UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FRANCISCO DANIEL BEZERRA DE SOUZA PRACIANO

REFINAMENTO DAS ESTIMATIVAS DE CARDINALIDADE NO PROCESSAMENTO DE CONSULTAS

FORTALEZA

2020

FRANCISCO DANIEL BEZERRA DE SOUZA PRACIANO

REFINAMENTO DAS ESTIMATIVAS DE CARDINALIDADE NO PROCESSAMENTO DE CONSULTAS

Dissertação apresentada ao Curso de Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Ciência da Computação

Orientador: Prof. Dr. Javam de Castro Machado

FORTALEZA

2020

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

P91r Praciano, Francisco Daniel Bezerra de Souza.
Refinamento das Estimativas de Cardinalidade no Processamento de Consultas / Francisco Daniel Bezerra de Souza Praciano. – 2020.
107 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2020.
Orientação: Prof. Dr. Javam de Castro Machado.

1. Sistemas Gerenciadores de Banco de Dados. 2. Processamento de Consultas. 3. Otimização de Consultas. 4. Estimação de Cardinalidade. 5. Aprendizagem de Máquina. I. Título.

CDD 005

FRANCISCO DANIEL BEZERRA DE SOUZA PRACIANO

REFINAMENTO DAS ESTIMATIVAS DE CARDINALIDADE NO PROCESSAMENTO DE CONSULTAS

Dissertação apresentada ao Curso de Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Ciência da Computação

Aprovada em: 03 de Março de 2020

BANCA EXAMINADORA

Prof. Dr. Javam de Castro Machado (Orientador)
Universidade Federal do Ceará (UFC)

Profa. Dra. Carmem Satie Hara
Universidade Federal do Paraná (UFPR)

Prof. Dr. Angelo Roncalli Alencar Brayner
Universidade Federal do Ceará (UFC)

Aos meus amados pais, Praciano e Angélica.

AGRADECIMENTOS

A Deus e São Francisco, pelas virtudes de fortaleza, esperança e fé.

Aos meus amados pais, Praciano e Angélica, que me amaram e deram condições para que eu estudasse com afinco. Se hoje eu tenho a possibilidade de finalizar este trabalho, é porque um dia vocês lutaram para que eu tivesse uma base solidificada.

Aos meus queridos irmãos, Rafael, Patrícia e Paulo, que me ajudaram com um companheirismo inigualável, auxiliando-me na persistência em busca dos meus sonhos.

Ao meu orientador, Prof. Dr. Javam Machado, pela convivência diária durante o andamento da pesquisa, por sua disposição em compartilhar seu conhecimento sobre banco de dados e, principalmente, guiar-me por meio dos conselhos dados durante as muitas reuniões que participamos nesse período. Dedico-lhe minha profunda gratidão pela oportunidade de ter compartilhado essa jornada sob sua orientação. Aproveito para estender esse agradecimento também à Profa. Dra. Rosélia Machado por cuidar de todos os trâmites burocráticos, além das pequenas conversas que tivemos durante os cafés.

Aos membros da banca examinadora, Profa. Dra. Carmen Hara e Prof. Dr. Angelo Brayner, pela disponibilidade para avaliar este trabalho, assim como pelas sugestões e críticas construtivas tecidas no âmbito do assunto desta dissertação. Ressalto que é uma honra tê-los em minha banca.

A todos os professores do Departamento de Computação da Universidade Federal do Ceará (UFC), pelos ensinamentos que me possibilitaram ter condições para desenvolver este trabalho. Em especial, agradeço ao Prof. Dr. Victor Campos e Prof. Dr. Fernando Trinta, pois foram eles os responsáveis por me apresentarem o mundo da academia durante o período que passei na iniciação científica. Também merece destaque o Prof. Dr. João Fernando devido à sua valiosa ajuda no momento em que iniciei meus estudos na UFC.

Aos amigos que compartilharam comigo os momentos de diversão que ajudaram a reduzir a pressão durante este trabalho: Emídio Fernandes, Eric Andrade, Fhiliphe Rodrigues e Makson Texeira.

Aos amigos que convivi diariamente no ambiente acadêmico: André Mendonça, Bruno Leal, Davi Torres, Diogo Martins, Eduardo Rodrigues, Edvar Bento, Felipe Timbó, Hélio Sales, Iago Chaves, Isabel Lima, Israel Vidal, José Serafim, Maria Maia, Paulo Amora, Sérgio Marinho e Victor Aguiar. Em particular, ao Ítalo Cavalcante que me acompanhou durante esta pesquisa e me ajudou na conclusão deste trabalho, bem como ao Lucas Falcão que me auxiliou

com as dúvidas da área de aprendizagem de máquina.

A todos os integrantes do Laboratório de Sistemas e Bancos de Dados (LSBD) no qual eu participei durante toda a minha caminhada. Sem dúvidas, a estrutura disponibilizada a mim e a interação com os seus integrantes foram importantes.

À UFC, por disponibilizar todo um aparato que viabilizou o desenvolvimento da pesquisa desta dissertação.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), pelo suporte financeiro concedido através de uma bolsa de estudos. O presente trabalho foi realizado com apoio da CAPES - Código de Financiamento 001.

Para finalizar, cumpre esclarecer que considero que não há seção melhor para dar início ao texto que descreve o trabalho realizado ao longo de todo esse período que levei para concluir esta dissertação, pois as pessoas aqui citadas foram importantes na consolidação deste trabalho, as quais eu devo todo o sentimento de gratidão. Sem vocês, podem ter certeza, esse trabalho não existiria de forma alguma. Por isso, a todos, muito obrigado!

“Work hard, have fun and make it happen.”

(Rafael Nadal)

RESUMO

Os *Sistemas de Gerenciamento de Bancos de Dados (SGBDs)* fazem uso de uma linguagem declarativa de modo a permitir que consultas aos dados armazenados sejam realizadas. Nessa abordagem, os usuários descrevem por meio de uma consulta textual aquilo que eles desejam obter dos SGBDs, deixando para estes a definição da maneira pela qual os dados serão processados e, por fim, recuperados. Por conseguinte, os SGBDs precisam escolher, dentre as diferentes possibilidades de executar uma dada consulta, aquela que, baseado em um processo de estimação, seja a de melhor desempenho. Denomina-se de cardinalidade o número de tuplas que resultam da aplicação de um operador relacional sobre uma relação. Essa informação é estimada pelos SGBDs para calcular o número de tuplas que resultam de uma operação e servem de entrada para uma outra no processamento de uma consulta. O processo de otimização, que procura gerar a melhor forma de processar uma consulta, faz uso dessas cardinalidades estimadas para tomar decisões de otimização, tais como escolher a ordenação de predicados. Isto posto, por meio desta dissertação, propomos uma nova abordagem para o cálculo das estimativas de cardinalidade das operações de uma consulta a fim de orientar o motor de execução dos SGBDs a realizar a escolha correta da maneira que a consulta será executada, isto é, escolher a melhor forma possível ou pelo menos aquela que não seja a pior. Acreditamos que acrescentar o uso de modelos de Aprendizagem de Máquina no processo de estimar a cardinalidade das consultas deve levar a execução mais eficiente das consultas. Para assegurar essa hipótese, foram realizados testes experimentais usando o SGBD PostgreSQL. Por fim, depreende-se dos experimentos preliminares apresentados neste trabalho que essa nova abordagem pode resultar em melhorias no âmbito do processamento de consultas em SGBDs, especialmente na geração de estimativas de cardinalidade.

Palavras-chave: SGBD. Processamento de Consultas. Otimização de Consultas. Estimação de Cardinalidade. Aprendizagem de Máquina.

ABSTRACT

Database Management Systems (DBMSs) make use of a declarative language in order to allow queries to stored data to be performed. In this approach, users describe through a textual query what they want to get from DBMSs, leaving them to define how data will be processed and ultimately retrieved. Therefore, DBMSs must choose from the different possibilities of executing a given query which one, based on an estimation process, is the one with the best performance. Cardinality is the number of tuples that result from applying a relational operator to a relation. This information is estimated by the DBMSs to calculate the number of tuples that result from one operation and serve as input to another in the processing of a query. The optimization process, which seeks to generate the best way to process a query, makes use of these estimated cardinalities to make optimization decisions, such as choosing predicate ordering. That said, through this dissertation, we propose a new approach to calculate the cardinality estimates of query operations in order to guide the execution engine of the DBMSs to make the correct choice of the way the query will be executed, that is, choose the best possible form or at least avoid the worst one. We believe that adding the use of Machine Learning models in the process of estimating query cardinality should lead to more efficient query execution. To ensure this hypothesis, experimental tests were performed using the PostgreSQL. Finally, it is possible to conclude from the preliminary experiments presented in this work that this new approach may result in improvements in query processing in DBMSs, especially in the generation of cardinality estimates.

