQL

O Quadro 1 especifica algumas diferenças entre os SGBDs abordados. Na coluna Atributos são destacadas as particularidades de cada SGBD, tipo de licenças, finalidade como foco, existência de documentação adequada e por fim a comunidade ativa.

Os SGBDs abordados são Open-Source, que permitem a exploração do código-fonte. No quesito foco, ambos permitem uma análise apropriada, mas com focos distintos. O PostgreSQL é considerado um SGBD robusto e suporta aplicações de alto nível detalhamento com um foco diferente do MySQL que tem uma visão baseada na agilidade e na maioria dos casos voltado a plataforma WEB.

A documentação e a comunidade ativa de ambos os SGBDs estão disponíveis no site do desenvolvedor e possuem uma comunidade ativa no auxilio da construção da ferramenta.

Atributos	**PostgreSQL**	**MySQL**
Licença	BSD	Pública GNU
Foco	Aplicações complexas	Agilidade
Documentação	Disponível no site do desenvolvedor	Disponível no site do desenvolvedor
Comunidade Ativa	Existente	Existente

Quadro 1. Informações sobre SGBDs

A escolha pelo banco de dados PostgreSQL foi tomada devido à acessibilidade rápida das informações, conhecimento da ferramenta e a comunidade ativa apoiar a pesquisa na área. Ambos os SGBDs analisados possuem argumentos convincentes para uma linha de estudo, porém o PostgreSQL foi uma decisão de projeto.

2.1.5 Processamento da consulta

O processamento de consultas refere-se ao conjunto de atividades envolvidas na extração de dados de um banco de dados. As atividades incluem tradução de consultas em linguagem de banco de dados de alto nível para expressões que podem ser usadas no nível físico do sistema de arquivos, uma série de transformações de otimizações da consulta e a avaliação real das consultas (KORTH; SILBERSCHATZ; SUDARSHAN, 2006).

Antes do processamento de qualquer consulta iniciar, o sistema precisa traduzir a consulta para uma forma mais utilizável e que possa ser traduzido para uma linguagem interna. A linguagem de banco de dados SQL é ideal para a ação humana, porém não para representação interna do sistema de consulta. A representação interna mais útil e adequada é baseada na álgebra relacional (KORTH; SILBERSCHATZ; SUDARSHAN, 2006).

Uma das primeiras ações que o sistema precisa tomar no processamento da consulta é traduzi-las para a linguagem interna do banco de dados. O processo de tradução das consultas é semelhante ao analisador *parser*, que é um algoritmo de análise sintática para as Gramáticas Livre de Contexto de um compilador. Ao efetuar a análise e gerar a forma interna da consulta, o analisador verifica a sintaxe de entrada da consulta e busca a relação entre os dados de entrada e se o padrão de nomes é entendível pelo banco de dados. O sistema constrói uma representação de árvore de análise da consulta que poderá traduzir para uma expressão da álgebra relacional (KORTH; SILBERSCHATZ; SUDARSHAN, 2006).

Na Figura 2 é possível acompanhar as etapas envolvidas no processamento de uma consulta e destacar as etapas abaixo:

- Análise e tradução convertem a consulta para interpretação interna do SGBD;
- Otimização transforma a expressão em um modelo de álgebra relacional;
- Avaliação é o mecanismo responsável pela execução da consulta, executa o plano e retorna a resposta.

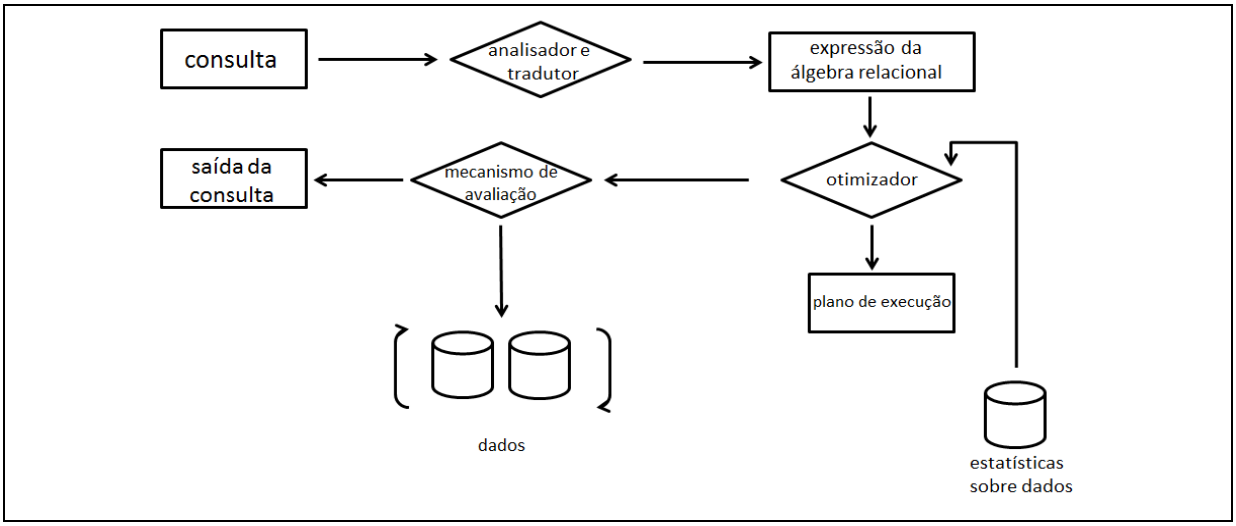


Figura 2. Etapa no processamento da consulta no banco de dados

Fonte*:* Adaptado de Silberschatz (2006).

Uma consulta no banco de dados pode ser efetuada de diversos métodos para obter determinado resultado. A consulta pode ser traduzida para uma expressão de álgebra relacional em um dentre vários métodos. A representação de uma consulta em álgebra relacional especifíca apenas como avaliar de forma parcial a expressão. O Quadro 2 ilustra uma consulta básica no banco de dados, buscando todos os saldos de uma tabela chamada conta com a condição de que o saldo seja menor que 2500.

```
SELECT saldo
FROM conta
WHERE saldo < 2500
```

Quadro 2. Consulta SQL

Na Seção 2.1.5.1 será detalhado como funciona a álgebra relacional e sua linguagem de consulta procedural.

2.1.5.1 Álgebra Relacional

A álgebra relacional é definida como uma linguagem formal de consultas procedurais e faz parte da manipulação do modelo relacional. Consiste em um conjunto de operações que determinam uma nova ligação a partir de uma ou mais relações já existentes no banco de dados. A álgebra relacional fornece os procedimentos específicos para a realização de uma sequência de operações, de tal forma a obter o resultado desejado independente da forma de tratamento do banco de dados (KORTH, SILBERSCHATZ e SUDARSHAN, 2006).

Álgebra relacional é uma coleção de operações utilizadas para manipular relações entre as consultas. As operações são utilizadas para selecionar tuplas (lista ordenada de colunas) de uma determinada relação ou simplesmente para combinar tuplas relacionadas a diversas relações existentes no banco de dados com o propósito de especificar uma requisição de recuperação (DATE, 2003).

Álgebra relacional foi projetada originalmente assumindo que as tabelas fossem como conjuntos. As tabelas em SQL são tidas como conjuntos múltiplos, desta forma são possíveis mais de uma ocorrência para o mesmo resultado (DATE, 2003).

Permitir a escrita de expressões relacionais é o objetivo principal da álgebra relacional. A definição dessas expressões tem por finalidade o princípio de “busca”, que baseado na operação define a busca dos dados a serem pesquisados e retornados pela consulta.

As operações da álgebra relacional são divididas em dois grupos. O primeiro inclui um conjunto de operações da teoria dos conjuntos: UNION (união), INTERSECTION (interseção), DIFFERENCE (diferença) e CARTESIAN PRODUCT (produto cartesiano). O segundo grupo consiste especialmente de operações projetadas para banco de dados relacionais como: SELECT (seleção), PROJECT (projeção) e JOIN (junção) (DATE,2003).

Com base no foco da pesquisa, somente as operações com relevância serão abordadas: SELECT, PROJECT, JOIN e CARTESIAN PRODUCT.

Conforme descrito no Quadro 3 é possível verificar a notação dos operadores de álgebra relacional, sua operação e sua sintaxe para utilização.

Operação	**Notação**	**Sintaxe**
Projeção (*Project*)	π	π <lista de atributos> - (Relação)
Seleção (*Select*)	σ	σ <condição de seleção> - (Seleção)
Junção natural (*Join*)	| X |	(Relação A | X | Relação B)
Divisão (*Division*)	÷	(Relação A ÷ Relação B)
Interseção (*Intersection*)	∩	(Relação A ∩ Relação B)
União (*Union*)	∪	(Relação A ∪ Relação B)
Produto Cartesiano (*Cartersian product*)	×	(Relação A × Relação B)
Diferença (*Set difference*)	–	(Relação A – Relação B)

Quadro 3. Notação e Sintaxe dos operadores de álgebra relacional

O operador de SELECT é utilizado para selecionar um subconjunto de tuplas de uma relação das quais devem satisfazer uma condição de seleção. Exemplo, a seleção de um subconjunto de tuplas da relação de EMPREGADOS que trabalham no departamento 2 ou que possuam um salário maior que 5000. Cada condição é especificada individualmente usando a operação de SELECT abaixo: (DATE, 2003).

Conforme Quadro 4, a operação de SELECT é denotada por:

σ <condição de seleção> (<nome da relação>)

Quadro 4. Notação da condição select

Conforme Quadro 5, operador SELECT é **comutativo**; isto é,

σ <cond1> (σ <cond2> (R))= σ<cond2> (σ<cond1> (R))

Quadro 5. Modelo de select comutativo

Uma seqüência de SELECTs pode ser aplicada em qualquer ordem, conforme Quadro 6. Além disso, pode-se sempre trocar operadores SELECT em cascata com a conjuntiva AND; isto é:

σ <cond1> (σ <cond2> (...(σ <condn> (R))...))= σ <cond1> AND <cond2> AND ... AND <condn>(R)

Quadro 6. Sequência de select com condição conjuntiva

O operador PROJECT seleciona algumas colunas da tabela e descarta outras. Quando houver interesse sobre certos atributos da relação, utiliza-se o PROJECT para projetar a relação sobre esses atributos. O PROJECT só pode ser executado em uma relação (DATE, 2003). É possível implementar a projeção sobre cada tupla (gerando uma relação que retornaria registros duplicados caso houvesse), e depois removendo os registros em caso de duplicidade (KORTH; SILBERSCHATZ; SUDARSHAN, 2006).

No sistema de projeção, caso os atributos na lista de projeção incluir a chave de relação, não haverá registros duplicados, desta forma a eliminação de duplicados não é exigida (KORTH, SILBERSCHATZ e SUDARSHAN, 2006).

π Nome, Universidade, Id_Professor (ALUNOS)

Quadro 7. Modelo de projeção

O Quadro 7 projetará uma nova relação com os atributos: Nome, Universidade e Id_Professor. A Tabela 1 descrita abaixo representa o resultado dessa operação.

Tabela 1. Representação do resultado citado no exemplo anterior

Nome	Universidade	Id_Professor
Filipe	UNIVALI	00001
Rafael	UNIVALI	00002
José	UNIVALI	00002
Ralf	UFSC	00001
Karine	UNIVALI	00001
Carlos	UNIVALI	00002
Pietro	UFSC	00002
Maria	UFSC	00003

A Tabela 1 representa os valores retornados pela consulta efetuada no Quadro 7, onde algumas colunas da tabela “alunos” foram selecionadas sem nenhuma restrição ou condição.

O operador JOIN é utilizado para combinar tuplas agrupadas por relações em uma única tupla. Esta operação é muito importante para qualquer banco de dados relacional, pois permite processar enlaces entre suas relações. Para ilustrar a operação de JOIN será executado o processo de recuperação de todos os atributos da relação ACADEMICO e EDUCADOR, onde o Id_educador de ambas as relações sejam iguais (DATE, 2003).

(ACADEMICO |×| EDUCADOR)

Quadro 8. Modelo de JOIN para combinar tuplas agrupadas

O modelo da operação do Quadro 8 retornará uma nova combinação dos atributos da relação ACADEMICO e EDUCADOR, conforme Tabela 2 abaixo:

Tabela 2. Modelo de junção de tabelas ACADEMICO e EDUCADOR

Id_Academico	Nome	Cidade	Id_Educador	E_Nome	E_Cidade
000001	Filipe	São Paulo	000009	Maria	Itajai
000002	Rafael	São Paulo	000010	Jose	Curitiba
000003	Jose	Curitiba	000007	Andréia	Blumenau
000004	Ralf	Blumenau	000008	Jussara	São Paulo
000005	Karine	São Paulo	000005	Pedro	São Paulo
000006	Carlos	São Paulo	000006	Fernando	Itajai
000007	Pietro	Curitiba	000003	Joel	Curitiba
000008	Maria	Itajaí	000004	Rafael	Curitiba
000009	Pedro	Itajaí	000002	Rubens	Itajai
000010	Cesar	Curitiba	000001	Marcelo	Blumenau

Fonte: Adaptado de Date (2003).

O resultado da Tabela 2 exige que exista uma combinação entre duas tabelas, denominadas ACADEMICO e EDUCADOR. Os relacionamentos são processados entre as junções combinando as tuplas de ambas as tabelas, resultando uma consulta com os valores que encaixam na comparação. O modelo utiliza a combinação da coluna “Id_Academico” para retornar os registros que combinam na condição de igualdade.

A operação CARTESIAN PRODUCT (produto cartesiano), denotada pelo símbolo × é uma operação de conjunto binária, mas as relações sobre as quais são aplicadas não necessitam ser provindas de uma união compatível (DATE, 2003). A operação retorna uma combinação de todas as relações de tuplas em questão. Possui como operadores fundamentais duas tabelas que resultam em uma nova tabela, sendo as linhas da Tabela A e Tabela B que quando combinadas geram o registro.

π Nome (σ Filial. Cod_Acessorio= Acessorio. Cod_Acessorio (Filial × Acessorio))

Quadro 9. Exemplo aplicado de CARTESIAN PRODUCT

Conforme descrito no Quadro 9, a operação de produto cartesiano resulta em uma nova tabela denominada “Tabela Resultante”, que agrupa um conjunto de alementos comuns entre dois ou mais conjuntos. O processo de intersecção na Figura 3 demonstra que quando agrupadas as tabelas “Acessorio” e “Filial” o retorno dos registros será feito por nome, com a condição de que a comparação entre o código “Cod_Acessorio” da tabela “Acessorio” for idêntico ao “Cod_Acessorio” da tabela “Filial”.