Keywords: DBMS. Query Processing. Query Optimization. Cardinality Estimation. Machine Learning.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

SGBD	*Sistema de Gerenciamento de Bancos de Dados*
OCC	*Otimizador de Consultas baseado em Custo*
SQL	*Linguagem de Consulta Estruturada*
PE	*Plano de Execução*
GBDT	*Gradient Boosting Decision Tree*
LEO	*DB2's LEarning Optimizer*
MSCN	Rede Neural Convolucional para Múltiplos Conjuntos
IMDB	Banco de Dados de Filmes da Internet
BD	*Banco de Dados*
IVA	*Independência de Valores de Atributos*
HC	*Hipótese da Contenção*
JBCS	*Journal of the Brazilian Computer Society*
LightGBM	*Light Gradient Boosting Machine*
FNC	*Forma Normal Conjuntiva*
ASA	*Árvore Sintática Abstrata*
ED	*Estrutura de Dados*
PC	*Plano de Consulta*
SPJ	*Seleção-Projeção-Junção*
DBA	*Administrador de Bancos de Dados*
CPU	Unidade Central de Processamento
MAE	*Máquinas de Aprendizado Extremo*
TR	*Tempo de Resposta*
TO	*Tempo de Otimização*
TE	*Tempo de Execução*
GOSS	*Gradient-based One-Side Sampling*
EFB	*Exclusive Feature Bundling*
EQM	*Erro Quadrático Médio*

LISTA DE SÍMBOLOS

$\mathcal{R}$	Conjunto de relações de um BD
$\mathcal{A}$	Conjunto de atributos de uma relação
$\mathcal{D}$	Domínio que define o intervalo de valores aceitos
Q	Consulta Seleção-Projeção-Junção
R_i	Relação i de um BD
A_j	Atributo j de uma relação
T_k	Tupla cujo índice é k
$R_i.A_j$	Atributo j da relação R_i
$T_k.A_j$	Atributo j da tupla T_k
n	Cardinalidade do conjunto $\mathcal{R}$
m	Cardinalidade do conjunto $\mathcal{A}$
$\mathcal{D}_{A_j}$	Domínio do atributo A_j
$\mathcal{V}_{A_j}$	Conjunto de valores do atributo A_j
$\mathbb{N}$	Conjunto dos números naturais
$\in$	Relação de pertinência
σ	Operação de seleção
$\bowtie$	Operação de junção
π	Operação de projeção
$\times$	Produto cartesiano
O	Operador Relacional
c	Cardinalidade de uma operação de consulta
$\hat{c}$	Estimativa de cardinalidade de uma operação de consulta
p	Predicado ou Cláusula
l	Número de cláusulas
$\mathcal{P}$	Composição de predicados
s	Seletividade

$\mathcal{T}$	Distribuição unidimensional ou multidimensional
f_v	Frequência do valor v em um atributo
q	Quantidade de valores distintos presentes no atributo
X	Conjunto de dados de treinamento
f	Função linear ou não linear
M	Modelo de Aprendizagem de Máquina
x	característica ou *feature*
y	Rótulo ou valor de saída
∂	Derivada parcial
L	Função de custo (ou perda)
$\|S\|$	Tamanho de S

SUMÁRIO

1 INTRODUÇÃO

Desde a apresentação do modelo relacional (CODD, 1970), os SGBDs passaram a utilizar uma linguagem declarativa , a *Linguagem de Consulta Estruturada* (SQL), amplamente adotada nos sistemas atuais, com o intuito de fornecer aos usuários uma maneira mais intuitiva e direta de recuperar os dados desejados, como pode ser observado na Figura 1. Em uma linguagem declarativa, o usuário informa quais são os dados que deseja, sem entrar em detalhes de como esses dados devem ser recuperados. Essa abordagem tem algumas vantagens. Por exemplo, os usuários preocupam-se somente com o que eles desejam, deixando para os SGBDs a tarefa de encontrar a melhor forma de uma linguagem imperativa recuperar os dados. Com isso, esses sistemas têm de avaliar e escolher a opção mais otimizada (ou próxima disso) para processar os dados requisitados. Ou seja, a árdua tarefa de encontrar a melhor maneira de responder a uma consulta é retirada da responsabilidade do usuário e, então, transferida para os SGBDs. Percebe-se, claramente, que essa transferência de responsabilidade traz benefícios devido ao fato das máquinas serem capazes de gerar e avaliar uma maior quantidade de alternativas para responder a uma consulta, tornando o processo de otimização das consultas automático. Para tanto, surge o desafio de definir e desenvolver técnicas que abordem e apresentem soluções para a geração automática de planos de execução para as consultas de usuários.

Para responder consultas SQL, os SGBDs precisam realizar três macroetapas, a saber: processar, otimizar e executar (ELMASRI; NAVATHE, 2000). Neste trabalho, o foco será na etapa de otimização. Tendo como base os fundamentos propostos no artigo seminal que apresentou o Sistema-R (ASTRAHAN *et al.*, 1976), adotou-se amplamente nos projetos dos SGBDs o uso de um *Otimizador de Consultas baseado em Custo* (OCC), que tem como objetivo enumerar um conjunto grande de Planos de Execução (PEs) equivalentes. Dada uma consulta declarativa, o OCC baseia-se em um modelo de custo para escolher o PE que apresenta o menor

Figura 1 – Interface da interação entre usuários e SGBDs.

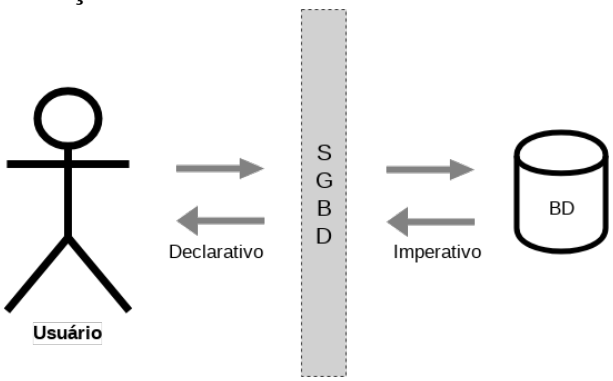
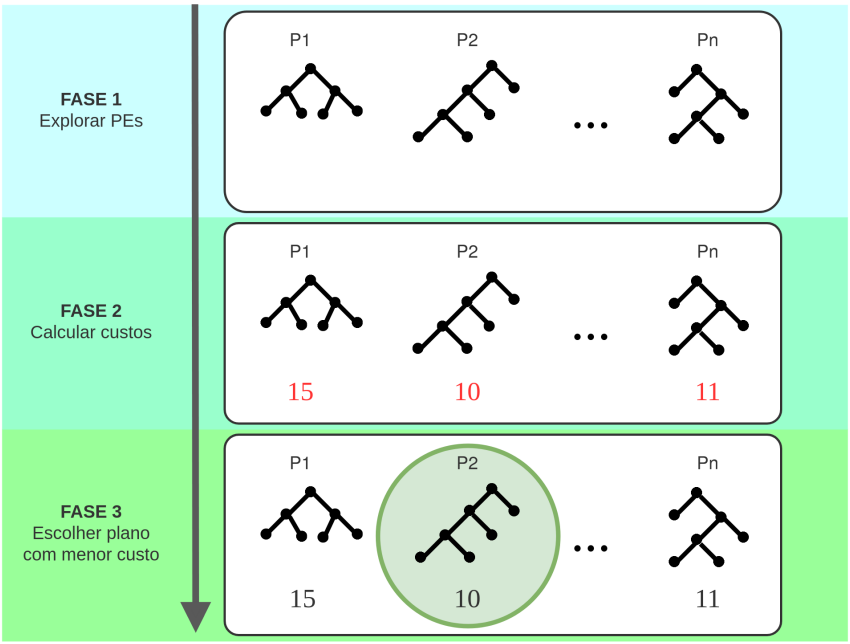


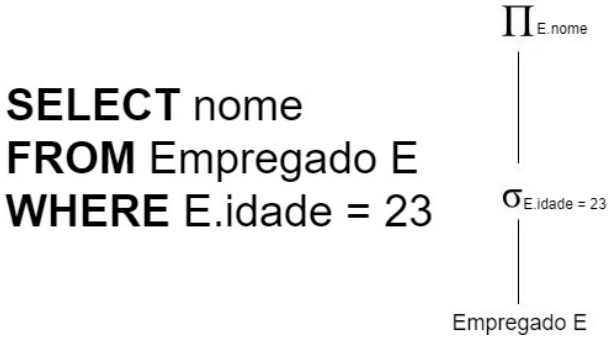
Figura 2 – Etapas realizadas por um OCC.



custo, como ilustrado na Figura 2. Esta imagem apresenta as três fases de um OCC, sendo que a primeira está relacionada com a exploração das possibilidades dos PEs equivalentes. Logo após, o custo desses PEs são calculados. No final, o plano com o menor custo é escolhido, sendo que no caso o selecionado foi o marcado por um círculo de cor verde.

Um *Plano de Execução* (PE) é normalmente estruturado em uma árvore binária cujos nós representam as operações relacionais existentes naquela consulta e as arestas caracterizam o fluxo de dados que ocorrerá entre os nós daquele plano de consulta. Um exemplo dessa estrutura pode ser visualizado na Figura 3, a qual também apresenta a consulta SQL associada que seleciona os empregados com idade igual a 23. Um modelo de construção de PEs seguindo as três macroetapas foi proposto no sistema Volcano (GRAEFE, 1994) e ainda hoje é amplamente implementado pelos SGBDs. Além disso, afirma-se que dois PEs são equivalentes quando a execução independente de um deles irá gerar o mesmo resultado do outro plano, sendo que ambos irão produzir o resultado esperado da consulta declarativa que representam. Já em relação ao modelo de custo, busca-se construí-lo de maneira a refletir a quantidade de recursos (e.g., processamento, memória) que será necessário a um PE no momento de sua execução. Em suma, o OCC tem como objetivo escolher o PE cujo custo requerido é o menor dentre o conjunto gerado de PEs equivalentes. Dessa forma, o SGBD executará a consulta submetida pelo usuário com um tempo de execução próximo do ótimo, isto é, perto daquele tempo mínimo necessário para processar os dados. A não escolha do plano ótimo pode ter vários motivos. Pode-se citar a

Figura 3 – Exemplo de consulta SQL com um dos seus PEs.



não enumeração do plano ótimo pela técnica de exploração, por erros nas técnicas de estimação, entre outros.

O custo de um PE está associado indiscutivelmente com a quantidade de dados que este necessitará processar. Como estamos no contexto relacional, torna-se necessário, então, a associação do custo com o conceito de cardinalidade, visto que este está relacionado com a quantidade de tuplas, as quais representam os dados a serem processados. Processar tuplas, nesse contexto, significa buscar as páginas que estão armazenadas nos arquivos do SGBD, colocá-las na memória principal e, por fim, executar o processamento. A cardinalidade de um operador relacional é a quantidade de tuplas que são geradas por ele, ou seja, é a quantidade de tuplas que este dará como saída após realizar a sua operação. Ademais, estende-se esse conceito para um PE como sendo a cardinalidade do seu nó raiz, haja vista que um PE é composto por vários operadores relacionais e o nó raiz representa a última operação a ser executada em todo o processamento. Muito embora os PEs equivalentes de uma consulta possuam a mesma cardinalidade, não é garantido que as cardinalidades dos nós intermediários sejam iguais em todos os planos. Dessa forma, o custo de cada PE pode variar devido a essas diferenças. Posto isso, fica claro que a cardinalidade é um parâmetro que deverá ser fornecido como entrada para o processo de otimização de consultas devido ao fato do custo de um PE variar de acordo com a quantidade de tuplas em que este terá de processar.

Para tanto, várias estimativas de cardinalidade são calculadas com o intuito de possibilitar essa mensuração de custo, dado que não é possível conhecer a cardinalidade de todas as operações relacionais dos planos de execução a priori, pois a cardinalidade dessas operações dependem das características presentes nas bases de dados do SGBD. Como o tempo de execução da consulta submetida depende da qualidade do PE escolhido, a precisão dessas estimativas pode impactar fortemente no tempo de execução de uma consulta.

1.1 Motivação e Caracterização do Problema

A despeito da existência de várias técnicas para calcular as estimativas de cardinalidade já propostas na literatura, uma gama de trabalhos já mostrou que os atuais SGBDs, que fazem uso dessas técnicas, ainda apresentam uma qualidade de estimação baixa devido à introdução de erros (ou imprecisões) provenientes das estimativas de cardinalidade, sobretudo quando a carga de trabalho envolve consultas complexas (IOANNIDIS; CHRISTODOULAKIS, 1991; MOERKOTTE *et al.*, 2009; GRAEFE *et al.*, 2012; LEIS *et al.*, 2015; LEIS *et al.*, 2018). As principais fontes de erros são: dados estatísticos desatualizados, *Independência de Valores de Atributos* (IVA), premissa da distribuição uniforme e *Hipótese da Contenção* (HC). Elas serão analisadas mais profundamente no Capítulo 2, mas já se adianta que elas ocorrem devido à maneira pela qual os SGBDs tratam os dados das suas bases. De acordo com (LEIS *et al.*, 2015), os erros introduzidos pelo modelo de custo subjacente são limitados. Mesmo que um modelo simplista, que não reflete completamente a realidade, seja utilizado para modelar o custo de execução de um plano, ainda sim a quantidade de erros introduzidos devido a essa escolha não afeta tanto no processo de otimização de consultas. No entanto, podem ocorrer muitos erros nas estimativas de cardinalidade. Na verdade, segundo (IOANNIDIS; CHRISTODOULAKIS, 1991), a propagação desses erros acontece de forma exponencial, dificultando, dessa forma, ainda mais a tarefa de seleção de um PE eficiente por parte do otimizador de consultas baseado em custo dos SGBDs.

A partir do cenário apresentado, constata-se que o estudo do tema aqui apresentado, especificamente nos sistemas que têm como base o modelo relacional, é relevante, tendo em vista tratar-se de problemática de grande importância na literatura de BD. Por conseguinte, esta dissertação visa abordar o Problema 1 declarado a seguir e propor uma nova técnica para o processo de geração das estimativas de cardinalidade dos operadores dos PEs de uma consulta. Esta técnica faz uso de um modelo de aprendizagem de máquina de modo a obter uma maior precisão nas estimativas de cardinalidade no cenário de otimização de consultas em SGBDs, especificamente nos relacionais.

Problema 1. ***Como calcular as estimativas de cardinalidade nos SGBDs utilizando aprendizagem de máquina?***

1.2 Hipótese de Pesquisa

A aprendizagem de máquina é uma subárea da Inteligência Artificial que tem como foco o desenvolvimento e a evolução de técnicas, métodos e modelos capazes de detectar padrões à medida que são apresentados aos dados (ALPAYDIN, 2009). É visível na literatura o interesse na integração dessas técnicas nos sistemas como os SGBDs em virtude de trazer a esses programas a capacidade de adaptar as suas estruturas internas de acordo com as características dos dados que eles processam (MARCUS; PAPAEMMANOUIL, 2019; WANG *et al.*, 2019). No contexto desta dissertação, busca-se construir uma abordagem baseada nessas técnicas a fim de evitar as fontes de erros que foram listadas anteriormente e, a partir disso, gerar estimativas de cardinalidade com uma maior acurácia. Com isso, como apresentado na Hipótese 1, consideramos que o uso dessa técnica resultará em um menor erro das estimativas e, consequentemente, planos de execução mais eficientes serão escolhidos e, portanto, o desempenho dos SGBDs será maior.

Hipótese de Pesquisa 1. ***Por meio do uso de técnicas de aprendizagem de máquina, os SGBDs podem melhorar a acurácia das estimativas de cardinalidade levando em consideração não somente as estatísticas básicas já presentes nos SGBDs atuais, mas também considerando as propriedades dos dados que são capturadas pelos modelos de aprendizagem. Dessa forma, espera-se que o plano de consulta a ser executado seja mais eficiente, diminuindo assim o tempo de execução.***

Por fim, este trabalho realizou uma avaliação experimental com o objetivo de avaliar tanto a eficiência da proposta quanto o gargalo que esta poderá trazer no momento em que for inserida em um SGBD.

1.3 Objetivos

Com a finalidade de superar as fontes de erros dos estimadores de cardinalidade atuais, o objetivo geral do trabalho aqui apresentado é a definição e o desenvolvimento de uma nova abordagem para o cálculo das estimativas de cardinalidade presentes no processo de otimização de consultas dos SGBDs. Esta abordagem deve orientar o motor de execução desses sistemas a realizar a escolha correta do plano de execução com o menor custo.

Tendo o objetivo geral definido, são estabelecidos os seguintes objetivos específicos de modo a possibilitar que o objetivo geral proposto seja alcançado:

- Investigar uma técnica de aprendizagem de máquina adequada para o cenário em que o processo de estimar as cardinalidades está inserido;
- Definir uma abordagem e arquitetura para gerar as estimativas de cardinalidade baseada em aprendizagem de máquina;
- Avaliar a abordagem proposta com o intuito de aferir a precisão da técnica e o tempo necessário para gerar o resultado.

1.4 Contribuições

No decorrer do desenvolvimento deste trabalho, várias contribuições de pesquisa foram levantadas, das quais vale pontuar as seguintes:

- Uma descrição das possíveis fontes de erros presentes nos estimadores de cardinalidade tradicionais;
- Uma abordagem baseada em aprendizagem de máquina para o cálculo das estimativas de cardinalidade;
- Uma arquitetura para a incorporação da abordagem apresentada;
- Uma experimentação no SGBD PostgreSQL da abordagem proposta para validação da Hipótese de Pesquisa 1.

Já no contexto da produção científica, vale destacar as seguintes contribuições que foram apresentadas em publicações científicas aprovadas em eventos da área de Banco de Dados.

- AMORA, P. R. P.; TEIXEIRA, E. M.; PRACIANO, F. D. B. S.; MACHADO, J. C. Smartltm: Smart larger-than-memory storage for hybrid database systems. In: XXXIII Simpósio Brasileiro de Banco de Dados, SBBD 2018, Rio de Janeiro, RJ, Brazil, August 25-26, 2018. [s.n.], 2018. p. 13–24.
- LIMA, M. I. V.; FARIAS, V. A. E. de; PRACIANO, F. D. B. S.; MACHADO, J. C. Workload-aware parameter selection and performance prediction for in-memory databases. In: XXXIII Simpósio Brasileiro de Banco de Dados, SBBD 2018, Rio de Janeiro, RJ, Brazil, August 25-26, 2018. [s.n.], 2018. p. 169–180.
- PRACIANO, F. D. B. S.; SOUSA, J. F. L. de; MACHADO, J. C. Uma análise experimental da utilização de diferentes tecnologias de armazenamento em um sgbd relacional. In: Anais do XXXIV Simpósio Brasileiro de Banco de Dados. Porto Alegre, RS, Brasil: SBC, 2019. p.259–264.