O cálculo é feito a partir do Nº total de colunas da Tabela <Acessorio> + o Nº total de colunas da tabela <Filial>. O total de colunas são 4 da tabela <Acessorio> somados com 3 da tabela <Filial> totalizando 7 colunas. O número total de linhas resultante é feito pelo produto cartesiano conforme Figura 3.

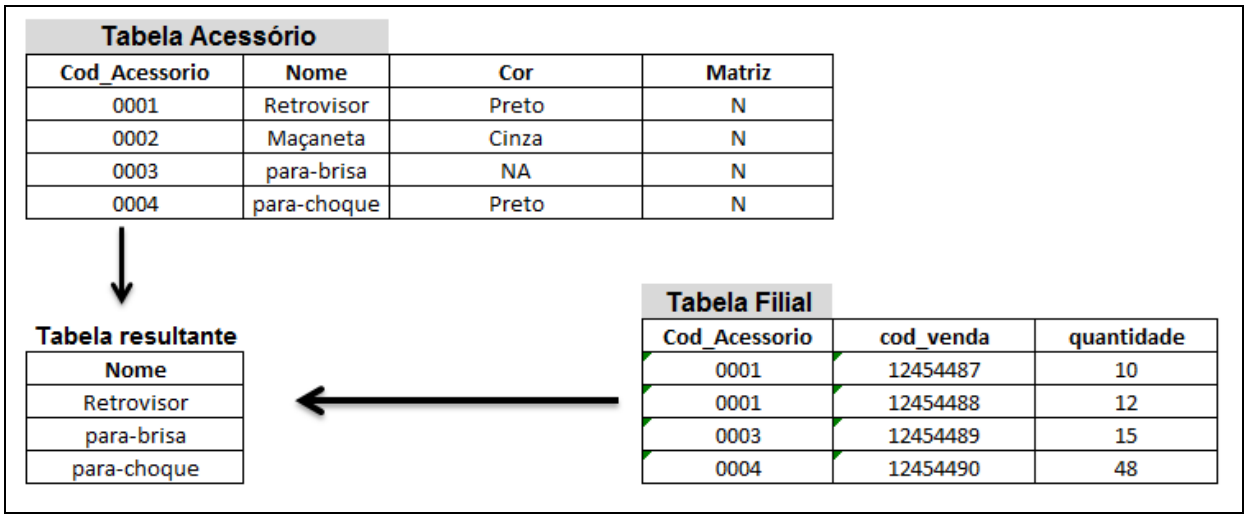


Figura 3. Exemplo de produto cartesiano

2.2 Método de otimização

A otimização de uma consulta SQL seleciona o processo de um plano de execução mais eficiente para a consulta. A real finalidade do otimizador de banco de dados é abstrair do usuário a complexidade da consulta e ser autônomo na escolha do melhor método. As técnicas para remodelar as consultas que desempenham a mesma funcionalidade, mas com tempo de execução menor que a consulta original é o fator determinante quando o assunto é otimização de banco de dados. O otimizador reduz o esforço manual e de alto custo para identificar, e caso necessário, corrigir comandos (analise léxica e sintática) do SQL (CAPELLO; CINTRA; MAGALHÃES, 2005).

Quando uma consulta é realizada no banco de dados, essa consulta é diluída em vários pedaços conhecidos como blocos de consulta. Depois de realizada a decomposição da consulta, o processamento é feito individualmente um a um sem ocorrer controle de concorrência. Quando se trata de blocos aninhados, esses são tratados como uma chamada de sub-rotina executada uma por vez pela tupla mais externa. No final esses blocos serão convertidos pela álgebra relacional (*ibidem*).

A representação do modelo relacional é feita por operações da teoria dos conjuntos como união, produto cartesiano, diferença e por algumas operações específicas de banco de dados relacionais como seleção, agregação e projeção.

Um bloco de consulta SQL não ordernado possui a cláusula SELECT que determina a relação dos atributos que representam os dados de retorno solicitados na consulta. Para a cláusula FROM, a responsabilidade é de relacionar todas as tabelas envolvidas na consulta. As condições de limitação e de especificação são feitas pela cláusua WHERE. A cláusula GROUP BY efetua o agrupamento das tuplas que possuem o mesmo valor, este é aplicado em funções de agregação. A cláusula HAVING específica à condição de agrupamento de tuplas da cláusula GROUP BY. A cláusula ORDER BY classifica os registros retornados pela consulta em ordem crescente ou decrescente. (*ibidem*).

Na Figura 4 será demonstrado um exemplo básico de consulta SQL. A consulta selecionará todos os registros da tabela “tb_parceiro” com a condição de que o “UF” seja exatamente igual a “SP”. Após a execução do select, o banco de dados agrupará o resultado por “UF” e “razão social” e ordenará pelo valor da “razão social”.

```
01.  select uf, razaosocial
02.  from tb_parceiro
03.  where uf = 'sp'
04.  group by uf, razaosocial
05.  order by razaosocial
```

Figura 4. Exemplo básico de Select

Na Seção abaixo é possível destacar as vantagens e desvantagens na utilização da heurística na obtenção da otimização da consulta.

2.2.1 Heurística na otimização

Uma das grandes desvantagens da otimização baseada em custo é o próprio custo da otimização. O custo da otimização da consulta pode ser reduzido inserindo algoritmos inteligentes, embora o número de planos para diversas avaliações de uma consulta SQL pode ser grande, desta forma, encontrar um plano ideal de redução de processamento requer um esforço computacional considerável (KORTH; SILBERSCHATZ; SUDARSHAN, 2006). No cenário da heurística, os otimizadores buscam utilizá-la para reduzir o custo da otimização.

A heurística trabalha diretamente com a álgebra relacional para transformar as consultas com o objetivo de realizar as operações de seleção o mais cedo possível.

Um otimizador de heurística utilizaria essa análise sem descobrir se o custo foi reduzido por essa transformação. Essa regra é considerada uma heurística, por que normalmente reduziria o custo na maioria dos casos (KORTH; SILBERSCHATZ; SUDARSHAN, 2006).

A operação de projeção, como a operação de seleção reduz o tamanho das relações. Na heurística é vantajoso realizar seleções antecedendo a projeção, pois no caso da seleção a maiores chances de obter-se a redução das relações (KORTH; SILBERSCHATZ; SUDARSHAN, 2006).

Os otimizadores de consulta mais práticos e dinâmicos como o *System R* foi um dos primeiros a implementar a linguagem SQL e tornar-se padrão das linguagens de consulta relacional na década de 70. Desenvolvido pela IBM, foi precursor do Sistema de Gerenciamento de Banco de Dados Relacional e demonstrou maior desempenho no processamento de transações. A arquitetura do *System R* e sua programação dinâmica utilizada na otimização de consulta serviram de base para muitos sistemas relacionais posteriores (*ibidem*).

O otimizador do *System R* considera apenas as ordens de junção em que o operando da direita de cada junção é considerado uma das relações iniciais *r1,... rn.* Essas ordens de junção são consideradas ordens de junção esquerda profunda (*ibidem*).

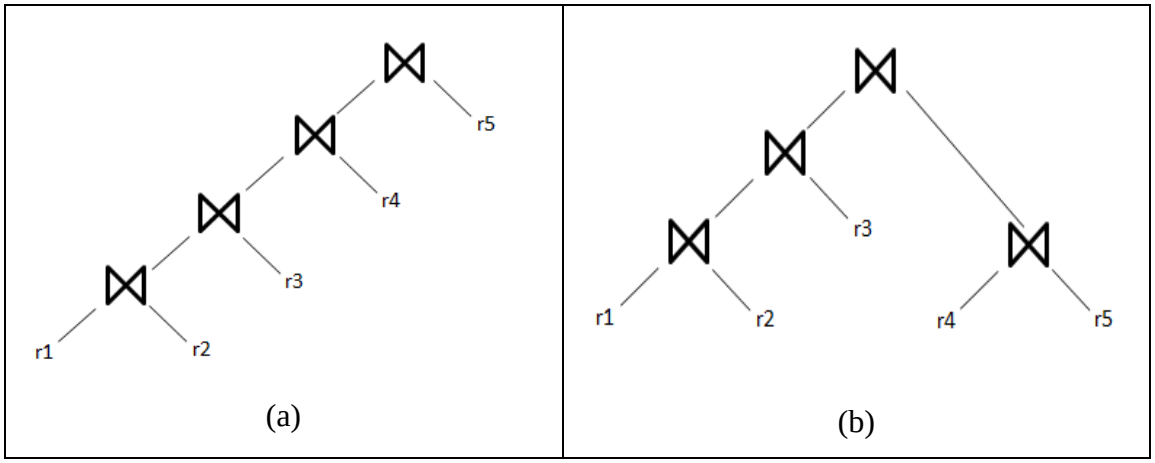


Figura 5. Árvore de junção: (a) junção profunda; (b) junção esquerda profunda.

No cenário de heurísticas, demasiadas aplicações executam a mesma consulta repetidamente, mas com valores de constantes de retorno diferente. Uma aplicação pode executar uma consulta para retornar transações recentes sobre uma determinada consulta, mas com valores diferentes. Muitos otimizadores potencializam individualmente cada bloco da

consulta. Sempre que a consulta é executada novamente, o plano de consulta armazenado na memória do banco de dados é reutilizado apenas utilizando novos valores (KORTH; SILBERSCHATZ; SUDARSHAN, 2006).

Durante uma consulta, mesmo com o uso da heurística a otimização baseada em custo impõe uma sobrecarga substancial no sistema de processamento da consulta. O esforço adicional da otimização da consulta baseada em custo é compensado pela redução do tempo de execução da consulta, que na maioria dos casos ocorre por acessos a discos de baixo desempenho (KORTH; SILBERSCHATZ; SUDARSHAN, 2006).

2.2.1.1 Algoritmo para Processamento e Otimização de consultas

Uma consulta expressa em uma determinada linguagem de consulta, tal como SQL, deve primeiro percorrer por um processo de análise léxica, análise sintática e por uma validação do banco de dados antes da execução (ELMASRI; NAVATHE, 2005).

A análise léxica ou *scanner* identifica e aponta os itens léxicos da linguagem escrita, tais como, palavras-chave do SQL, nomes de atributos e relacionamentos no texto da consulta. A análise sintática ou *parser* verifica a sintaxe da consulta, para determinar se ela foi escrita de acordo com as regras sintáticas do SGBD e são conhecidas como regras gramaticais da linguagem de consulta. A consulta também precisa ser validada por meio de que todos os atributos de entrada e nomes de relacionamentos sejam válidos e se possuem significados semânticos no esquema do SGBD (ELMASRI; NAVATHE, 2005).

O SGBD é responsável por planejar uma estratégia de execução para recuperar um determinado resultado, a partir de uma consulta nos arquivos do banco de dados. Uma consulta pode ser formada por vários métodos e estratégias possíveis, e o processo de escolha de uma estratégia adequada para o processamento da consulta é chamado de otimização de consulta conforme analisado na seção 2.2.1 (Heurística na otimização).

O plano de execução é na verdade uma denominação imprópria, pois a velocidade de processamento variará pelo tipo de consulta imposta. Encontrar a melhor estratégia demanda muito tempo, exceto para consultas mais simples que requerem menos comparações e níveis de profundidade reduzidos na árvore de consulta (ELMASRI; NAVATHE, 2005).

Na Figura 6 são demonstrados os diferentes passos do processamento de uma consulta em alto nível. O módulo de otimização de consulta tem o papel de produzir um plano de execução, e o gerador de códigos gerarem o código que executa aquele plano. O processador

em tempo de execução do banco de dados tem a função de executar o código da consulta, sendo ela no plano de compilação ou código interpretado, a fim de obter um resultado para consulta determinada. No tempo de execução, se resultar em um erro em tempo real, toda a mensagem gerada é feita pelo processador em tempo de compilação (*ibidem*).

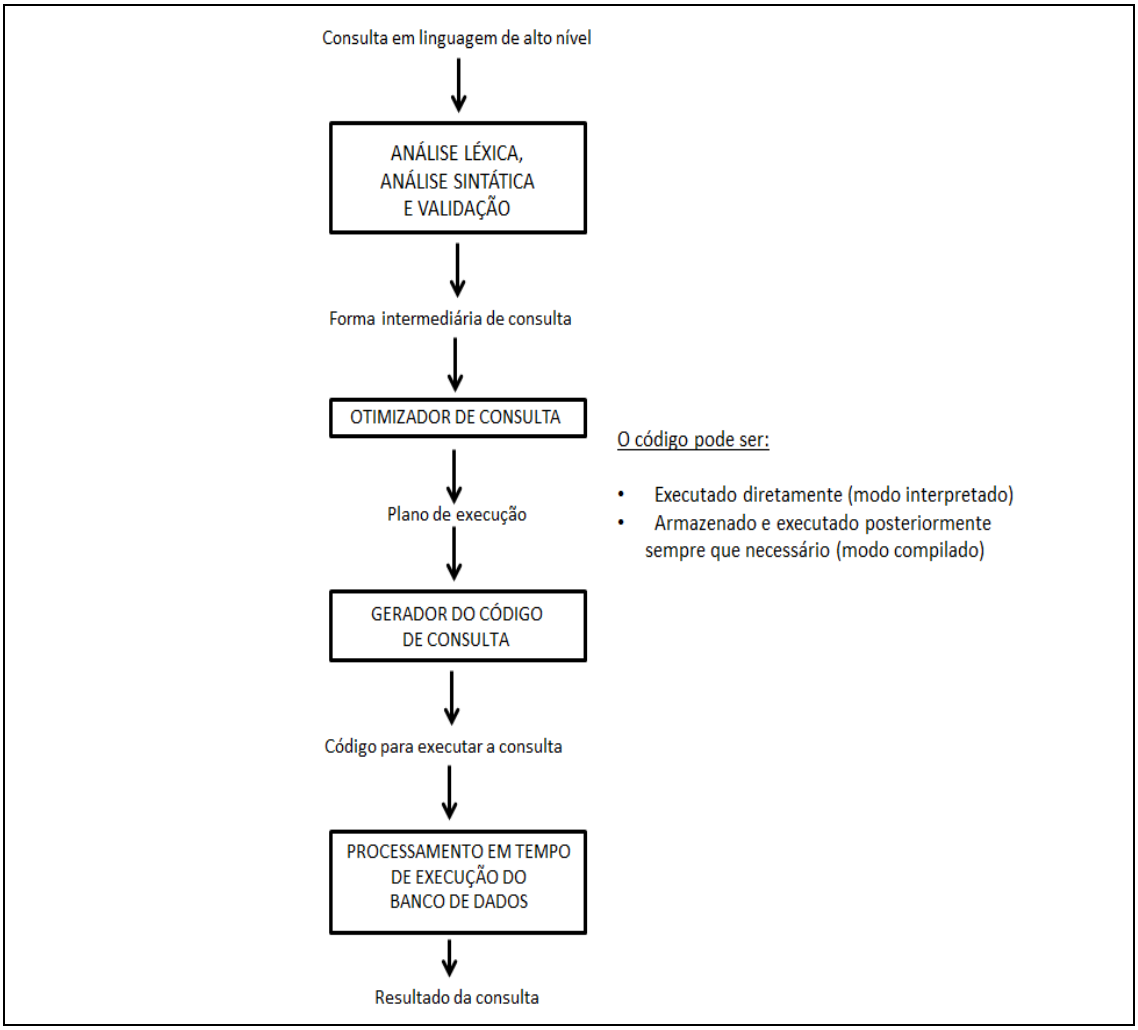


Figura 6. Passos típicos durante a execução de uma consulta de alto nível.
Fonte*:* Adaptado de Elmasri e Navathe (2006).

Um SGBD relacional deve avaliar de forma sistemática diversas estratégias de execução de consulta e optar pelo plano de execução que ele considera ótimo ou razoavelmente eficiente. De modo geral, o SGBD possui diversos algoritmos genéricos de acesso ao banco de dados capazes de implementar operações relacionais, tais como SELECT ou JOIN ou a combinações de ambas as operações. As estratégias de execução são as únicas que podem ser implementadas pelos algoritmos de acesso ao SGBD e que se aplicam a uma consulta específica ou a um projeto de banco de dados em particular podendo ser considerados pelo módulo de otimização de consulta (*ibidem*).

2.2.1.2 Estrutura do Otimizador de Consultas

Como a maioria dos sistemas computacionais implementam poucas estratégias, o número de estratégias consideradas pelo otimizador é limitado. Para cada estratégia escolhida pelo otimizador, é gerado um custo que é calculado com base na complexidade da consulta e na quantidade de estratégias para resolução (KORTH; SILBERCHATZ, 1995).

Alguns SGBDs reduzem o número de estratégias que precisam ser levadas em consideração fazendo um processo conhecido como estimativa heurística de um plano considerado bom. Baseado nessa linha de escolha o otimizador considera cada estratégia possível, mas tão logo ele verifica se o custo de cada estratégia analisada é melhor ou igual a melhor estratégia considerada anteriormente (KORTH; SILBERCHATZ, 1995).

Uma consulta pode ser dividida em várias subconsultas simplificando a seleção da estratégia e permitindo ao otimizador reconhecer casos em que uma determinada subconsulta venha aparecer diversas vezes. O reconhecimento de subconsultas iguais é análogo ao reconhecimento de subexpressões iguais em diversos compiladores com otimização baseados na linguagem de programação (KORTH; SILBERCHATZ, 1995).

O exame de uma consulta em busca de subconsultas iguais e a estimativa de custo de um grande número de estratégias impõe certa sobrecarga de trabalho no processamento das consultas. No entanto, o custo adicional da otimização de consulta é considerado compensatório pela economia no tempo de execução da consulta (*ibidem*).

2.2.1.3 Complexidade no Desempenho de Algoritmos

Um algoritmo consiste de um conjunto de operações e regras não ambíguas, as quais especificam que para cada entrada é definida por uma sequência finita de operações que determinará uma saída correspondente (TOSCANI; VELOSO, 2008).

O propósito de um algoritmo é resolver um problema que quando, para determinada entrada, produza uma saida correta (se disponíveis tempo e memória suficiente para sua execução). O fato de um algoritmo resolver um determinado problema não significa que ele seja aceitável na prática. Os recursos disponíveis como espaço e memória são determinantes no processo de desempenho em casos práticos (TOSCANI; VELOSO, 2002).

O algoritmo mais simples está distante de ser razoável em termos de eficiência. A exemplo disso está o caso da solução de sistemas não lineares. O método de *Cramer*, calculando o determinante através de sua definição, requer milhões de anos para resolver um

sistema de uma matriz 20x20. Nesse caso, o sistema poderia ter sido resolvido através do método de Gauss em um tempo razoável (*ibidem*).

A Tabela 3 apresenta o desempenho desses algoritmos que efetuam o cálculo determinante de uma matriz n x n, baseado em tempo de operação de um computador real.

Tabela 3. Tamanho do Problema x Tempo de execução

N	Método de Cramer	Método de Gauss
2	22 μ	50 μ
3	102 μ	159 μ
4	456 μ	353 μ
5	2.35 ms	666 μ
10	1.19 min	4.95 ms
20	15.225 séculos	38.63 ms
40	5x10^33 séculos	0.315 s

Fonte*:* Adaptado de Toscani e Veloso (2002).

Com o crescente avanço tecnológico, permitiu a criação de máquinas com alto poder de processamento. As máquinas se tornando mais rápidas passam a resolver algoritmos e problemas de maior escala e complexos e é a complexidade do algoritmo que determina o tamanho máximo do problema resolvível. Para um algoritmo rápido, quaisquer alterações são consideráveis, e o conjunto de problemas resolvíveis por ele aumenta significativamente. Em consideração a algoritmos mais complexos essa diferença de desempenho é menor (*ibidem*).

A análise da complexidade de um algoritmo é realizada de maneira muito particular, pois a complexidade possui medidas que tem parametros específicos do algoritmo. Alguns estudos mais aprimorados permitem prever alguns aspectos da complexidade do algoritmo resultante (*ibidem*).

O objetivo da análise de algoritmos é melhorar, se possível, seu desempenho e optar, entre determinados algoritmos disponíveis o melhor para determinado caso. Existem determinados critérios de avaliação de um algoritmo como: simplicidade, exatidão de resposta, quantidade de trabalho requerido e otimalidade (*ibidem*).

2.2.1.4 Medidas de Complexidade

O esforço requerido por um algoritmo não pode ser descrito simplesmente por um número, pois o número de operações básicas efetuadas, de modo geral, não é o mesmo e varia de acordo com o tamanho de entrada (TOSCANI; VELOSO, 2002; SIPSER, 2007; ARORA; BARAK, 2009).

No projeto de um algoritmo, a complexidade deve ser considerada como um fator integrante do processo de construção do algoritmo (TOSCANI; VELOSO, 2002; ZIVIANI, 2007). Nesse aspecto, o projetista precisará tomar a decisão de optar por qual algoritmo utilizar, estudando diversas opções e analisando aspectos de tempo de execução, espaço ocupado e demais considerações importantes sobre a aplicação (ZIVIANI, 2007).

Na análise de algoritmos, existem dois tipos de problemas bem distintos (ZIVIANI, 2007):

- Análise de um algoritmo particular (complexidade do algoritmo): Qual é o custo computacional para resolver determinado problema? Neste caso, são analisadas diversas características. Faz-se necessária uma análise de quantas vezes cada parte do algoritmo será executada, seguida do fator de memória necessária;
- Análise de uma classe de algoritmos (complexidade do problema) busca o algoritmo de menor custo possível para resolver determinado problema específico.

Quando possível determinar o menor custo possível para resolver problemas de determinada classe é encontrada a dificuldade inerente para resolver tais problemas. Quando um algoritmo é igual ao menor custo possível, pode-se concluir que o algoritmo é considerado ótimo para a medida de custo considerada (ZIVIANI, 2007).

Existem diversas maneiras de medir o custo de utilização de um algoritmo. Uma delas é medir o tempo de execução e o espaço requerido pelo programa em um computador real, sendo a execução medida diretamente. As medidas de tempo extraídas desta forma são muito inadequadas e os resultados jamais são generalizados. Os resultados variam muito de acordo com o hardware utilizado, com a quantidade de trabalho sendo executado em segundo plano por outros programas e entre outros fatores (*ibidem*).

Uma forma adequada para medir o custo de um algoritmo é por meio da utilização de um modelo matemático baseado em um computador idealizado, como, o computador MIX proposto pelo professor da Universidade de STANDFORD em 1968. É importante ignorar as operações de menor relevância e considerar apenas as mais significativas (*ibidem*).

Segundo Toscani e Veloso (2002, p. 9), “A expressão quantidade de trabalho requerido também é chamada de complexidade do algoritmo”. O desempenho do algoritmo é diretamente proporcional a sua saída, pois quanto maior a entrada de dados para processar,

maior será o tempo de execução do algoritmo. O tamanho de um algoritmo de entrada *n* afetará no tempo de resposta, que poderá ser incisivo ou depreciativo.

Quando se trata de medida de tempo, a análise de algoritmos assimila o número de operações que são consideradas relevantes para processar a consulta e reflete esse valor como uma função com a variável *n*, que indica o tamanho da entrada. As comparações efetuadas variam de operações aritméticas, comparações, organização vetorial e funções lógicas. De maneira geral, o pior caso ou complexidade pessimista possui o maior número de operações utilizadas para qualquer entrada de dados. A complexidade pessimista será abordada na próxima seção.

Na técnica de operação fundamental, selecionam-se as instruções mais importantes para o código (geralmente a quantidade de execuções destas instruções está relacionada ao tamanho da entrada), e para definir a complexidade, cria-se uma função que dada à entrada resulte na quantidade de vezes que as operações fundamentais são executadas.

Há três perspectivas de desempenho utilizadas para mensurar a complexidade de um algoritmo determinado. Todas estas três perspectivas não se diferenciam pelo volume da entrada de um problema, mas pela forma na qual a entrada está configurada. Por exemplo: alguns algoritmos de ordenação são mais eficientes se um vetor está parcialmente ordenado, esta configuração de entrada diminui a quantidade de operações necessárias para realização da tarefa. De acordo com Toscani e Veloso (2002) são perspectivas de desempenho:

- Complexidade otimista (melhor caso) é medida de acordo com o "menor esforço necessário", ou seja, a configuração de entrada na qual demandará menos tempo de execução;
- Complexidade pessimista (pior caso) mede a complexidade de acordo com o "maior esforço necessário", ou seja, a configuração de entrada na qual demandará mais tempo de execução;
- Complexidade média (caso médio) mede o esforço necessário levando em consideração as probabilidades de cada uma das possíveis configurações de entrada, trabalhando com uma média ponderada. O resultado está sempre relacionado à complexidade média dos casos.

2.2.1.5 Ordens Assintóticas

As ordens assintóticas no contexto de complexidade são utilizadas para auxiliar a representação das funções de complexidade de acordo com sua ordem de crescimento.

Na complexidade de algoritmos existem várias comparações de ordem assintótica que podem ser definidas, porém as mais utilizadas são: O, Θ e Ω. Essas ordens comparam o comportamento assintótico das funções (TOSCANI; VELOSO, 2002).

A notação O (ômicron) define uma cota assintótica superior. A exemplo disso pode-se exemplificar a função quadrática "$g(n) = n^2$" que cresce mais rapidamente do que a linear "$f(n) = 7n + 20$" (a partir de determinado ponto). Desta forma, é possível afirmar que "f(n) é O(g(n))".

A notação Θ define um limite assintótico exato. Em funções quadráticas é possível definir que: "$f(n) = 7n^2 + 20$" e "$g(n) = n^2 + 3$". Logo f(n) é Θ (g(n)) e g(n) é Θ(f(n)), onde ambas as funções possuem o mesmo ritmo de crescimento assintótico.

A notação Ω define uma cota assintótica inferior. A função cúbica "$g(n) = 7n^3 + 5$" possui ritmo de crescimento menor do que comparado à função exponencial "$f(n) = 2^n$" (a partir de determinado ponto), ou seja, nesse caso, é possível afirmar que a exponencial f(n) é Ω (g(n)) (TOSCANI; VELOSO, 2002).

2.2.1.6 Análise de Complexidade Pessimista

Dentre todas as perspectivas, a complexidade no pior caso é o critério de avaliação mais utilizado, pois visa representar o pior comportamento de um determinado algoritmo, ou seja, a configuração de entrada que levará mais tempo na execução A metodologia para calcular a complexidade pessimista é baseada em estruturas algorítmicas (TOSCANI e VELOSO, 2002).

A soma total do desempenho de um algoritmo é o conjunto da soma de todas as suas partes executadas. De modo geral, a complexidade de um algoritmo poderá ser descoberta a partir da complexidade de suas partes (TOSCANI; VELOSO, 2002).

De acordo com Toscani e Veloso (2002), há algumas técnicas que auxiliam no cálculo da complexidade pessimista de um algoritmo: complexidades conjuntivas e disjuntivas. Em complexidades conjuntivas, todas as instruções são executadas sempre, em conjunto. Um exemplo de algoritmo conjuntivo é um algoritmo que determine o valor mínimo e máximo de uma sequência de números (Figura 7).

Um exemplo de algoritmo conjuntivo é demonstrado na Figura 7, que tem o objetivo encontrar o menor valor de um determinado vetor. Para encontrar o menor valor é criada uma variável conhecida como "menor" que recebe um valor inicial. Quando percorrido os índices do vetor é verificado se o valor atual do vetor é menor que o tamanho da variável "menor". Caso o valor da posição do vetor seja inferior a variável "menor", ela assumirá esse valor. O processo é finalizado quando o algoritmo percorrer todas as posições do vetor e o atual valor da variável "menor" seja o menor valor do vetor. O processo para encontrar o maior valor do vetor é o mesmo, porém a comparação verifica todas as posições do vetor até encontrar o maior valor. Quando encontrado o maior valor a variável "maior" assumirá esse valor.

```
void min_max ( double vetor[] ){

      //encontra menor valor
      double menor = min (vetor);

      //encontra maior valor
      double maior = max (vetor);

      //imprime o menor e maior valor
      cout << "Menor é: " << menor;
      cout << "Maior é: " << maior;

}
```

Figura 7. Determinar o MIN e MAX de uma sequência de números.
Fonte*:* Adaptado de Toscani e Veloso (2002).

Quanto ao algoritmo descrito na Figura 7, para calcular sua complexidade é necessário analisar as funções “min” e “max” que são sempre executadas pelo algoritmo. Cada uma delas possui a complexidade pessimista de "f(n) = n - 1", pois no pior caso será percorrido todas as posições do vetor, logo, para um vetor de dez elementos, a quantidade de operações fundamentais executadas seriam nove. Desta forma, pode-se afirmar que cada uma das funções é importante e são consideradas como operações fundamentais. No caso, tem-se um caso de algoritmo conjuntivo.