É importante notar que mesmo que as duas primeiras publicações apresentadas

acima não estejam incluídas totalmente no âmbito do trabalho apresentado nessa dissertação, muito dos conceitos basilares que foram desenvolvidos e apresentados naqueles trabalhos foram significativos para a evolução desta dissertação.

Além das publicações já apresentadas, estamos trabalhando na redação de um artigo científico que irá descrever as técnicas desenvolvidas neste trabalho, além de apresentar e discutir os resultados científicos aqui alcançados. Nosso planejamento é submeter este novo artigo ao jornal científico *Journal of the Brazilian Computer Society* (JBCS).

1.5 Organização

Acerca da organização do texto desta dissertação, buscou-se seguir a seguinte estrutura:

- No Capítulo 2, são pontuados e apresentados inicialmente o embasamento fundamental ao estudo do trabalho que será desenvolvido adiante, vislumbrando principalmente apresentar os conceitos tanto do processamento de consultas quanto da aprendizagem de máquina;
- No Capítulo 3, busca-se relacionar e descrever os trabalhos mais relevantes que possuem uma similaridade com o método desenvolvido nesse texto, além daqueles mais tradicionais. A partir disso, é apresentada uma discussão sobre essas técnicas juntamente com o posicionamento do nosso trabalho em relação a eles;
- Já o Capítulo 4 aborda o método proposto nessa dissertação, pormenorizando os seus detalhes intrínsecos com o intuito de explanar o funcionamento do método com clareza;
- Posteriormente, no Capítulo 5, avalia-se o método proposto de maneira experimental diante de pontos de vistas distintos de modo a fornecer suporte às conclusões alcançadas nesse trabalho;
- Por fim, no Capítulo 6, apresentam-se as principais conclusões alcançadas no âmbito desta pesquisa. Ademais, direcionamentos para uma possível extensão do método proposto a ser realizado em trabalhos futuros são apresentados.

2 FUNDAMENTAÇÃO TEÓRICA

Apresentam-se, neste capítulo, os principais conceitos que estão envolvidos no processamento de consultas em SGBDs e na área de aprendizagem de máquina de modo a embasar o restante deste trabalho. Além do mais, as principais fontes de erros dos atuais SGBDs nas estimativas de cardinalidade são descritas, juntamente com o modelo de aprendizagem de máquina que será utilizado no Capítulo 4.

Inicialmente, a seção 2.1 mostra alguns conceitos e formalizações necessárias do modelo relacional, permitindo que uma linguagem mais específica seja utilizada nos próximos capítulos. Em seguida, na seção 2.2, uma visão geral acerca da maneira pela qual os SGBDs respondem às consultas dos usuários é apresentada juntamente com as três etapas envolvidas no processamento de consultas, dando ênfase na fase de otimização. Posteriormente, na seção 2.3, são apresentados os aspectos relativos à área de aprendizagem de máquina e o funcionamento do algoritmo *Light Gradient Boosting Machine* (*LightGBM*) (KE *et al.*, 2017), responsável por gerar o modelo que será utilizado no método proposto, é apresentado em detalhes.

2.1 Modelo Relacional

O modelo relacional é o modelo de dados mais utilizado pelos principais SGBDs devido à sua simplicidade e à sua fundamentação teórica pujante (STONEBRAKER; HELLERSTEIN, 2005). Não se busca nesse texto a apresentação dos conceitos desse modelo na sua totalidade, uma vez que tais conceitos são muito bem descritos em livros-texto, por exemplo (ELMASRI; NAVATHE, 2000). Assim, descreveremos a seguir brevemente os conceitos mais importantes para o contexto desta dissertação. Além disso, para detalhar o problema das estimativas de cardinalidade abordado neste trabalho de um modo preciso, faz-se necessário, portanto, a adoção de uma notação matemática padronizada. Na literatura, há variadas notações para o tópico de otimização de consultas. Resolvemos seguir uma notação matemática semelhante àquela utilizada por (KÖNIG, 2001).

No contexto relacional, os dados são organizados em tuplas de um conjunto $\mathcal{R}$ de $n \in \mathbb{N}$ relações, as quais iremos representar com a seguinte notação: $\mathcal{R} = \{R_1, R_2, ..., R_n\}$. Por sua vez, cada relação $R_i \in \mathcal{R}$ é composta por um conjunto $\mathcal{A}$ de $m \in \mathbb{N}$ atributos cuja representação será igual a $\mathcal{A}_{R_i} = \{R_i.A_1, R_i.A_2, ..., R_i.A_m\}$. Em relação aos possíveis valores que um atributo $A_j \in R_i$ qualquer pode armazenar, existe o conceito de domínio $\mathcal{D}_{A_j}$ do atributo a fim de definir quais os

valores que poderão ser guardados nesse atributo. Além disso, há também o conjunto $\mathcal{V}_{A_j} \subseteq \mathcal{D}_{A_j}$ que representa os valores que estão presentes realmente nos dados armazenados no atributo A_j. Ademais, ressalta-se que cada tupla $T_k \in R_i$ é composta por m valores, sendo um valor para cada um dos atributos da relação R_i. O valor do atributo A_j de uma tupla T_k será representado por $T_k.A_j$. Diferenciam-se, portanto, as notações $R_i.A_j$ e $T_k.A_j$. Enquanto esta representa o valor do atributo A_j de somente uma única tupla T_k, aquela representa todos os valores presentes no atributo A_j da relação R_i, ou seja, as notações $R_i.A_j$ e $\mathcal{V}_{R_i.A_j}$ são intercambiáveis. Um exemplo de uma instância de uma relação *Empregado* de um possível BD cujo objetivo é modelar os dados de uma empresa é apresentado na Tabela 1, e esta relação única do BD tem o conjunto de atributos $\mathcal{A}$ = {Id, Nome, Idade, Salário}. Essa relação modela os dados dos possíveis funcionários da base de dados de uma dada empresa. Por exemplo, se uma pessoa deseja saber quais os salários que existem nessa base de dados, basta consultar o conjunto $\mathcal{V}_{Empregado_{salário}}$.

As informações armazenadas nas relações $\mathcal{R}$ são acessadas por meio dos chamados operadores relacionais, sendo alguns deles: seleção, junção, projeção e produto cartesiano. Utilizaremos a notação padrão presente na literatura para cada um dos operadores, ou seja, σ, $\bowtie$, π e $\times$, respectivamente. Destaca-se que, neste trabalho, os principais são os operadores de seleção e junção por estarem intimamente relacionados ao processo de geração das estimativas de cardinalidade.

Tabela 1 – Exemplo de uma relação Empregado com quatro atributos.

Id	**Nome**	**Idade**	**Salário**
1	João	35	5.000
2	Maria	49	7.500
3	Morgana	18	1.500
4	José	22	2.200

2.2 Processamento de Consultas

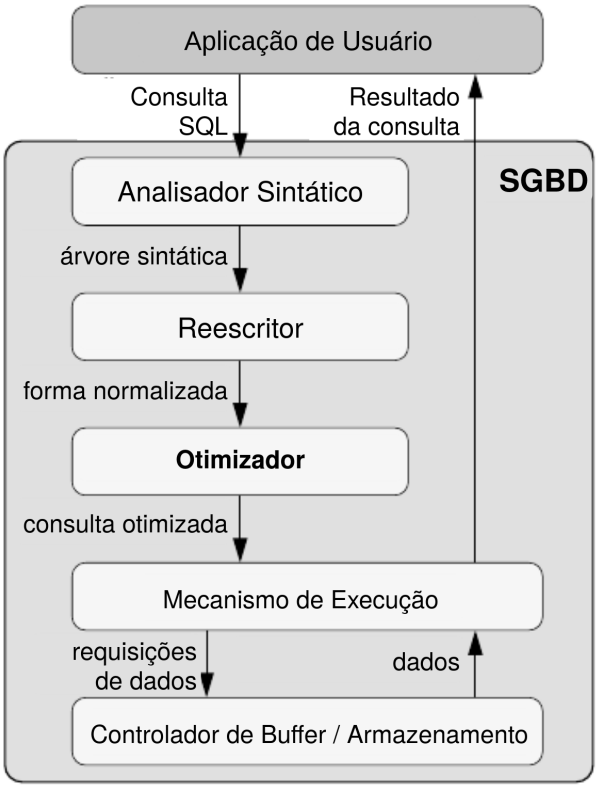
2.2.1 Visão Geral

As consultas submetidas pelos usuários aos SGBDs são declarativas, de alto nível, sendo necessário realizar três etapas para executá-las, como mencionado anteriormente. Dentro de cada uma dessas etapas, vários passos intermediários são essenciais a fim de que as consultas possam ser executadas e, assim, a resposta seja gerada para o usuário. Na primeira etapa, a de processamento, ocorre o processo de tradução de uma linguagem de alto nível para uma

expressão equivalente na álgebra relacional. Esta, por sua vez, é dada como entrada para a segunda etapa, a de otimização, com o propósito de que seja otimizada. Por fim, é na terceira etapa, a de execução, que a consulta é realmente executada, gerando o resultado final para o usuário. Iremos abordar nas próximas subseções cada uma dessas etapas, mas antes vamos apresentar uma visão genérica sobre a maneira pela qual os SGBDs realizam tais tarefas.