Então a complexidade dada pela soma das complexidades é:

$C_P[min](n) + C_P[max](n)$, logo:
$C_P[min\_max](n) = 2.(n-1)$, ou seja, é Θ (n)

Quadro 10. Soma das complexidades do algoritmo

Conforme Figura 8, possui um algoritmo de operações fundamentais disjuntivas, ou seja, quando uma é executada, a outra não. Deste modo, a complexidade do algoritmo pessimista passa a considerar que a função com maior comportamento assintótico, com maior crescimento, é a que representará a complexidade do algoritmo. No exemplo o algoritmo que recebe uma sequência não nula de números. A sua ordenação dependerá do valor do seu primeiro elemento que caso determinada condição, poderá efetuar a operação de soma ou de ordenação, conforme Figura 8.

Logo, a complexidade é dada pelo máximo entre:

C_P [ordenação] (n) C_P [somatório] (n), logo:
C_P [ordena_somatório] (n) = Máx $\{n^2, n\} = n^2$, ou seja, é $\Theta (n^2)$

```
double ordena_somatorio ( double &vetor[] ){
      double ret = 0.0;
      if (vetor[0] > 0){
            ordena_vetor(vetor);
            ret = vetor[0];
      }else{
            ret = somatorio(vetor);
      }
      return ret;
}
```

Figura 8. (Disjuntiva): ordenar ou realizar seu somatório.
Fonte*:* Adaptado de Toscani e Veloso (2002).

2.2.2 Exemplo de Pesquisa Sequencial de Vetores

Para mensurar a quantidade de trabalho de um determinado algoritmo, é escolhido como determinante uma operação chamada operação fundamental, e então é iniciada uma contagem do número de execuções que uma operação é executada no algoritmo. A principal operação deve ser escolhida como fundamental, onde o número de vezes que é executada expresse a quantidade de trabalho do algoritmo podendo dispensar as outras medidas impostas (SANTOS, 2009).

A pesquisa sequêncial de vetores é um método simples para uma determinada estrutura de dados. Esse método pode ser utilizado com um vetor unidimensional ordenado ou não, mas é muito utilizado quando os registros não estão ordenados. A pesquisa inicia no primeiro índice do vetor e avança sequencialmente passando por todos os índices. O resultado

da busca termina quando uma condição for satisfeita, determinada por uma condição imposta pelo algoritmo (SANTOS, 2009).

```
int busca_sequencial (int chave, int vetor[n]){
for (i = 0; i < n; i++){
    if(vetor[i] == chave){
return i;
  }
}
return -1;
}
```

Quadro 11. Pesquisa sequência de vetores
Fonte: Santos (2009).

Outro exemplo de cálculo de complexidade pessimista pode ser dado por um exemplo de busca linear em um vetor. No Quadro 4, onde o vetor A[7] possui 7 posições (dados como entrada). O algoritmo percorrerá o vetor buscando um valor chave que seja igual ao solicitado. Satisfeita estas condições, será retornada a posição do valor (índice no vetor). É possível afirmar que a complexidade do algoritmo é O(n), sendo "n" o tamanho do vetor, pois no pior caso se encontrará a chave na última posição.

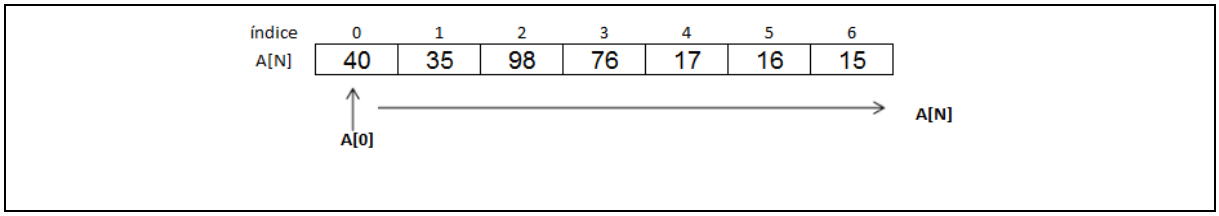


Figura 9. Busca linear em vetor
Fonte*:* Adaptado de Santos (2009).

Conforme descrito na Figura 9 mostra como é percorrido um vetor unidimensional de sete posições, onde inicia na primeira posição a esquerda e percorre o vetor até a penúltima posição, N-1.

2.2.2.1 Complexidade Média

Dentre as medidas de complexidade, a complexidade média é, em muitos casos, a mais apropriada de ser medida devido ao uso mais frequente do algoritmo. A complexidade

média de um algoritmo é efetuada a partir de um cálculo da média ponderada do desempenho de cada configuração de entrada pela probabilidade de ela ocorrer (TOSCANI; VELOSO, 2002).

O cálculo da complexidade média é muito semelhante a uma média ponderada. Pode-se representá-lo através da Equação 1 (TOSCANI; VELOSO, 2002):

Onde:

S – Todas as configurações de entrada possíveis, ou seja, situações com complexidade diferenciada.

n – tamanho da entrada.

P(i) – probabilidade de determinada situação (configuração de entrada) "i", ocorrer.

$$\sum_{i=1}^{s} \left[p(i) \times c_i(n) \right]$$

Equação 1

Na Figura 9 é possível verificar que a sua complexidade média é medida através da principal operação fundamental, onde a chave possui o valor que é procurado no vetor, retornando sua posição. A busca por uma determinada chave em um vetor desordenado implica em uma busca sequencial. Deste modo, pode-se afirmar que para uma entrada de tamanho cinco, têm-se cinco casos diferentes.

1. Encontrar o valor no primeiro elemento: executando apenas uma vez a operação;
2. Encontrar o valor no segundo elemento: executando apenas uma vez a operação;
3. Encontrar o valor no terceiro elemento: executando apenas uma vez a operação;
4. Encontrar o valor no quarto elemento: executando apenas uma vez a operação;
5. Encontrar o valor no quinto elemento: executando apenas uma vez a operação;

A complexidade para esses casos é somada a partir da quantidade de execuções. Complexidade [1,2,3,4,5]. Para casos onde a entrada é desconhecida, a quantidade de execuções será n e a complexidade [1,2,3,4...n] (TOSCANI; VELOSO, 2002).

Logo, a fórmula reduzida pode ser vista na Equação 2, onde, aplicadas propriedades matemáticas foi possível simplificar a fórmula original e tornando-a mais fácil, menor e equivalente.

Onde, C_M[busca_sequencial](n) é definido pelo número total de entradas. A fórmula (n + 1)/2 soma o total de elementos de entrada mais um e divide esse valor por dois, encontrando o ponto médio do vetor, independente do vetor estar ordenado ou não.

$$C_M\left[busca\_sequecial\right](n) = \sum_{i=1}^{n}\left[\frac{1}{n}\times i\right] = \frac{n+1}{2} \;\; é \;\; \Theta(n) \qquad \text{Equação 2}$$

2.2.2.2 Seção Teorema Mestre

Para entender o método do Teorema Mestre é necessário absorver como o método de divisão e conquista é sintetizada. Dividir o problema que será resolvido em diversas partes menores e buscar encontrar soluções para os fragmentos que foram divididos e só então combinar os resultados obtidos em uma única solução (ZIVIANI, 2007).

O objetivo da utilização desse paradigma é segmentar o problema em diversos subproblemas, dos quais são versões menores do problema original, levando a soluções mais elegantes e eficientes e principalmente quando utilizado o método de recursividade. A técnica de divisão e conquista tem por finalidade resolver problemas de ordenação de um determinando conjunto de elementos (ZIVIANI, 2007).

$$T(n) = aT(n/b) + f(n)$$

Equação 3

Na Equação 3, o valor da constante “a” deve ser maior ou igual a 1 $a \geq 1$ e o valor da constante “b” obrigatoriamente deve ser maior que 1, b> 1. Para a função “f(n)”, essa é uma função assintóticamente positiva (descreve como o tempo de execução cresce à medida que a entrada aumenta). A Equação 3 descreve a recorrência do tempo de execução de um algoritmo.

Processos do algoritmo:

- Dividir o problema de entrada “n” em diversos subproblemas, sendo cada um com tamanho n/b;
- Resolver cada subrotina de forma recursiva;
- Combinar os resultados dos subproblemas em uma solução global do problema original em f(n) operações executadas.

Conforme descrito na Equação 4, um exemplo da aplicação do teorema mestre:

$$T(n) = 9T(n/3) + n$$

Equação 4

Onde se assume que: a = 9, b = 3 e f (n) = n

Processos do cálculo:

$n^{\log_b a} = n^{\log_3 9} = n^2$ portanto, pode-se assumir que obtem-se:

$f(n) \in O(n^{\log_b a - \varepsilon})$ assumindo que $\varepsilon = 1$.

$T(n) = \Theta(n^{\log_b} \log n)$, se $f(n) = \Theta(n^{\log_b})$,

$T(n) = \Theta(f(n))$, se $f(n) = \Omega(n^{\log_b a+ \varepsilon})$ para $\varepsilon > 0$.

Depois de efetuadas as substituições, é possível concluir que:

$T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$

3 DESENVOLVIMENTO

O projeto de desenvolvimento foi baseado na análise de Complexidade de Algoritmos sob o SGBD PostgreSQL. O estudo realizado sob o Código-Fonte tem por objetivo encontrar a forma como a consulta realizada no SGBD é estruturada e mensurar o esforço computacional implícito, extraindo a sua complexidade.

Nesta seção é documentado o processo de desenvolvimento da ferramenta proposta para este trabalho de TTC II. Nesta etapa, também será discutido o processo de serialização através do JSON (Javascript Object Notation) e a razão de ser utilizado para auxiliar na busca da árvore do plano de execução. A seção de desenvolvimento é divida em: projeto, resultados, ferramenta e testes.

3.1 Projeto

A finalidade desta seção é descrever o processo de construção da ferramenta, partindo da análise das cláusulas SQL, desenvolvimento, testes e análise de resultados.

3.1.1 Algoritmos de Consulta do PostgreSQL

Os algoritmos de consulta foram pré-selecionados durante o planejamento do TTC I. As cláusulas de seleção que foram utilizadas no contexto deste trabalho são: SELECT, FROM, WHERE, GROUP BY / HAVING e ORDER BY.

Na Figura 10, encontra-se descrita a estrutura da cláusula SELECT.

```
SELECT [ ALL | DISTINCT [ ON ( expressão [, ...] ) ] ]
    * | expressão [ AS nome_de_saída ] [, ...]
    [ FROM item_do_from [, ...] ]
    [ WHERE condição ]
    [ GROUP BY expressão [, ...] ]
    [ HAVING condição [, ...] ]
    [ { UNION | INTERSECT | EXCEPT } [ ALL ] seleção ]
    [ ORDER BY expressão [ ASC | DESC | USING operador ] [, ...] ]
```

Figura 10. Cláusulas da estrutura SELECT

Fonte: PostgreSQL (2012).

O comando SELECT retorna linhas de uma ou mais tabelas. O método de processamento do comando SELECT pode ser observado:

- Quando requisitada uma consulta no banco de dados, todos os elementos de uma lista FROM são computados, caso nenhuma outra condição seja implícita;
- Quando especificada uma cláusula WHERE, todas as linhas que não satisfazem essa condição são expurgadas do resultado de saída, retornando apenas as que satisfazem a condição descrita. Essas condições são tratadas de forma booleana (verdadeiro ou falso);
- Quando especificada a cláusula GROUP BY, ela determinará um conjunto de linhas selecionadas em um determinado conjunto de linhas que corresponderão a um ou diversos agrupamentos;
- Quando solicitada uma ordenação, utiliza-se a cláusula ORDER BY que ordena os registros do resultado e classifica de acordo com a condição da consulta definida.

As cláusulas descritas fizeram parte do corpo do projeto. Na estrutura, as queries foram analisadas de forma a obter-se a sua complexidade de modo pessimista (pior caso). Na estruturação das consultas, as cláusulas SQL foram analisadas de forma isolada buscando os resultados precisos dos métodos de processamento das consultas. Na próxima seção será destacada a estruturação e a formação de cada cláusula SQL.

3.1.2 Estrutura das Queries do PostgreSQL

A estrutura do trabalho está baseada no contexto das cláusulas SQL do SGBD PostgreSQL. As cláusulas foram analisadas a partir do Código-Fonte e sua Complexidade de Algoritmo mensurada. A versão do SGBD PostgreSQL analisado é a 9.1.3, atual versão da iniciação do projeto.

A forma como as cláusulas do PostgreSQL estão organizadas é de forma sistemática e complexa, pois as consultas estão subdivididas em diversos arquivos e chamadas de biblioteca com funções específicas. A utilização de cada cláusula (SELECT, FROM, WHERE, HAVING, GROUP BY e ORDER BY) tem variação de acordo com a consulta estipulada. O acesso de cada consulta passa por um processo de validação durante sua execução e fica a cargo do SGBD decidir o melhor plano de execução sempre buscando o menor caminho, resultando no melhor plano.

As próximas seções explicam detalhadamente o processo das consultas e as suas complexidades definidas. As figuras foram construídas com base na análise do Código-Fonte do PostgreSQL, onde também foi possível extrair a complexidade das instruções.

3.1.3 SELECT e FROM

As cláusulas SELECT e FROM possuem uma estrutura de arquivo conhecido como "nodeHashjoin.c", onde todas as funções estão disponíveis e todo processo de formalização da consulta é estruturado e implementa o algoritmo de Hash Join híbrido. O arquivo pode ser encontrado dentro de "postgresql-9.1.3\postgresql-9.1.3\src\backend\executor\nodeHashjoin".

Na Figura 11 é possível analisar a seletividade da cláusula SELECT. A cláusula seleciona todos os registros de determinada tabela que é definida na consulta. Todos os registros são selecionados sem restrição.

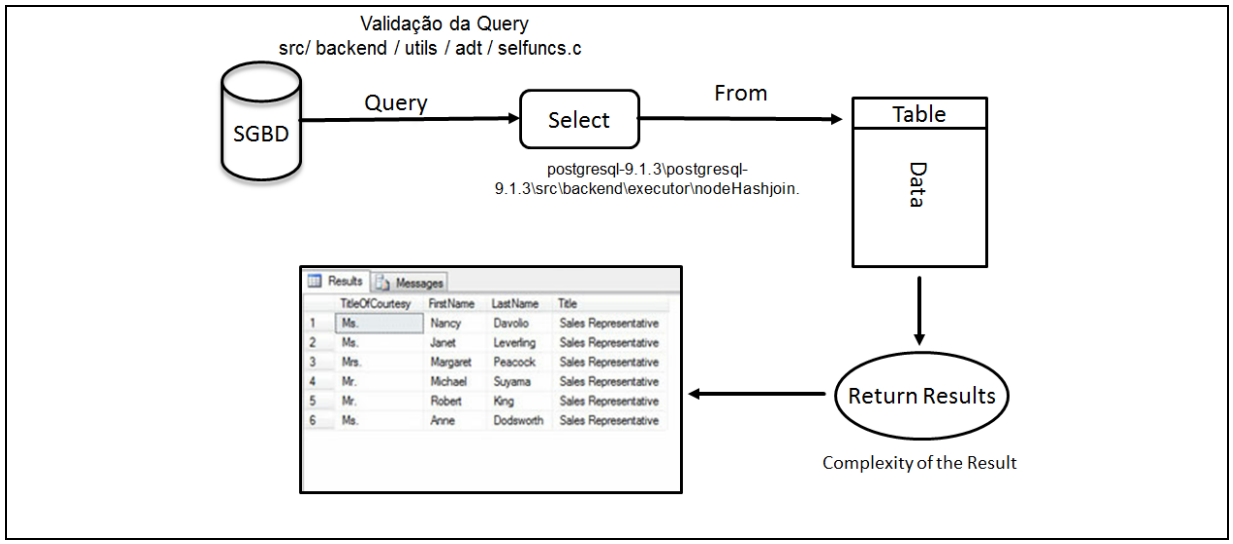


Figura 11. Lógica de cálculo para seletividade da cláusula SELECT

A Complexidade do Algoritmo presente é O(N), pois não é possível mensurar a quantidade de registros internos na tabela. Todos os registros são retornados sem exceção.

3.1.4 Cláusula GROUP BY

A cláusula GROUP BY pertence à estrutura Backend/executor assim como a cláusula SELECT e FROM. Seu modelo possui um nó projetado para lidar com as consultas. Seu plano exterior deve entregar tuplas que são classificadas na ordem especificadas pelas colunas de agrupamento (tuplas consecutivas). Desta forma a única necessidade é comparar as tuplas

adjacentes para localizar os limites de grupo. O objetivo é retornar uma tupla para cada grupo de tuplas de entrada correspondentes. A cláusula HAVING pertence ao mesmo escopo de execução do GROUP BY onde é utilizada para filtrar as linhas de um determinado resultado agrupado. Ao executar o comando GROUP BY em uma consulta SQL a chamada da função verifica de existe a cláusula HAVING para executar, caso não, a mesma é ignorada e o SELECT retorna ao loop até finalizar a busca. O arquivo de execução pode ser encontrado dentro do diretório "postgresql-9.1.3\postgresql-9.1.3\src\backend\executor" e é denominado "nodeGroup.c".

A condição da cláusula GROUP BY pode acompanhar a cláusula WHERE, como pode ocorrer na ausência da mesma. A complexidade do algoritmo sofrerá variação quando acompanhada e a complexidade aumentará variando de acordo com as condições estabelecidas na consulta.

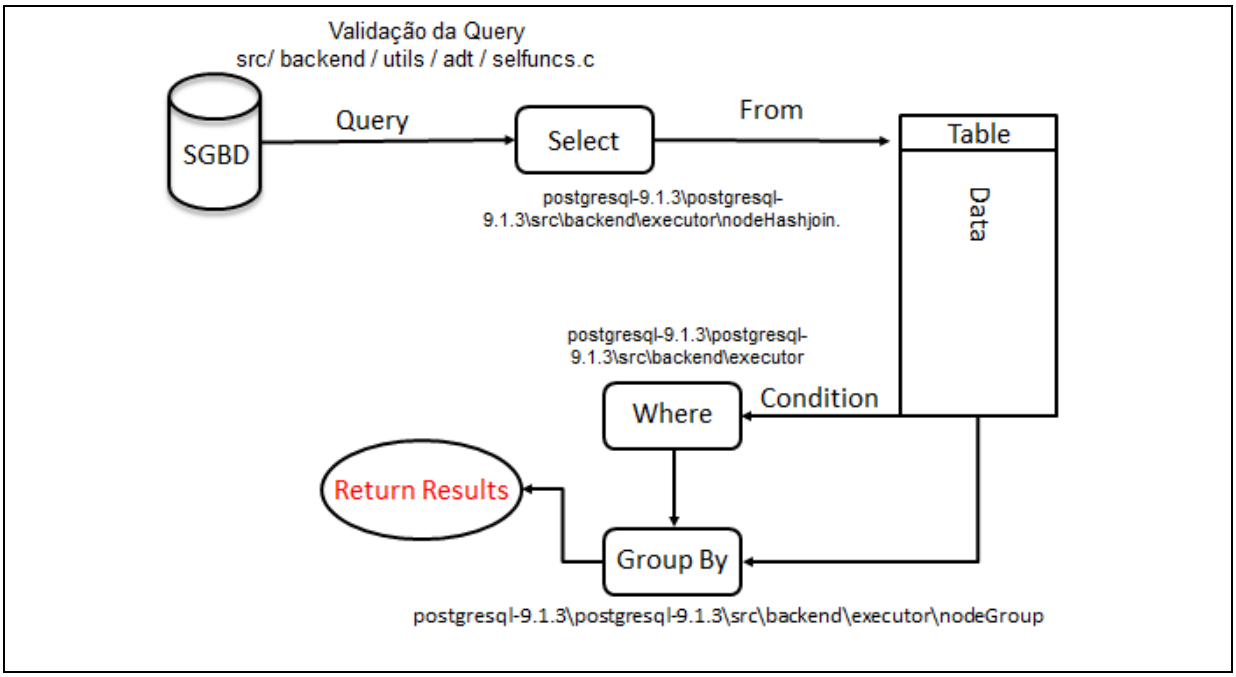


Figura 12. Lógica de cálculo para seletividade da cláusula GROUP BY

Na Figura 12 é possível acompanhar o processo efetuado em uma consulta, assim como descrito no parágrafo anterior. Sua complexidade é definida por O(n) muito similar com a cláusula ORDER BY. Seu método de agrupamento é feito através de "HashTable" (estrutura de alto desempenho na extração de informações) e utilização de índices.

3.1.5 Cláusula ORDER BY

A cláusula ORDER BY trabalha com diversas funções e chamadas de procedimentos em um mesmo arquivo denominado “nodeIndexscan.c”, encontrado dentro do diretório “postgresql-9.1.3\postgresql-9.1.3\src\backend\executor”. Essa cláusula trabalha com operadores simples, com valor de comparação constante "indexkey op constant" conforme instrução desenvolvida na linha 636 do arquivo de execução. O operador simples com uma entrada não constante "indexkey op expression" cria uma chamada “ScanKey” com todos os valores preenchidos, exceto o valor da expressão e cria-se uma estrutura para conduzir a “IndexRuntimeKeyInfo” que é responsável pela avaliação da expressão nos momentos corretos e as seleciona, conforme Figura 13.

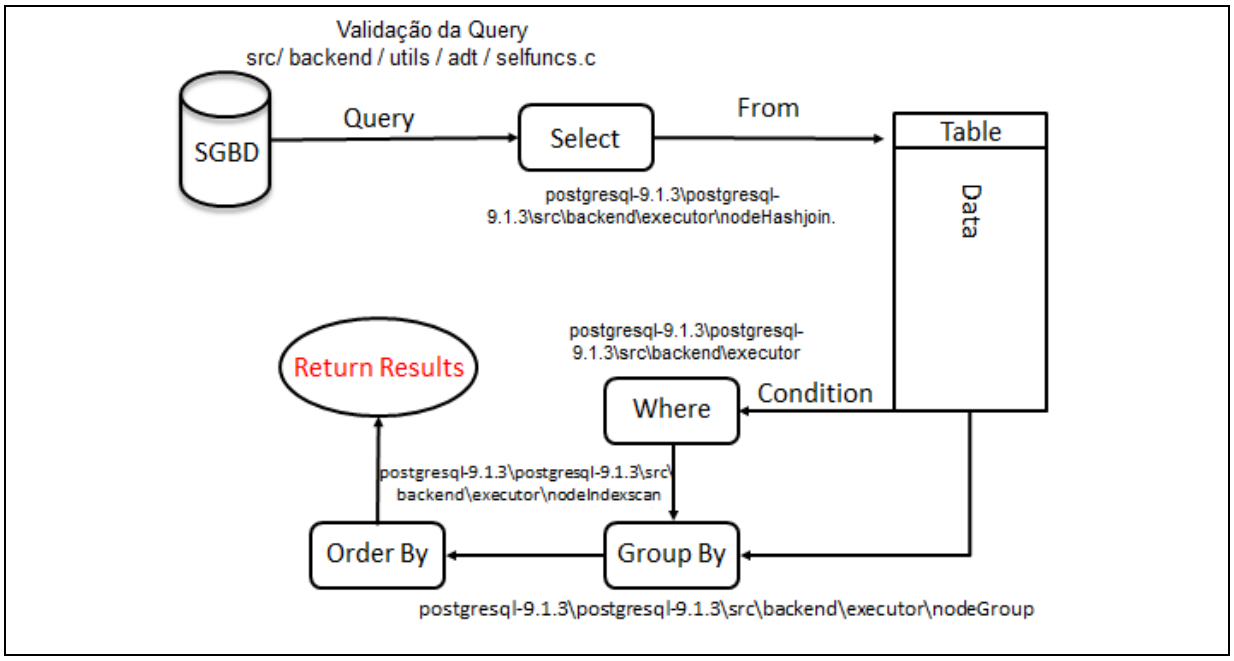


Figura 13. Lógica de cálculo para seletividade da cláusula ORDER BY

A complexidade da cláusula ORDER BY é definida por $O(n^2)$ devido à ordenação seguir a métrica de ordenação do algoritmo QuickSort. A soma da complexidade da cláusula SELECT + ORDER BY gera a complexidade quadrática.

3.1.6 Cláusula WHERE

A cláusula WHERE determina uma condição de pesquisa, onde a condição de pesquisa é qualquer expressão de valor e retorna um valor do tipo BOOLEANO (Verdadeiro ou falso). Quando definido um plano de consulta SELECT a cláusula FROM delimita de qual tabela os dados serão extraídos, nesse momento quando definida a utilização da cláusula

WHERE o banco de dados já busca os algoritmos de otimização e delimita os valores que se encaixam na condição marcando-os como verdadeiro. O plano de execução da cláusula pode ser encontrado no diretório "postgresql-9.1.3\postgresql-9.1.3\src\backend\executor" no arquivo denominado "execQual.c". A cláusula WHERE pode especificar duas condições:

- "restriction condition" compara uma expressão que faz referência a um valor de uma determinada coluna, sendo uma expressão constante ou uma coluna em uma mesma tabela do banco de dados;
- "join condition" compara uma expressão que faz referência ao valor de uma coluna em uma tabela para uma expressão que faz referência ao valor de uma coluna de outra tabela.

Na Figura 14 são apresentados os passos realizados na cláusula WHERE. O primeiro passo é selecionar todos os registros da tabela informada durante a consulta. Após posicionar o ponteiro para tabela solicitada é necessário verificar se existe alguma cláusula de condição. Conforme demonstra na Figura 14, existe uma condição denominada WHERE, onde a condição será comparada com os registros da tabela já selecionados. Todos os registros são comparados e caso o valor comparado for compatível com o resultado da tabela, este é selecionado. Independente de o valor ser compatível, todos os registros são analisados fazendo com que a condição tenha o mesmo esforço independente do retorno da consulta.

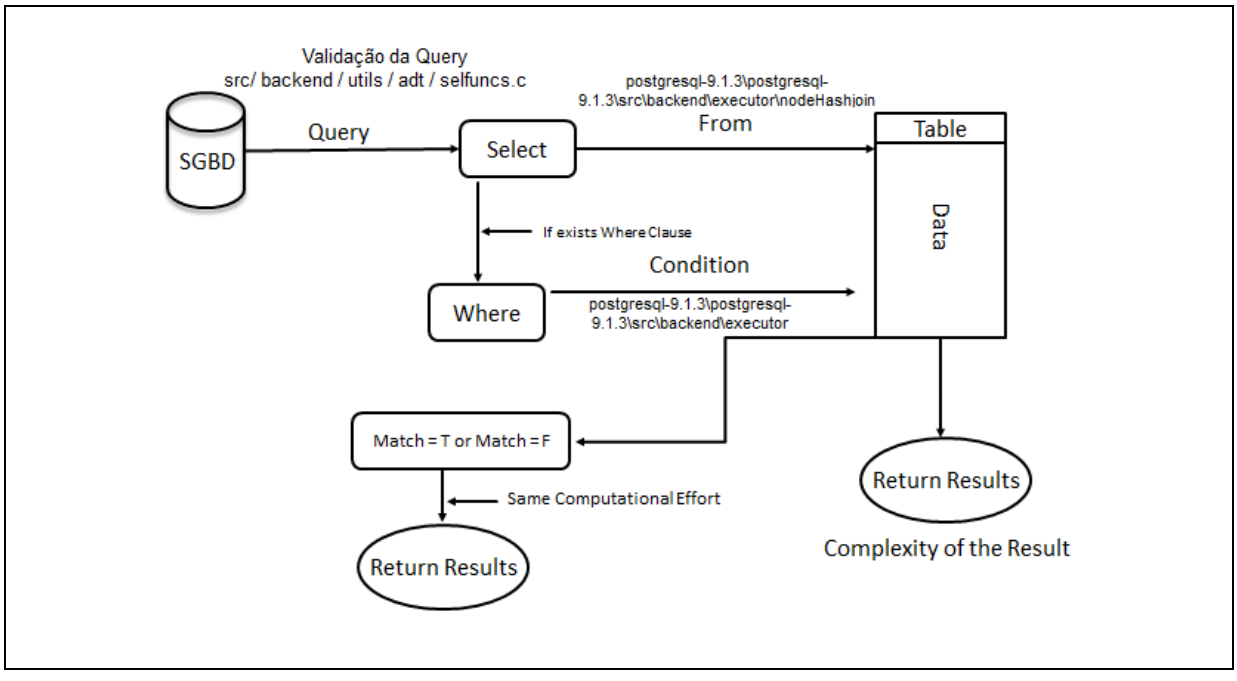


Figura 14. Lógica de cálculo para seletividade da cláusula WHERE

A complexidade da cláusula WHERE é definida por K.O(N) onde K é o número definido de condições. A soma das complexidades de SELECT + WHERE gera a complexidade O(N), pois ambas são executadas juntamente e possuem a mesma ordem.