Por serem softwares complexos, os SGBDs são desenvolvidos de modo modular com vistas à repartição dos problemas em diferentes módulos, fazendo com que tanto o entendimento quanto o desenvolvimento da solução dos problemas sejam facilitados. A Figura 4 apresenta uma arquitetura simplificada e genérica criada a partir da tentativa de generalizar as arquiteturas utilizadas pelos principais SGBDs. Por completude, vamos descrever sucintamente cada módulo, ressaltando aqueles que estão diretamente relacionados com o processamento de consultas. Na Figura 4, as setas do lado esquerdo são utilizadas para simbolizar o caminho percorrido pelas consultas no momento em que estas são submetidas ao SGBD até a apresentação do resultado final para o usuário. Isto é, a consulta submetida pela aplicação do cliente caminhará, entre os módulos, na seguinte ordem: Analisador Sintático, Reescritor, Otimizador, Mecanismo de Execução, Controlador de *Buffer* / Armazenamento.

Figura 4 – Arquitetura simplificada dos SGBDs.



Fonte: Adaptado de (ZUKOWSKI, 2009).

Iniciaremos descrevendo os dois primeiros módulos, Analisador Sintático e Reescritor, os quais são responsáveis por transformar a consulta do usuário escrita em uma linguagem de alto nível declarativa para uma representação interna denominada de árvore sintática que servirá de base para todas as fases posteriores. Para tanto, as etapas de análise léxica, sintática e semântica são implementadas nesses módulos. Ademais, outras transformações (reescritas) são realizadas a fim de padronizar uma forma normal de consultas representadas internamente nos SGBDs. Por exemplo, os predicados de uma consulta devem estar na *Forma Normal Conjuntiva* (FNC) (ELMASRI; NAVATHE, 2000).

Seguindo a ordem, chegamos ao módulo Otimizador. Neste, realiza-se a principal e a mais dificultosa atividade: geração e otimização dos PEs equivalentes à consulta submetida pelo usuário. Necessita-se que esta geração seja realizada em um tempo pequeno, visto que essas atividades serão realizadas à medida que as consultas são submetidas e, posteriormente, executadas. Por conta disso, várias técnicas foram propostas na literatura visando diminuir o tempo de execução da geração dos PEs, sendo a mais utilizada a técnica baseada em programação dinâmica. Além disso, é nesse módulo que se encontra o processo de estimar o custo de cada PE que, por sua vez, envolve as estimativas de cardinalidade dos operadores.

Ulteriormente à geração do PEs, o módulo Mecanismo de Execução é responsável por realizar a execução propriamente dita do PE escolhido. É nesse módulo que o resultado final da consulta será produzido. Portanto, faz-se necessária a garantia de que o resultado seja gerado corretamente. Há muitos modelos de execução na literatura que visam diminuir o tempo de execução das consultas. Não obstante, o Volcano (GRAEFE, 1994) é o modelo mais aceito pela comunidade científica e o PostgreSQL, o SGBD que será utilizado nos experimentos, o implementa. Por conseguinte, falaremos mais adiante desse modelo específico.

Por fim, chegamos ao módulo Controlador de *Buffer* / Armazenamento. Implementa-se, nesse módulo, os recursos necessários para a manipulação dos dados, isto é, as informações de um BD são acessadas por meio de uma interface fornecida por esse módulo. Neste trabalho, não trataremos essa parte em detalhes, uma vez que o foco aqui dado está relacionado, particularmente, com as técnicas de otimização de consultas.

2.2.2 Etapa I: Processamento

Inicialmente, na fase de processamento, são efetuadas as análises léxica, sintática e semântica. Na primeira, realiza-se o agrupamento dos caracteres presentes no texto da consulta

de modo a produzir os conhecidos itens léxicos (ou *tokens*). Posteriormente, a identificação e a verificação desses *tokens* de acordo com as palavras-chaves definidas na SQL são executadas. Dando prosseguimento, a análise sintática vai verificar se a sintaxe presente na consulta está de acordo com a gramática da SQL, isto é, se aqueles *tokens* estão na sequência definida pelas regras sintáticas da linguagem. Por fim, a análise semântica é realizada com o intuito de averiguar se as relações e os atributos referenciados pela consulta são semanticamente válidos, ou seja, se realmente existem no catálogo armazenado no SGBD. Para tanto, uma tabela de símbolos é guardada para manter todos os objetos presentes no catálogo associados ao seu identificador, possibilitando a verificação da existência de um dado identificador ou não. Ao mesmo tempo em que se realiza a análise semântica, é efetuada também a etapa de normalização (e fatorização) cujo objetivo é lidar com a formatação estrutural da consulta, isto é, adicionar novas variáveis para representar subexpressões, remover possíveis redundâncias e colocar os fatores na FNC.

Depois que essas etapas acima foram realizadas, então a consulta agora é representada por uma *Árvore Sintática Abstrata* (ASA), derivada da gramática utilizada na análise sintática. Posteriormente, esta árvore é transformada em uma expressão equivalente da álgebra relacional, a qual estará organizada internamente por uma *Estrutura de Dados* (ED), sendo árvore a mais utilizada pelos SGBDs. Isto é, cada consulta será representada por uma expressão relacional equivalente organizada em árvore, sendo que cada um dos nós dessa árvore é uma das operações da álgebra relacional. Na literatura, esta expressão é denominada de *Plano de Consulta* (PC) (às vezes chamadas também de Plano Lógico de Consulta) tendo em vista diferenciá-la do PE (ou Plano Físico de Consulta). Enquanto que no PC não se possui informações sobre a implementação dos operadores da álgebra relacional, no PE encontram-se todas as informações necessárias para a real execução da consulta, isto é, neste plano já se tem a informação de quais algoritmos serão utilizados para executar os operadores relacionais, em qual ordem serão processadas as junções, etc. Em suma, o PE difere do PC no nível de detalhamento. Naquele, já se possui todas as informações de execução, enquanto que neste algumas minúcias são abstraídas, abordando somente os detalhes lógicos. Finalmente, depois que o PC foi obtido, dá-se início à etapa de otimização, que será abordada na próxima subseção.

Para fechar a presente subseção e aproveitando que estamos abordando a etapa de processamento que é responsável por lidar com a estrutura das consultas SQL, é importante definir estruturalmente quais os tipos de consultas SQL que serão considerados neste trabalho. Para isso, a Definição 1 apresenta o conceito de consultas *Seleção-Projeção-Junção* (SPJ).

Definição 1. Uma consulta SPJ Q escrita na SQL é composta por no máximo três blocos, a saber: *SELECT*, *FROM* e *WHERE*. No primeiro bloco, definem-se os atributos que estarão presentes no resultado da consulta, isto é, operação de projeção. O segundo bloco conterá as relações que envolvam os dados requisitados pela consulta, ou seja, representa a operação de produto cartesiano entre as relações. Por fim, no bloco *WHERE* serão definidas as condições que definem se um determinado dado deverá ou não entrar no resultado, ou seja, operação de seleção. Um exemplo é demonstrado na Figura 3. Nesta, um *Plano de Execução* (PE) é apresentado em que duas operações são realizadas: seleção e projeção.

A definição 1 é importante visto que ela serve para restringir o tipo de consultas SQL que estará sendo abordada em todas as partes deste trabalho, uma vez que a expressabilidade da SQL é muito ampla, permitindo que várias consultas com estruturas diferentes sejam consideradas válidas. Pode-se considerar que as consultas SPJ são aquelas que possuem os blocos mais básicos da linguagem. Não obstante, muita das consultas utilizadas em ambiente de produção possuem pelo menos os três blocos das consultas SPJ. Ademais, é necessário pontuar que somente os blocos *SELECT* e *FROM* são obrigatórios, pois é possível que uma consulta SPJ não requeira que certas cláusulas da operação de seleção sejam impostas aos dados do resultado da consulta. Além disso, considera-se como cláusulas de seleção os predicados cuja forma é a seguinte $R_i.A_j$ θ v, em que $\theta \in \{<, \leq, =, \neq, >, \geq\}$ e v representa um valor constante. Mais de uma cláusula é permitida em uma consulta, desde que elas estejam ligadas por um operador de conjunção, ou seja, somente predicados conjuntivos são aceitos. Por fim, no que tange aos predicados de junção, são permitidos aqueles que são do tipo *equi-join*, isto é, $R_{i_1}.A_{j_1} = R_{i_2}.A_{j_2}$.

2.2.3 Etapa II: Otimização

Dado que a consulta escrita em SQL já foi processada e transformada em uma expressão equivalente da álgebra relacional, partimos do ponto em que um PC inicial já está pronto. Para cada consulta, é possível haver vários planos diferentes em relação à maneira em que os dados são processados, mas gerando o mesmo resultado no final. Existem duas estratégias para gerar planos equivalentes, a transformativa e a sintética (ELMASRI; NAVATHE, 2000). Na primeira abordagem, inicia-se de um plano básico e algumas transformações são aplicadas a fim de que um novo plano seja obtido. Já em relação à sintética, um plano é construído do início a partir da adição de um operador relacional por vez até que todos os operadores da consulta tenham sido adicionados no plano. Nota-se que essas duas estratégias são utilizadas

pelos SGBDs no módulo de otimização. Ademais, cada um desses planos alternativos pode possuir um custo de processamento diferente, sendo que este custo pode impactar diretamente no desempenho. Assim, há de se escolher aquele que apresenta o menor custo.