3.2 Modelagem

A modelagem da ferramenta tem o objetivo de descrever todos os requisitos contextualizados para modelar a ferramenta proposta, mapeando todos os requisitos da ferramenta como: casos de uso e diagrama de sequência. Na subseção a seguir encontra-se o projeto desenvolvido.

3.3 Requisitos Funcionais (RF)

Requisitos Funcionais caracterizam as funcionalidades que devem ser contempladas pela ferramenta para que a mesma cumpra com o proposto. Os requisitos funcionais levantados para a proposta do trabalho são demonstrados abaixo:

- RF01: O sistema deverá permitir ao usuário montar a consulta;
- RF02: A ferramenta deverá fornecer exemplos e a sintaxe dos comandos;
- RF03: A ferramenta deverá demonstrar o plano de execução;
- RF04: A ferramenta deverá demonstrar a complexidade da consulta.

3.4 Regras de Negócio (RN)

Regras de Negócio especificam o funcionamento de regras que são consideradas pertinentes à ferramenta. As regras de negócios são pautadas e definem as características e a condução do projeto, declaram as políticas ou condições da ferramenta que devem ser satisfeitas (PAULA FILHO, 2009). As regras de negócio especificadas para produção do trabalho estão relacionadas abaixo:

- RN01: O usuário só poderá utilizar métodos de consulta como: SELECT, FROM, WHERE, HAVING, GROUP BY e ORDER BY.

3.5 Requisitos não Funcionais (RNF)

Os Requisitos não Funcionais definem as restrições, características e propriedades da ferramenta desenvolvida. Os requisitos devem ser enunciados de forma quantitava e precisa

(PAULA FILHO, 2009). Abaixo estão descritos os requisitos identificados e definidos na ferramenta.

- RNF01: O sistema utilizará plataforma Desktop;
- RFN02: O sistema deverá ser desenvolvido na linguagem Microsoft C#;
- RFN03: O sistema deverá utilizar os algoritmos analisados no banco de dados PostgreSQL.

3.6 Considerações sobre Projetos Similares

No decorrer do desenvolvimento deste projeto, foram efetuadas pesquisas de soluções similares de ferramentas que buscam apoiar a análise de consulta SQL com ênfase em complexidade de algoritmos.

O processo de busca foi realizado com embasamento teórico em artigos científicos como: IEEE – ACM e pela ferramenta de busca Google. Os verbetes utilizados na busca foram na língua estrangeira inglesa e no português e estavam caracterizados da seguinte forma: Desempenho de Queries SQL aplicadas em algoritmos, Complexidade de Algoritmos, métodos de processamento de consultas SQL em banco de dados PostgreSQL. Ao final, nenhuma ferramenta possui o mesmo foco em descobrir como o processamento é efetuado, sua ordem assintótica e qual a complexidade gerada pela consulta. As ferramentas e pesquisas na maioria dos casos eram voltadas a Benchmarks e análise de algoritmos, fugindo assim do objetivo do trabalho que é unir as áreas de Complexidade de Algoritmos e Banco de Dados.

A ferramenta proposta foi modelada com o objetivo de integrar as áreas de banco de dados e complexidade de algoritmos, buscando medir qual o esforço necessário para processar uma consulta SQL e buscar sua complexidade, tendo como propósito mensurar a complexidade da consulta, efetuadas no SGBD PostgreSQL.

3.7 Resultados

O processo de construção da ferramenta contempla diversos tópicos pertinentes que, ao decorrer da análise e levantamento de requisitos, foram ganhando forma e permitindo consolidar a criação da ferramenta.

Durante o processo formalização da ferramenta os seguintes itens serão descritos e detalhados: resultados obtidos, árvore de processamento, validação da ferramenta e o plano de execução e como se chegou ao resultado baseando-se em pesquisas.

3.8 Ferramenta

A criação da ferramenta foi dividida em módulos para aperfeiçoar o processo de desenvolvimento. Após a análise e definição de todos os requisitos e objetivos específicos, à ferramenta foi construida em seis funcionalidades:

1. Conexão: o módulo de conexão permite conectar em diversos bancos de dados localmente ou na rede;
2. Consulta SQL: determina a consulta que será executada no banco de dados conectado;
3. Ações: opções que permitem manipular as consultas, copiar, colar, desfazer a ação, importar um SQL já desenvolvido e submeter a consulta;
4. Result query: retorna os resultados submetidos na consulta SQL executados. Os resultados são obtidos e retornados da mesma forma que no SGBD PostgreSQL;
5. Explain: retorna a árvore de processamento da consulta, seu tipo de estratégia, custo operacional, quantidade de registros por nó executado e os filtros de pesquisas provenientes das cláusulas de condição estabelecidas na consulta;
6. Complexity: exibe a complexidade da consulta resultante.

Conforme descrito na Figura 15, é demonstrado como funciona o processo de preenchimento dos campos de conexão com o banco de dados, como efetuar a consulta, botões de ação e requisição, além do resultado da complexidade da consulta gerada.

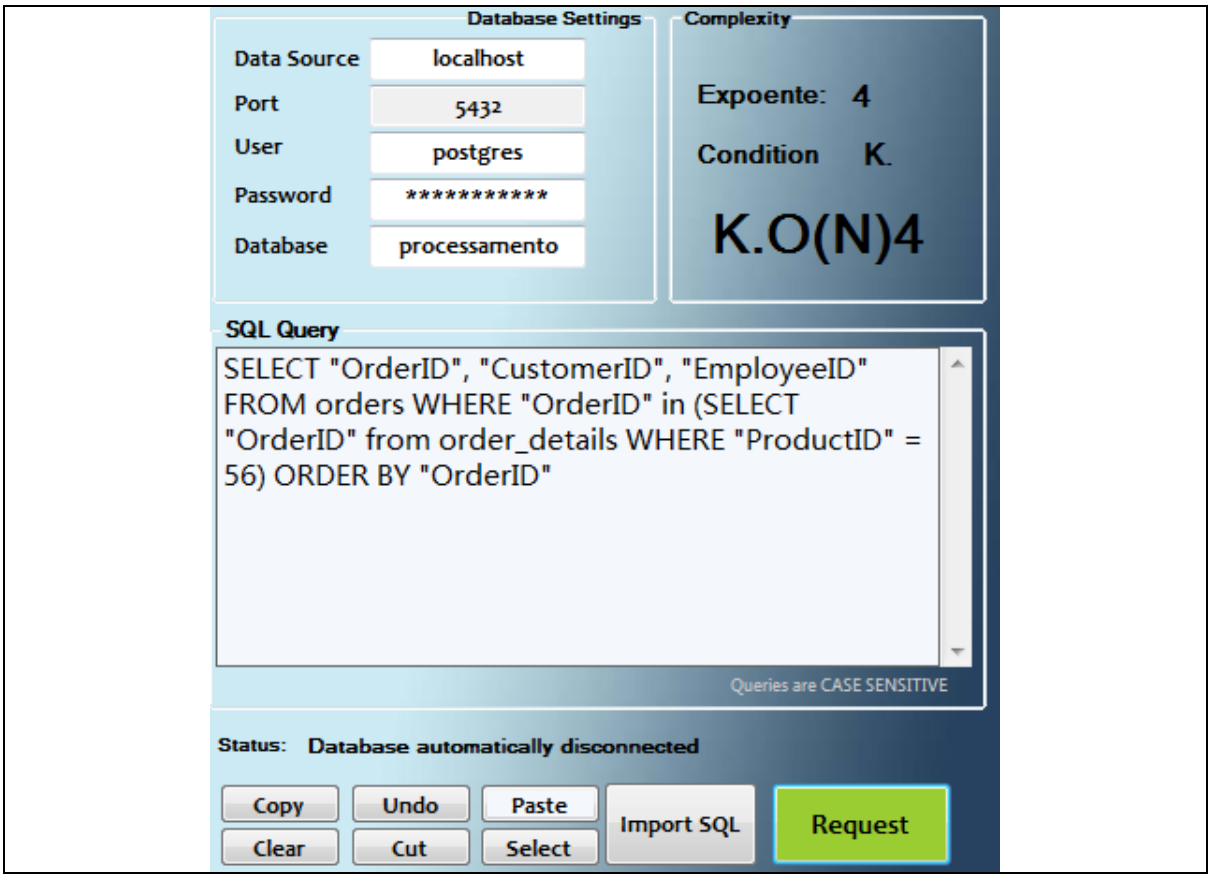


Figura 15. Configurações, pesquisa e resultado da complexidade

O processo de composição da consulta é exibido em forma de árvore. A estrutura a seguir, descrita pela Figura 16, demonstra todos os passos executados pelo SGBD para retornar o resultado solicitado. As funções descritas na Figura podem ser analisadas no Quadro 13.

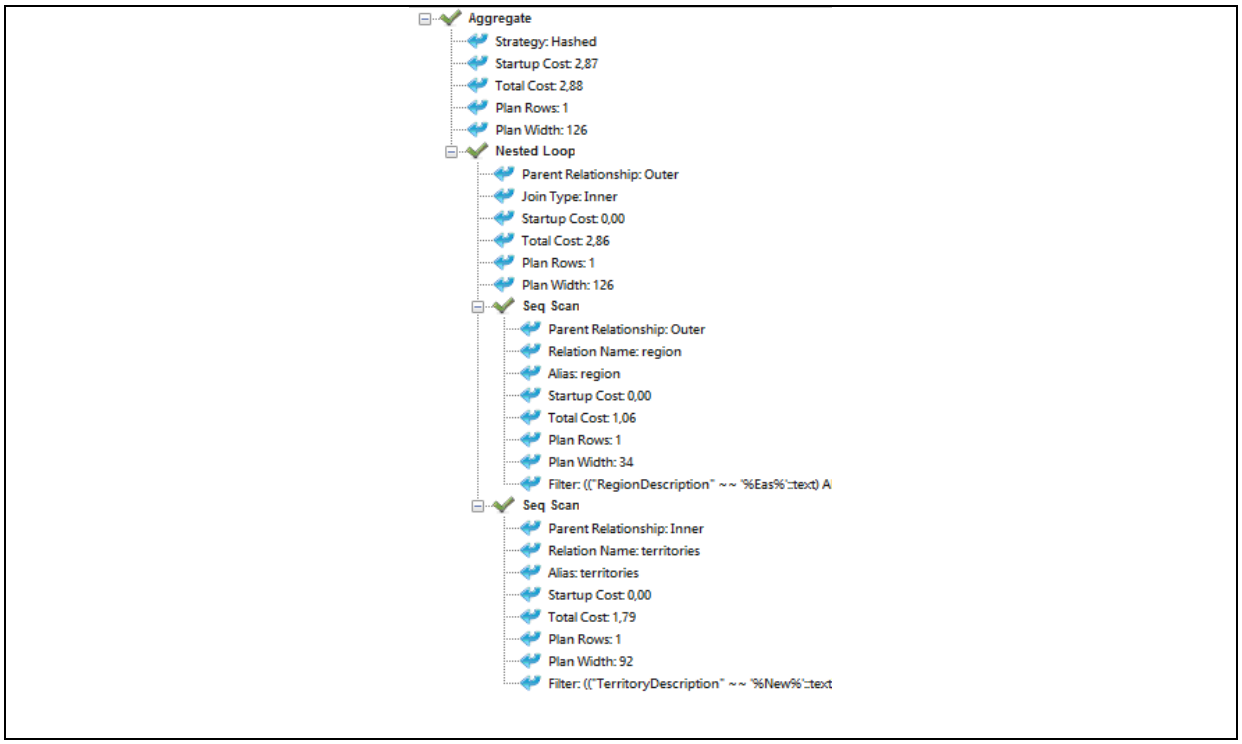


Figura 16. Árvore do processamento da consulta

A interface da ferramenta é de fácil absorção, visualmente divida em três partes. A primeira parte descreve a conexão com o banco de dados, consulta SQL, botões de ação, configurações e o resultado da complexidade gerada pela consulta. Na área centralizada fica o resultado da consulta gerada, denominada "Result Query", onde são exibidos os registros compatíveis com a consulta estipulada. A última parte e mais importante é o "Explain" gerado pela consulta SQL, que exibe a estrutura que constitui a consulta. Na Figura 17, disponível a seguir é possível visualizar a interface da ferramenta e seus métodos de processamento e exibição.