De fato, essa escolha seria possível caso o custo de cada um dos diferentes planos pudesse ser calculado previamente. Entretanto, calcular esse custo é dispendioso caso os planos realmente precisem ser executados. Uma alternativa viável e bastante utilizada na prática é o uso de heurísticas que fornecem algumas melhorias na forma de processar os dados. Dessa forma, nessa fase de otimização, os SGBDs buscam aplicar as conhecidas equivalências da álgebra relacional. Algumas estão relacionadas na Tabela 2, juntamente com algumas heurísticas a fim de que um PC otimizado seja aquele que apresente uma melhor maneira de processar os dados requisitados, isto é, o custo do plano escolhido é um dos menores diante daqueles possíveis. É importante ressaltar que nesse estágio não há ainda a preocupação de qual algoritmo utilizar para cada uma das operações, em qual ordem aplicá-las, etc. Além disso, é necessário afirmar também que se um novo PC é gerado a partir de um outro plano válido através da aplicação de uma das equivalências, tem-se que este novo plano também é válido, pois a transformação aplicada não altera o resultado final. Em suma, nesta fase de otimização são utilizadas essas equivalências conjuntamente com as heurísticas de tal sorte que o PC final represente uma das melhores alternativas para realizar o processamento da consulta.

Tabela 2 – Exemplos de equivalências da álgebra relacional.

Equivalência	**Nome**
$R \cup S = S \cup R$ $R \cap S = S \cap R$	Comutatividade das operações de união e interseção
$R\theta(S\,\theta T) = (R\theta S)\theta T$, no qual $\theta \in \{\bowtie, \times, \cup, \cap\}$	Associatividade das operações de junção, produto cartesiano, união e interseção
$\sigma(R\theta S) = \sigma(R)\theta\sigma(S)$, em que $\theta \in \{\cup, \cap\}$	Comutatividade da seleção e operações de união e interseção
$\pi(R\theta S) = \pi(R)\theta\pi(S)$, em que $\theta \in \{\cup, \cap\}$	Comutatividade da projeção e operações de união e interseção

Com um PC otimizado em mãos, o próximo passo em direção à execução das consultas é realizar as escolhas de como executar cada uma das operações relacionais presentes neste plano, definir a ordem de execução, os métodos de acesso, entre outras decisões e, por fim, ter o plano de execução da consulta. Para tanto, os SGBDs implementam um algoritmo de programação dinâmica e alguns deles também implementam outros, tais como algoritmos genéticos, que irá explorar o grande espaço de busca e, dessa forma, enumerar os possíveis planos

de execução. É importante ressaltar que esse problema de escolha do PE ótimo é conhecido por ser NP-completo (IBARAKI; KAMEDA, 1984).

Dado que um conjunto de PEs equivalentes foi calculado, agora resta a etapa de escolher o PE com o menor custo de processamento. Como dito anteriormente, é necessário executar cada um dos PEs de modo a obter o custo associado individualmente. Todavia, não é plausível que todos os PEs sejam executados para se encontrar o melhor plano. Com isso, esse custo vai ser estimado a partir de estatísticas presentes no catálogo dos SGBDs, como veremos a seguir.

2.2.3.1 Estimativas de Custo

Visando guiar o processo de avaliar e escolher um PE dentre o conjunto de planos equivalentes, um modelo de custo é adotado nos SGBDs. Esse modelo tem como finalidade capturar a correlação existente entre o custo computacional das operações presentes em um PE e o tempo de execução que um dado PE vai levar para ser executado, e, finalmente, gerar o resultado da consulta submetida (ELMASRI; NAVATHE, 2000). Há uma variedade de propostas de modelo de custos na literatura. Devido à nossa opção de utilizar o PostgreSQL como o SGBD que será usado na avaliação experimental, vamos apresentar o seu modelo de custo, o qual possui algumas similaridades com os demais presentes em outros sistemas.

Por ser um SGBD tradicional, isto é, que adota uma interface de armazenamento secundário baseada em disco, o custo computacional de uma operação relacional será uma média ponderada entre o custo de processamento e o custo das operações de leitura e escrita que serão realizadas no dispositivo de armazenamento secundário. Ou seja, o custo de execução de um operador é definido, neste modelo, como sendo a soma ponderada da quantidade de páginas de disco acessadas para realizar a operação juntamente com a quantidade de instruções de máquina necessárias para processar os dados que estão presentes na memória principal. O ponderamento dessa soma é ajustada pelos *Administradores de Bancos de Dados (DBAs)*, sendo que uma configuração padrão genérica é fornecida pelos SGBDs. Ademais, pontua-se que esse ponderamento é realizado com o intuito de refletir as diferenças de impacto no desempenho existentes entre as operações de Unidade Central de Processamento (CPU), as operações de leituras e escritas sequenciais, assim como as aleatórias (LEIS *et al.*, 2015).

Dado um operador relacional O, o seu custo computacional t_O, ou seja, seu tempo

de execução, será estimado pela seguinte equação:

$$t_O = N_{seq} \cdot C_{seq} + N_{ale} \cdot C_{ale} + N_T \cdot C_T + N_{idx} \cdot C_{idx} + N_O \cdot C_O \tag{2.1}$$

O que cada um desses fatores representa é apresentado na Tabela 3. Como um PE é composto por uma sequência de operações relacionais, consequentemente seu tempo de execução t_{PE} será o somatório do custo de cada uma das operações, ou seja:

$$t_{PE} = \sum_{O \in PE} t_O \tag{2.2}$$

Tabela 3 – Descrição dos fatores da equação 2.1.

Fator	Descrição
N_{seq}	Número de páginas acessadas sequencialmente
N_{ale}	Número de páginas acessadas aleatoriamente
N_T	Número de tuplas a serem processadas
N_{idx}	Número de acessos a estrutura de índice
N_O	Número de instruções
C_{seq}	Custo de acesso sequencial
C_{ale}	Custo de acesso aleatório
C_T	Custo de processar uma tupla
C_{idx}	Custo de processar uma tupla pela estrutura de índice
C_O	Custo de processar uma operação

Finalmente, define-se o tempo de execução t_Q de uma consulta Q como sendo o custo do PE que apresenta o menor tempo de execução dentre os PEs equivalentes que foram gerados, ou seja:

$$t_Q = \min_{t_{PE}} \{PE_1, PE_2, ..., PE_k\} \tag{2.3}$$

Com isso, é possível observar que o tempo de execução é dependente de duas variáveis, a saber: a quantidade de dados que serão processados representada pelos vários N's, relacionados com a cardinalidade, e o custo de processamento das operações representado por cada um dos C's. Ou seja, essas duas variáveis precisam ser conhecidas a priori a fim de possibilitar que o custo de cada um dos PEs seja avaliado e, por fim, o plano mais eficiente seja escolhido. Como dito acima, os valores de C's são definidos pelos DBAs ou já vêm por padrão. Porém, não há como saber quais os valores reais de N's antes mesmo de executar os PEs. É nesse momento que o problema abordado nesse trabalho se encontra. Para escolher um PE, os SGBDs precisam calcular um grande número de estimativas de cardinalidade que dependem da estrutura da consulta submetida, como será abordado adiante.

2.2.3.2 Estimativas de Cardinalidade

A etapa de estimativas de cardinalidade visa responder perguntas do tipo: "Quantas tuplas satisfazem uma certa condição?"de modo a conhecer a quantidade de tuplas que uma operação terá como saída. Desta forma, vamos iniciar essa subseção definindo o conceito de cardinalidade.

Definição 2. A cardinalidade de um operador relacional O presente em um plano de execução é a quantidade de tuplas que o operador dará como saída no momento da execução da consulta e será representado por $c(O)$. Ademais, representa-se a estimativa usando a notação $\hat{c}(O)$. Quando o operador O for conhecido, pode-se omiti-lo e utilizar somente c e $\hat{c}$.

Apresentado o conceito acima, agora podemos relacioná-lo aos efeitos que as operações relacionais geram nas suas cardinalidades de entrada. Iniciando pela operação de projeção, temos que não há nenhuma mudança na cardinalidade fornecida como entrada, ou seja, a quantidade de saída será igual a de entrada, pois ao realizar uma projeção em um determinado conjunto de dados é obtido como resultado a remoção de alguns atributos, porém nenhuma tupla é removida, a menos que se considere a exclusão de duplicatas. Neste caso, a projeção poderá influenciar na cardinalidade entrada. Como não iremos abordar a remoção de duplicatas neste trabalho, então iremos considerar que essa operação não altera a cardinalidade quando aplicada. Restam-nos, neste momento, relacionar a cardinalidade com as operações de seleção, produto cartesiano e, por fim, junção.

Em relação à operação de seleção, uma consulta SPJ pode possuir diversos predicados no seu bloco de seleção (*WHERE*), os quais podem ser vistos como filtros de tuplas, isto é, cada um dos predicados realizará uma filtragem nas tuplas de tal maneira que somente aquelas que satisfazem todos os predicados irão aparecer no resultado final da consulta. Como a cardinalidade está atrelada à quantidade de tuplas que o resultado de cada operação relacional terá, assim a operação de seleção apresenta uma influência na cardinalidade, pois algumas tuplas podem ser removidas do conjunto de dados fornecido como entrada. Para avaliar esse impacto, necessita-se da definição do conceito de seletividade, o qual é apresentado na definição a seguir.

Definição 3. A seletividade de uma dada consulta Q ou de um predicado $\mathcal{P} = p_1 \ \& \ p_2 \ \& \ ... \ \& \ p_l$ ($l \in \mathbb{N}$) é definida como sendo a fração de tuplas que permanecem no resultado após a aplicação da consulta ou do predicado. Utilizaremos os símbolos $s(Q)$ e $s(\mathcal{P})$ para representar a seletividade,

respectivamente, para consulta ou predicado. Na literatura, encontramos o termo fator de filtragem como sinônimo de seletividade.

O processo da operação de seleção está relacionado com a seletividade, pois ela define a porção de tuplas que satisfaz a uma dada consulta. Por sua vez, para calcular a seletividade é necessário ter a priori informações sobre como os dados estão distribuídos, isto é, necessita-se conhecer as estatísticas dos dados, por exemplo a frequência dos valores presentes em um dado atributo. A frequência f_v associada a um valor v que aparece em um atributo A_j qualquer, isto é, $v \in \mathcal{V}_{A_j}$, de uma relação R_i é determinada como sendo a quantidade de tuplas de R_i presentes no BD cujo valor do atributo A_j é v. Tendo o conhecimento das frequências, é possível construir a distribuição dos dados, unidimensional ou multidimensional, sendo cada uma formalizada nas duas definições abaixo.

Definição 4. A distribuição unidimensional dos dados de um único atributo $R_i.A_j$ é definida como sendo um conjunto de pares $\mathcal{T}_{R_i.A_j} = \{(v_1, f_1), (v_2, f_2), ..., (v_q, f_q)\}$ em que cada par (v_k, f_k) é composto pelo valor $v_k \in \mathcal{V}_{R_i.A_j}$ juntamente com a sua frequência associada f_k. Vale ressaltar que q é a quantidade de valores distintos presentes no atributo $R_i.A_j$.

Definição 5. A distribuição multidimensional (*joint*) dos dados de d atributos $R_i.A_{j_1}, R_i.A_{j_2}, ..., R_i.A_{j_d}$ é definida como sendo um conjunto de pares $\mathcal{T}_{R_i.A_{j_1}, R_i.A_{j_2}, ..., R_i.A_{j_d}} = \{(v_1, f_1), (v_2, f_2), ..., (v_q, f_q)\}$ em que cada par (v_k, f_k) é composto pelo valor $v_k \in \bigtimes_{z=1}^{d} R_i.A_{j_z}$ juntamente com a sua frequência associada f_k.

De posse das distribuições dos dados, as seletividades podem ser conhecidas com a ajuda da frequência que cada valor possuir nos dados armazenados no SGBD. Se mais de um atributo for usado no predicado de seleção $\mathcal{P}$ de uma consulta Q, será necessário usar a distribuição multidimensional dos dados $\mathcal{T}$. Com essa informação, é possível calcular a cardinalidade de uma operação de seleção O através da seguinte equação na qual $c_{entrada}$ representa a cardinalidade de entrada:

$$c(O) = c_{entrada} \cdot s(\mathcal{P}) \tag{2.4}$$

Por fim, chegamos nas operações de produto cartesiano e junção. No contexto relacional, o produto cartesiano vai receber duas entradas, as relações R_1 e R_2, e terá como saída todas as combinações possíveis entre as tuplas de R_1 com as de R_2. Ou seja, a cardinalidade desse operador vai ser a multiplicação da cardinalidade de R_1 com a de R_2. De maneira mais

genérica, a cardinalidade de saída será a multiplicação das cardinalidades de entrada. Desta forma, é necessário também utilizar a distribuição dos dados a fim de conhecer as cardinalidades fornecidas como entrada. Por outro lado, a junção pode ser definida como sendo uma operação de produto cartesiano seguida de uma operação de seleção. Por conseguinte, a informação da distribuição dos atributos de junção será suficiente para encontrar o impacto que essa operação terá nas cardinalidades dadas na entrada da operação.

Tendo o conhecimento da distribuição dos dados, os SGBDs são capazes de calcular o custo dos PEs. Portanto, o que se busca nesses sistemas é construir uma estrutura que seja eficiente tanto no uso de memória quanto no processamento cujo objetivo é realizar uma aproximação da distribuição dos dados presentes em cada relação, uma vez que não é viável, na prática, manter estruturas que reflitam totalmente a distribuição dos dados. Desta forma, estimativas de cardinalidades são feitas pelos SGBDs usando essas estruturas subjacentes. Vejamos um exemplo do processo de estimar as cardinalidades da consulta apresentada na Figura 3.

Suponhamos que a relação Empregado tenha 10000 tuplas (diferente da instância mostrada na Tabela 1) e que elas estão distribuídas de maneira uniforme nos valores do atributo idade cujo $\mathcal{D}_{\mathcal{A}} = \{1, 2, 3, ..., 99, 100\}$. Ou seja, a distribuição unidimensional desse atributo é $\mathcal{T}_{R.A} = \{(v, 100) \mid \forall v \in \mathcal{D}_{\mathcal{A}}\}$. Obviamente, em cenários reais muitas vezes não é possível conhecer essa distribuição, principalmente as distribuições multidimensionais.

Então, para calcular as cardinalidades de cada uma das operações presentes naquele plano, faz-se necessária a informação de seletividade do predicado de seleção $idade = 23$. Como é uma distribuição uniforme, então a seletividade para cada valor é:

$$s(A = v) = \frac{100}{10000} = \frac{1}{100} \tag{2.5}$$

Com isso, temos que a cardinalidade da operação de seleção pode ser calculada da seguinte forma:

$$c(\sigma) = |Empregado| \cdot s(idade = 23) = 10000 \cdot \frac{1}{100} = 100 \tag{2.6}$$

Nota-se que $|Empregado|$ representa o tamanho da relação Empregado. Portanto, esta consulta irá produzir 100 tuplas no resultado final.

Observe que durante todo o processo assumimos que tanto a distribuição dos dados quanto a informação de seletividade estavam disponíveis a priori. No entanto, em base de dados reais essa premissa não é satisfeita, por isso a necessidade de estimá-las. Caso essas estimativas

não sejam próximas das cardinalidades reais, pode acontecer que um PE ótimo em tempo de compilação, isto é, anterior à execução, se mostre subótimo no momento de sua execução devido aos erros introduzidos pelas estimativas de cardinalidade. Na próxima subseção, serão apresentadas algumas fontes que podem gerar essas imprecisões.

2.2.3.3 Fontes de Erros

Muitos trabalhos já mostraram que os estimadores de cardinalidade utilizados nos SGBDs modernos ainda possuem pontos fracos que fazem com que a acurácia dessas técnicas seja baixa em alguns cenários específicos, alguns deles citados no início desta dissertação. Na realidade, quando esses erros ocorrem, os SGBDs têm uma perda de desempenho devido ao plano de execução escolhido se mostrar subótimo no momento da sua execução real (LEIS *et al.*, 2017). As principais fontes de erros advém da maneira pelo qual os SGBDs tratam os dados armazenados em sua base, isto é, esses sistemas assumem premissas relacionadas às características que devem estar presentes nos dados armazenados. Entretanto, quando essas premissas não se refletem nas bases de dados armazenadas, ocorrem erros com grande ordens de magnitude (IOANNIDIS; CHRISTODOULAKIS, 1991). Na literatura, há quatro principais fontes de erros listadas, que iremos abordar a seguir.

Como dito anteriormente, não é possível conhecer a priori a distribuição dos dados. Para superar esse fato, os SGBDs armazenam algumas estatísticas relacionadas aos dados de modo a possibilitar algum tipo de mensuração. Por exemplo, a quantidade de valores únicos que aparecem em um atributo. Portanto, a primeira fonte de erro acontece quando essas estatísticas não estão atualizadas. Dessa forma, serão utilizadas estatísticas desatualizadas, o que pode impactar na qualidade das estimativas de cardinalidade.

A segunda fonte de erro está relacionada com o conceito de correlação de atributos que está definido logo a seguir e a premissa abordada é conhecida como IVA.

Definição 6. Dois atributos quaisquer A e B são ditos correlacionados quando os valores presentes no atributo A podem ser utilizados para predizer os valores que aparecem no atributo B. Caso contrário, os dois atributos são ditos independentes.