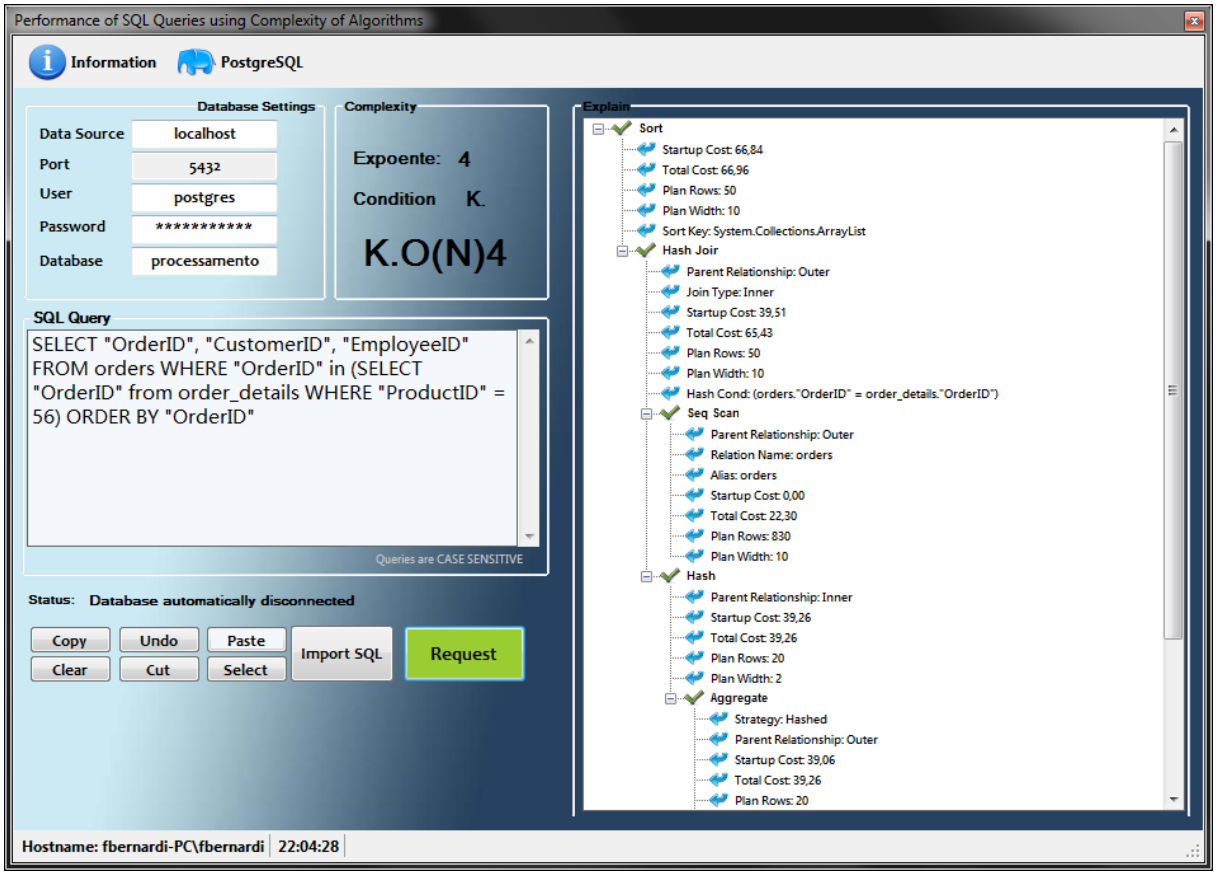


Figura 17. Visão geral da ferramenta

O ciclo de processamento parte da conexão com a base de dados definida pelo utilizador, onde é possível conectar-se a qualquer banco de dados PostgreSQL. Os campos descritos abaixo no Quadro 12 são obrigatórios para garantir uma conexão bem sucedida.

Conexão	**Descrição**
Data Source	Destina-se a configuração de conexão para um determinado banco de dados, local ou servidor.
Port	Definição por onde os dados serão passados e o SGBD estará esperando para receber as informações.
User	Usuário do SGBD.
Password	Senha do usuário do SGBD.
Database	Banco de dados que será utilizado.

Quadro 12. Informações necessárias para conexão

Durante o fluxo de processamento da consulta que é dividido em cinco passos, os primeiros dois passos são vitais para o correto funcionamento e retorno do resultado esperado. O primeiro passo, conexão com o SGBD é responsável pelas informações da conexão como: usuário, senha, nome do banco de dados e o local a ser conectado. A seguir, no passo dois, denominado como "consulta" é efetuada a query que será executa no banco de dados. No passo 3, denominado "Explain" é possível ver o plano que o otimizador de consultas do PostgreSQL gerou. O melhor plano é definido pelo SGBD e a única possibilidade de otimizá-lo é executar o "Explain" antecedendo a consulta. O otimizador de consultas é um dos componentes do banco de dados que busca determinar o modo mais eficiente para executar uma consulta. Após executada a consulta, é gerada sua árvore do plano de execução e a ferramenta calcula a sua complexidade baseado nos resultados obtidos. Na Figura 18 é possível ver o ciclo de execução da ferramenta.

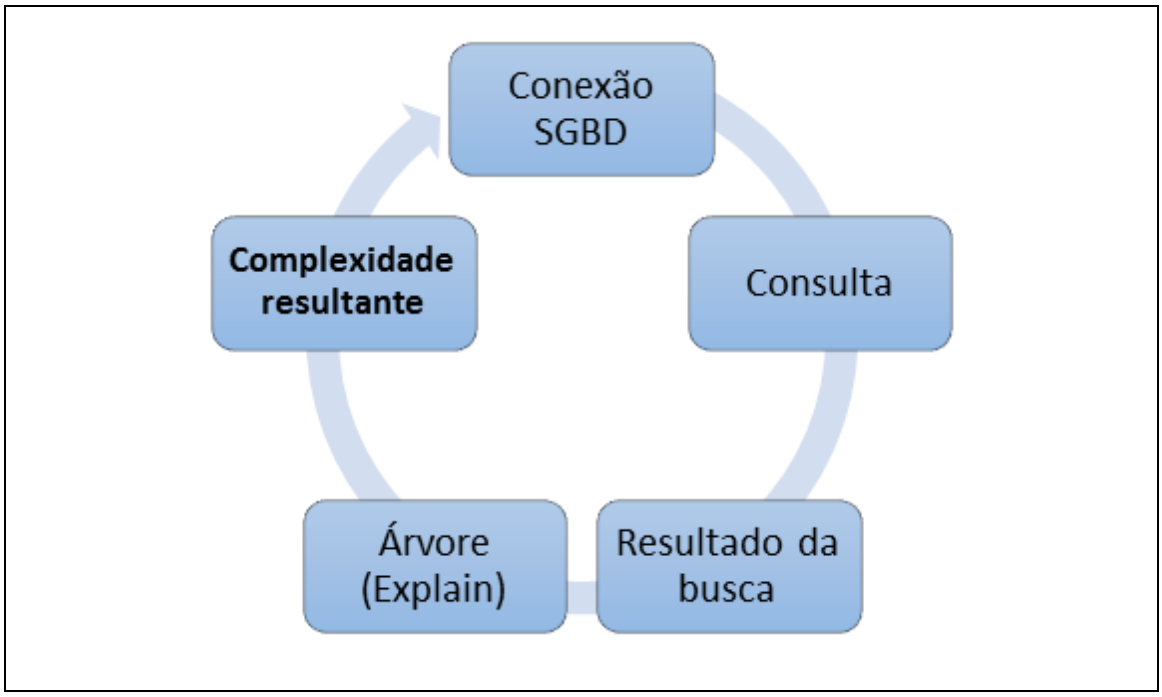


Figura 18. Ciclo de processamento da consulta

3.8.1 Árvore do Plano de Consulta

A árvore de exibição ou TreeView (visão em forma de árvore) possibilita que seja observado como a consulta foi processada em forma hierárquica, tal como dados de subplanos de execução. Com a utilização da árvore de exibição é possível separar cada nó da consulta de forma isolada e testá-los individualmente ou em forma de conjunto, buscando a complexidade singular e por grupos.

Cada consulta SQL gera em seu plano de execução uma chave denominada "Node Type" que simboliza qual a ação que a árvore gerou durante a criação do melhor plano de execução. A chave retorna características que permite extrair a complexidade das consultas SQL após sua execução. Em suas propriedades é exibido o tempo de execução, tempo inicial da execução, quantidade de registros e a largura do plano mensurado em bits.

Durante a execução da consulta, diversas chaves do tipo "Node Type" são exibidas, variando de acordo com a consulta executada. Cada chave possui um “Value” que é buscado na *query* estabelecida durante a montagem do SQL e que resulta em uma ação. As complexidades extraídas estão disponíveis no Quadro 13 e é possível analisar os tipos de chave que foram utilizados para desenvolver a ferramenta e qual o significado de cada.

Chave	**Valor**	**Descrição**	**Complexidade**
Node Type	Seq Scan	Sempre utilizada quando a cláusula SELECT estiver presente.	N
	Sort	Valor presente quando a cláusula de ordenação ORDER BY for requisitada na consulta.	N^2
	Hash Join	Quando uma subconsulta gera um resultado e este precisa ser comparado com outra consulta.	N^2
	Hash	Armazena os dados de uma subconsulta que é comparado com outra consulta.	N^2
	Aggregate	Utilizado durante um agrupamento, media e em juncões de projeção e seleção.	Não possui complexidade
	Nested Loop	São juncões de laço-aninhados. Utilizados quando existe necessidade de utilizar um plano diferente.	N^2

Quadro 13. Valores da chave Node Type

Os algoritmos de relação JOIN são divididos em três planos de execução, de forma que cada plano é utilizado de uma forma específica e definido pelo próprio *planner* do SGBD durante a execução (SMITH, 2010). Quando ocorrer de existir duas ou mais relações na consulta, ocasionando um *join* e todos os planos executáveis já foram encontrados, o *planner*

verificará qual algoritmo de *join* executará. Os algoritmos abaixo estão dispostos durante a execução:

1. Hash Join: O método conhecido como “Hash Join” tem como objetivo encontrar os conjuntos de tuplas (que é formada por uma lista ordenada de colunas). Este algoritmo é muito eficiente quando utilizado em consultas que envolvem tabelas de pequeno a grande porte, gerando uma estrutura “Hash Table”.
2. Merge Join: No método "Merge Join" é efetuada uma ordenação das relações da consulta e depois ordenadas. É percorrido de forma paralela criando tuplas resultantes das junções de todos os atributos que foram descritos na consulta.
3. Nested loop join: Este método de loop aninhado gera uma combinação de linhas do tipo sequencial, indexado e de digitalização. O tempo para executar é diretamente proporcional ao número de linhas na tabela da relação externa, multiplicados pelos registros da interna. São consideradas todas as formas possíveis para se juntar cada linha de um tabela com todas as outras linhas de outra tabela. Existe a possibilidade da utilização do “Index Nested Loop” que fará uso dos índices para otimizar o processo.

Durante o processo de construção do plano, o *planner* examina a quantidade de relações. Caso seja superior a duas, o mesmo constrói uma árvore de passos de junção, formando duas entradas resultantes em cada ramo. O real motivo deste método é encontrar dentre todas as possibilidades o plano aplicável e menos custoso (SMITH, 2010).

A árvore do plano de execução tem um papel fundamental no processo de seleção e construção do melhor plano e durante o método de execução de uma consulta SQL no banco de dados. A Figura 19 descreve como as informações são processadas durante a construção da árvore de execução e suas características.

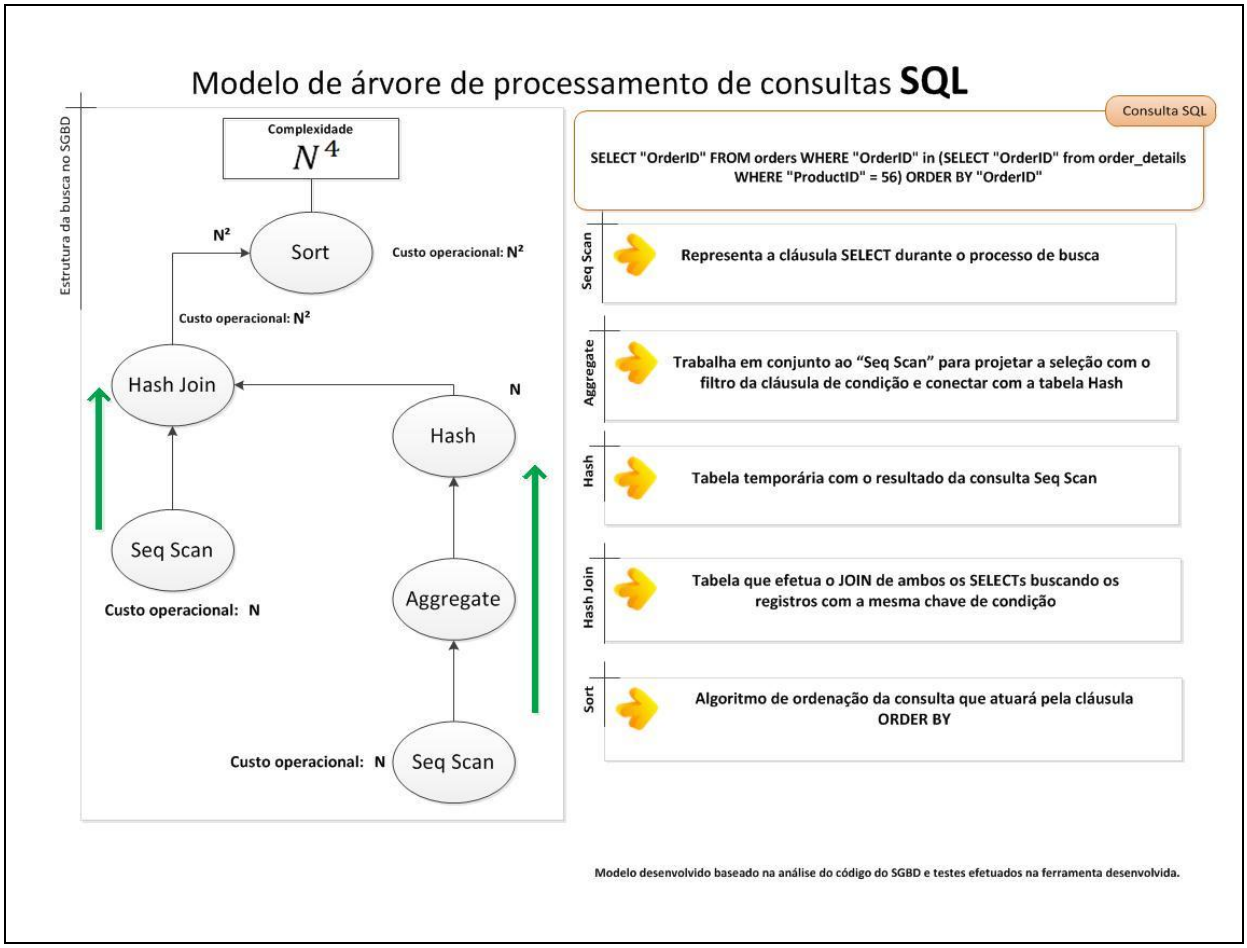


Figura 19. Modelo de árvore de processamento de consultas SQL

Na Figura 19 pode ser descrito como um algoritmo modelo, onde o algoritmo é executado unindo duas tabelas para buscar os registros com a mesma chave e que se encaixam na condição imposta na consulta. Além disso, existe a necessidade de ordenar os registros encontrados resultando na soma de uma complexidade quadrática, conhecido como método "SORT".

Sempre que o número de junções durante a consulta se tornar grande, fica inviável verificar todas as possibilidades de otimização para a consulta específica e aguardar por uma determinação do plano considerado excelente e com um baixo custo. Quando ocorrer esse impasse, o algoritmo genético denominado (GA) é acionado para buscar o melhor método fazendo o cruzamento das junções e criando diversos planos independente do custo. Após gerar todos os planos, os métodos gerados com maior custo operacional são descartados

restando apenas os de baixo custo que são recombinados para buscar uma maior otimização (SMITH, 2010).

3.8.2 Verificação e validação da ferramenta

Os testes de verificação da ferramenta foram realizados a fim de analisar se todos os processos propostos no projeto foram implementados corretamente e contempla as especificações descritas nos requisitos.

Os processos de validação da ferramenta foram divididos em três etapas durante o processo de implementação: (*i*) testes de funcionalidade; (*ii*) verificação e validação da complexidade retornada pela ferramenta, tendo como entrada *queries* com cláusulas isoladas; (*iii*) verificação e validação da complexidade retornada pela ferramenta, tendo como entrada *queries* compostas de diversos tipos de cláusulas.

Na etapa (*i*) foram avaliados os processos da ferramenta que determinam se todos os levantamentos de requisitos e construção dos artefatos estão de acordo com o previsto, se a ferramenta foi construída de forma correta e se os requisitos descritos no projeto estão em conformidade com o proposto.

Na etapa (*ii*) o banco de dados criado foi submetido a uma carga de dados para análise de complexidade gerada pela ferramenta, baseado nas condições impostas durante a consulta e comparados com resultados obtidos das análises externas da ferramenta. Cada lote de testes utilizou especificamente um tipo de cláusula de consulta SQL. Com o auxilio do *planner* do banco de dados, foi possível montar a estrutura das consultas exatamente da mesma forma que o SGBD e aumentar a precisão dos resultados.

A última etapa (*iii*) foi de validação das cláusulas agrupadas. As entradas de instruções SQL com funções de junção, subconsultas e agregação como (AVG, SUM, COUNT) foram submetidas a analises para determinar se o comportamento foi o esperado. Durante todo processo de construção das cláusulas, os resultados gerados eram comparados com outros resultados para buscar as similaridades de comportamento em consultas diferentes, muitas vezes completamente distintas. O Quadro 14 descreve as etapas de testes.

Etapa	O que foi executado	Resultado
1	Levantamento de pré-requisitos Construção dos artefatos.	Todas as definições do projeto estão em conformidade com a especificação.
2	Análise individual das cláusulas SQL (SELECT, FROM, WHERE, ORDER BY e GROUP BY). Ex: select * from order_details	Os testes foram positivos quando efetuados na ferramenta e está em conformidade com os testes executados fora da ferramenta.
3	Agrupamento das cláusulas SQL utilizando Join entre as tabelas. Ex: SELECT "OrderID", "CustomerID", "EmployeeID" FROM orders WHERE "OrderID" in (SELECT "OrderID" from order_details WHERE "ProductID" = 56) ORDER BY "OrderID"	O comportamento dos agrupamentos foram os esperados. As cláusulas agrupadas retornaram os resultados esperados.

Quadro 14. Validação dos testes

Por meio dos testes efetuados e resultados obtidos, pode-se afirmar que todos os testes realizados durante o processo construção e pós-desenvolvimento estão de acordo com os requisitos especificados. Todos os objetivos propostos foram cumpridos neste projeto.

3.8.3 Plano de execução

O plano de execução nada mais é que diluir uma consulta em diversas partes (blocos de consulta), processá-la de forma individual já que não ocorre controle de concorrência e agrupá-las para gerar a saída requisitada.

Durante o processo de pesquisa dos métodos de execução, não foi possível chegar a nenhuma solução conclusiva do método de processamento, desta forma foi necessário estudar

o Código-Fonte do SGBD para assimilar as ligações efetuadas pelo PostgreSQL. Conforme descrito na seção “Árvore do Plano de Consulta”, cada cláusula possui uma chave e um valor conforme descrito na Figura 20, disponível no Apêndice A1.

A cláusula de SELECT “Node Type” é conhecida pelo valor “Seq Scan”. Para recuperar essas informações e calcular a complexidade da pesquisa, um algoritmo foi desenvolvido para percorrer a "árvore de processamento" ou do plano de execução.

O Quadro 15, disponível no Apêndice A2, descreve a função recursiva que auxilia na construção da complexidade das consultas executadas. Sempre que a chave e valor coincidirem com as chaves estabelecidas na função, é incrementado um valor de acordo com seu peso. Diversas estratégias foram analisadas para chegar ao algoritmo com maior aptidão de acertos.

4 CONCLUSÕES

O objetivo principal deste trabalho é aproximar a área de Complexidade de Algoritmos e banco de dados. Para isso acontecer, se fez necessário estudar uma relação que aproximasse as duas frentes de estudo. Diante da área de banco de dados, o PostgreSQL que é um dos mais utilizados em todo mundo foi utilizado como tema de pesquisa. A Complexidade de Algoritmos foi aplicada para medir o esforço de trabalho e retornar a complexidade das consultas executadas neste SGBD.

O estudo realizado buscou mecanismos para unir duas áreas completamente distintas com foco em entender o processo das consultas SQL no banco de dados PostgreSQL e extrair qual a Complexidade do Algoritmo executado pelo *Planner* durante o processo de formalização da consulta.

A primeira tarefa realizada neste Trabalho Técnico-científico de Conclusão do Curso foi reunir informações suficientes sobre o SGBD, buscar auxílio na comunidade desenvolvedora e analisar o Código-Fonte que, sem dúvida, auxiliaram na compreensão do contexto. Todas as informações levantadas foram de cunho relevante, permitindo a continuidade do trabalho até seu estágio final.

O processo de levantamento de informações permitiu que diversas ferramentas fossem analisadas buscando similaridades, porém nenhuma ferramenta possui foco na Complexidade de Algoritmos sob um banco de dados, independente o SGBD.

Durante a etapa de especificação da ferramenta ainda não era possível prever como interpretar e identificar com clareza as funcionalidades retornadas pelo SGBD, durante o processo de execução de uma consulta SQL. A partir da análise do Código-Fonte, do qual foi desmembrado no TTC 1, foi possível descobrir a complexidade individuais das cláusulas definidas, assim como, analisar como as chamadas das funções eram efetuadas, quais os tipos de parâmetros eram passados para construir uma saída.

Após obter a complexidade individual, partiu-se para estruturação das cláusulas agrupadas, utilizando o plano de execução. Uma solução encontrada foi à utilização do JSON, que permite representar os objetos em Java Script.

A implementação do JSON possibilitou que a estrutura das consultas processadas pelo SGBD fosse extraída, facilitando a modelagem da árvore de processamento e suas características, a fim de, montar a complexidade das consultas. Cada cláusula processada possui uma característica individual que, quando agrupadas, geram resultados dispersos as já analisadas.

Por fim, após a construção da ferramenta, a mesma passou por diversos testes, buscando avaliar e validar se realmente toda análise estava em conformidade com o especificado e planejado. As cláusulas de seleção se comportaram conforme esperado e os testes foram positivos com as análises efetuadas e os testes submetidos. Os resultados do projeto mostraram que a ferramenta é eficaz e pode ser considerada uma alternativa positiva para ser utilizada nas aulas da disciplina de Complexidade de Algoritmos do Curso de Ciência da Computação na UNIVALI.

O impasse quanto à descoberta das complexidades foi devido a necessidade do desmembramento do Código-Fonte do PostgreSQL para conseguir mensurar a complexidade das cláusulas SQL. A dificuldade quanto ao entendimento da estrutura das classes, bibliotecas e métodos de otimização com utilização de ponteiros são considerados um fator desafiador.

Como trabalhos futuros, existe a possibilidade da ferramenta conectar-se em diversos SGBDs, pois hoje a ferramenta está limitada apenas ao PostgreSQL. Essa opção tornaria a ferramenta mais abrangente e concreta. Infelizmente não são todos os SGBDs que possuem o código aberto, que resultaria em uma limitação da quantidade de SGBD’s disponíveis.

Um fator determinante quando se trata de SGBDs é a utilização de índices. O projeto atual não fez menção a índices, estratégias de utilização e nenhum tipo de auxilio para este. A hipótese de inserir índices no planejamento de trabalhos futuros agregaria qualidade à ferramenta, pois permitiria a exploração de diversas opções de desempenho e personalização dos métodos de processamento.

REFERÊNCIAS

ARORA, S.; BARAK, B. **Computational Complexity**: a modern approach. Cambridge Nova York: University Press, 2009.

CAPELLO, R; CINTRA S.J.P; MAGALHAES, C.G. **Otimização de Consultas Relacionais**. 2005. 13f. (Trabalho de Pós-Graduação em informática) - Instituto de Computação, Universidade Estadual de Campinas, 2005.

DATE, C.J. **Introdução a sistemas de bancos de dados.** C.J. Date 1990. Rio de Janeiro: Campus,1990.

DATE, C.J. **Introdução a sistemas de bancos de dados.** 7.ed. C.J. Date 1990. Rio de Janeiro: Campus, 2000.

DATE, C.J. **Introdução a sistemas de bancos de dados.** 8.ed. C.J. Date 2003. Rio de Janeiro: Campus, 2003.

GUTTOSKI, P.B. **Otimização de Consultas no PostgreSQL Utilizando o Algoritmo de Kruskal**. 2006. 94 f. Dissertação (Mestrado em Computação) - Programa de Pós-Graduacão em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná, Curitiba, 2006.

ELMASRI, R. **Sistemas de banco de dados.** 4.ed**.** São Paulo: Addison, 2005.

GILMORE, J.G.; TREAT,H.R. **Beginning PHP and PostgreSQL 8: From Novice to Professional.** New York: Apress, 2006.

MACHADO, F.E.; ABREU,Mauricio. **Projeto de Banco de Dados.** 11.ed. São Paulo: [s.n.], 1996.

KLINE, K.E.; KLINE, D. **SQL:** o guia essencial - manual de referência do profissional. Rio de Janeiro: Alta Books, 2010.

MILANI, A. **MySQL:** guia do programador. São Paulo: Novatec, 2006.

MILANI, A. **PostgreSQL:** guia do programador. São Paulo: Novatec, 2008.

OLIVEIRA, Kaui Aires. **PostgreSQL x MySQL. Qual Escolher? –** Disponível em: < http://www.devmedia.com.br/postgresql-x-mysql-qual-escolher/3923> Acesso em: 04 abr. 2012.

PAULA FILHO, W.P. **Engenharia de Software.** 3.ed. Rio de Janeiro: LTC, 2009.

POSTGRESQL. **Documentation** - Disponível em < http://www.postgresql.org/docs/manuals/archive/ > Acesso em: 18 maio. 2012.

SANTOS, W. Pesquisa de Dados. – Disponível em: <http://www.do.ufgd.edu.br/WellingtonSantos/Algo/Pesquisa.pdf> Acesso em: 05 maio. 2012.

SILBERSCHATZ, A.; KORTH, H.F. **Sistemas de banco de dados**. 2.ed. São Paulo: Makron Books, 1995.

SILBERSCHATZ, A.; KORTH, H.F.; SUDARSHAN, S. **Sistemas de banco de dados**. 3.ed. São Paulo: Makron Books, 1999.

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

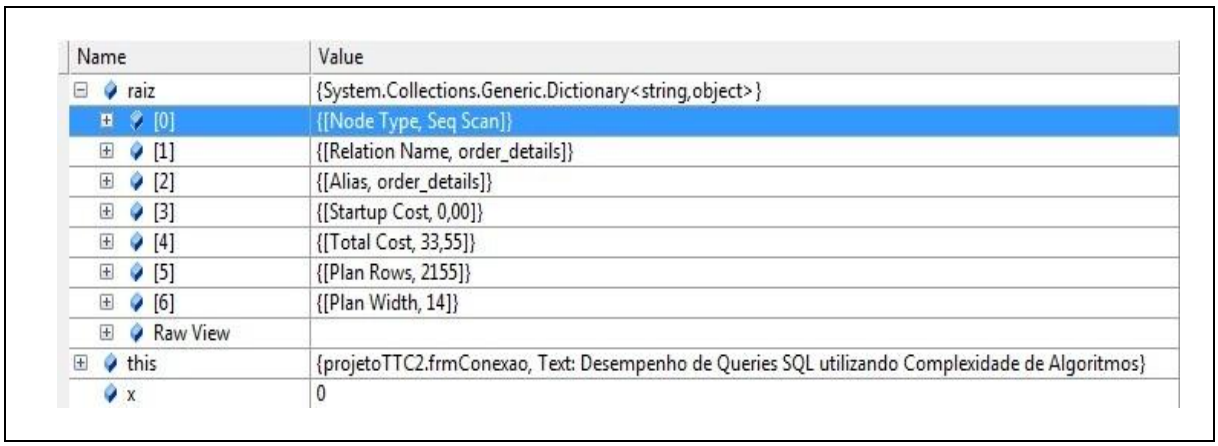
SIPSER, M. **Introdução à Teoria da Computação.** 2. ed., São Paulo: Cengage Learning, 2007. ISBN: 978-85-221-0499-4.

SMITH, **G. PostgreSQL 9.0 High Performance.** 1. ed., Birmingham: Packet Publishing, 2010.

TOSCANI, L.V; VELOSO, S.A.P. **Complexidade de Algoritmos**. 2.ed. Rio Grande do Sul: Sagra Luzzato, 2008.

APÊNDICE A. CLÁUSULAS SQL

A.1. RELAÇÃO DAS CARACTERÍSTICAS



Name	Value
raiz	{System.Collections.Generic.Dictionary<string,object>}
[0]	{[Node Type, Seq Scan]}
[1]	{[Relation Name, order_details]}
[2]	{[Alias, order_details]}
[3]	{[Startup Cost, 0,00]}
[4]	{[Total Cost, 33,55]}
[5]	{[Plan Rows, 2155]}
[6]	{[Plan Width, 14]}
Raw View	
this	{projetoTTC2.frmConexao, Text: Desempenho de Queries SQL utilizando Complexidade de Algoritmos}
x	0

Figura 20. Características das cláusulas SQL

A.2. RETORNO DA COMPLEXIDADE

```
public int calcComplexity(dynamic raiz){

int x = 0;
  if (raiz is ArrayList)
      {

       foreach (var busca in raiz)

       {

          x = x + calcComplexity(busca);

         }
        }else{
             foreach (var busca in raiz)
             {
         if (busca.Key.Equals("Plan"))
           {
             x += calcComplexity(raiz["Plan"]);
           }

         if (busca.Key.Equals("Plans"))
           {

             x += calcComplexity(raiz["Plans"]);

           }

         if   (busca.Key.Equals("Node   Type")   &&   busca.Value.Equals("Seq
Scan"))
           {
             x += 1;
           }

         if (busca.Key.Equals("Node Type") && busca.Value.Equals("Sort"))
          {
             x += 2;
          }
        }
      }

      return x;
 }
```

Quadro 15. Função recursiva de Complexidade de Algoritmos