Um exemplo de atributos correlacionados é o par de atributos idade e salário de uma base dados que guarda as informações de pessoas. Em geral, quanto maior a idade, maior é o salário de uma pessoa. Por esse motivo, os valores do atributo idade podem ser

utilizados para predizer os valores dos salários das pessoas. Por outro lado, salário e CPF são atributos independentes, uma vez que os valores de salário não têm relação com os valores do CPF das pessoas. Veja que em uma mesma base de dados podem existir atributos tanto correlacionados quanto independentes. Como é difícil capturar as propriedades de correlação entre os atributos, muitas das técnicas usadas nos otimizadores de consultas dos SGBDs atuais partem do pressuposto que todos os atributos são independentes. Consequentemente, a ocorrência de atributos correlacionados leva a erros de grande magnitude nesses sistemas. É importante notar que muitas bases de dados reais possuem uma grande quantidade de atributos correlacionados, sendo o IMDB um exemplo claro desse tipo de dados (LEIS *et al.*, 2015). Dada uma consulta Q cujo predicado de seleção pode ser representado por $\mathcal{P} = p_1 \ \& \ p_2 \ \& \ ... \ \& \ p_l$. Assumindo a premissa de que todos os atributos são independentes, a seletividade de $\mathcal{P}$ pode ser calculada a partir da seguinte Equação 2.7. Ou seja, basta conhecer a seletividade de cada p_i que a seletividade de $\mathcal{P}$ será obtida através da multiplicação da seletividade de cada p_i.

$$s(\mathcal{P}) = s(p_1 \ \& \ p_2 \ \& \ ... \ \& \ p_l) = \prod_{i=1}^{l} s(p_i) \tag{2.7}$$

Uma outra fonte de erro advém da premissa assumida de que a distribuição dos dados $\mathcal{T}$, unidimensional ou multidimensional, é uniforme. isto é, para os casos simples que envolvem um único atributo, veremos que a técnica de histogramas soluciona essa fonte de erro. O problema ocorre quando se considera mais de um atributo de uma só vez, pois nesse caso a técnica de histogramas não é viável para a representação da distribuição multidimensional $\mathcal{T}$ dos atributos. Portanto, resta a opção de assumir que a combinação dos valores únicos presentes nos atributos segue uma distribuição uniforme, ou seja, cada grupo formado pela combinação dos valores dos atributos possui uma quantidade de tuplas iguais. Posto isso, quando os dados de atributos enviesados estiverem presentes, o SGBD irá produzir estimativas distantes da realidade dos dados. Por exemplo, considere um BD que mantém dados de filmes e seu respectivo gênero. É improvável que haja uma distribuição uniforme dos filmes entre os gêneros, isto é, há de haver um gênero de tal sorte que esse terá uma maior quantidade de filmes associados, como o gênero de terror ou ação deve ter uma maior quantidade de filmes que o gênero de fantasia.

Por fim, existe a premissa conhecida como *Hipótese da Contenção* (HC). Dados dois atributos A e B, essa premissa afirma que existe um valor correspondente no atributo B para todos os valores presentes no atributo A, isto é, $\mathcal{V}_A \subseteq \mathcal{V}_B$ e, dessa forma, $|A| \leq |B|$. Com esta premissa, o processo de estimar a cardinalidade de junções entre chaves primárias e estrangeiras torna-se

facilitado por meio da Equação 2.8 em que $s(R_i \bowtie R_j)$ é obtida por meio da Equação 2.9 cuja validade se apoia na existência do conceito de restrição de integridade referencial que garante que os valores da chave estrangeira (atributo A) possuem um valor correspondente na chave primária (atributo B). Entretanto, em cargas de trabalhos reais pode haver o que é denominado como junções muitos para muitos (junções m-n), isto é, junções que envolvam dois atributos que um mesmo valor de um atributo pode estar associado a mais de uma ocorrência no outro atributo e vice-versa. Com isso, as estimativas de cardinalidade desse tipo de junções serão prejudicadas quanto a qualidade.

$$c(R_i \bowtie R_j) = s(R_i \bowtie R_j) \cdot |R_i| \cdot |R_j| = min(|R_i|, |R_j|) \tag{2.8}$$

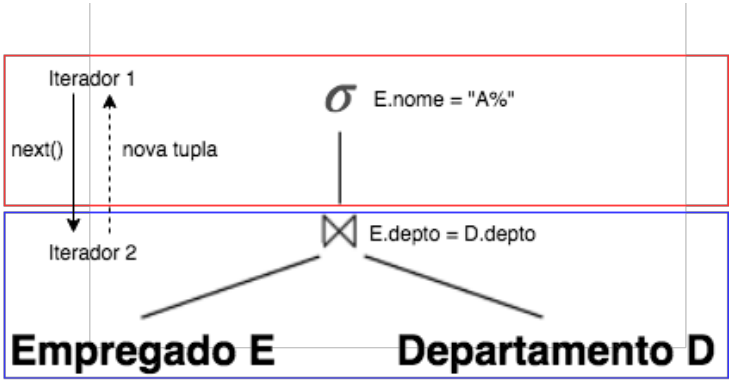
$$s(R_i \bowtie R_j) = \frac{|R_i \bowtie R_j|}{|R_i \times R_j|} = \frac{min(|R_i|, |R_j|)}{|R_i| \cdot |R_j|} = min(\frac{1}{|R_i|}, \frac{1}{|R_j|}) \tag{2.9}$$

2.2.4 Etapa III: Execução

Após todas as etapas descritas acima, tem-se como resultado um PE que já está preparado para ser executado e, por fim, produzir o resultado da consulta Q submetida. Nesta seção, vamos descrever brevemente o modelo Volcano que é amplamente adotado desde sua proposição e é o modelo que o PostgreSQL utiliza para realizar a execução das consultas. Esse modelo, também conhecido como *pipeline*, *pull* ou baseado em *stream*, tem sido utilizado para implementar os operadores da álgebra relacional presentes em um PE devido à sua simplicidade de implementação e aos seus benefícios de desempenho.

O seu surgimento ocorreu a partir da tentativa de evitar que os PEs gerados contivessem uma grande quantidade de operações de leitura e escrita desnecessárias. Como essas operações possuem um impacto enorme na eficiência dos SGBDs, minimizá-las se tornou mandatório para o alcance de um processamento de consulta eficiente. Consequentemente, as operações que são passíveis de exclusão devem ser evitadas no momento da geração do PE. Para tanto, os operadores relacionais são implementados nesse modelo como iteradores, os quais possuem três operações – *open*, *next* e *close* – cuja utilidade será a de materializar a interação entre eles. Um exemplo é apresentado na Figura 5. Como mostrado nesse exemplo, cada operador de um PE será implementado por meio de um iterador, sendo que a comunicação com um outro iterador (caso exista) dar-se-á através das suas operações. Resumidamente, a finalidade das operações *open*, *next* e *close* é, respectivamente, de preparar o iterador para o início da execução do plano, gerar novas tuplas à medida em que são requisitadas e, por fim, liberar os recursos que foram

Figura 5 – Visualização de um PE simples implementado no Volcano.



requisitados durante a execução do plano.

Em suma, os PEs são implementados por meio de vários iteradores que se comunicam através de chamadas àquelas três operações supracitadas. Esse processo ocorre até que todas as tuplas do resultado de uma consulta Q sejam geradas.

2.3 Aprendizagem de Máquina

A aprendizagem de máquina é uma subárea da Inteligência Artificial cujo objetivo é o desenvolvimento de técnicas computacionais que possibilitam construir sistemas capazes de extrair informações sobre os dados processados. Segundo (WEISS; KULIKOWSKI, 1991), um sistema de aprendizado é um programa computacional capaz de tomar decisões baseadas na experiência contida em exemplos solucionados com sucesso anteriormente, isto é, essa área visa propor técnicas que consigam criar modelos representativos dos padrões existentes em um dado conjunto de dados X. Dito de uma maneira mais formal, partindo da premissa que existe um padrão que pode ser modelado por uma função, linear ou não, f em X, as técnicas de aprendizagem de máquina são capazes de criar um modelo M baseado em X preparado para modelar uma função $\hat{f}$ que aproximará o padrão de f, isto é, $\hat{f}(X) \approx f(X)$. Esse modelo será utilizado para gerar saídas da função $\hat{f}$ fornecendo como entrada novos dados que não foram aplicados na construção de M. Ou seja, deseja-se que o modelo gerado tenha a habilidade de generalizar o comportamento de f capturado a partir de X para dados que não estão presentes nesse conjunto. De acordo com (ALPAYDIN, 2009), o poder de generalização de um modelo define se ele é representativo tanto para os dados já visualizados quanto para aqueles que ainda serão vistos. Posto que o objetivo é realizar predições para novas entradas, então se busca construir modelos com alto poder de generalização.

O conjunto de dados X, também conhecido como conjunto de dados de treinamento,

é composto por amostras (ou exemplos) x que são formadas por um conjunto de características (no inglês, *features*) $\{x_1, x_2, x_3, ..., x_n\}$, por exemplo idade, região, preço, etc. São essas as características que servirão de entrada para o algoritmo de aprendizagem a fim de construir um modelo M. Há dois tipos de características: numéricas e categóricas. As numéricas são representados por números inteiros ou reais, enquanto que as categóricas são representados por números booleanos ou valores textuais que representam categorias diferentes. Ademais, define-se como sendo a dimensão de X a quantidade n de características e como sendo o tamanho de X a quantidade de amostras x.

Geralmente, as técnicas de aprendizagem de máquina podem ser divididas em três abordagens de aprendizado distintas: supervisionado, não-supervisionado e por reforço. No aprendizado supervisionado, abordagem adotada neste trabalho, é necessário possuir um conjunto de treinamento X rotulado, isto é, para cada amostra $x \in X$, há um, e somente um, valor y denominado de rótulo ou valor de saída. Esse valor representa a resposta correta que um modelo M deveria apresentar como resultado ao dar como entrada a amostra x associada. É ele que vai guiar o processo de aprendizagem da solução e vai ser utilizado também para dizer se o modelo está representativo ou não. Ou seja, o rótulo dependerá do tipo de tarefa que deseja ser aprendido por M. Além disso, da mesma maneira que as características, o rótulo pode ser numérico ou categórico. Quando for numérico, o processo de gerar um modelo representativo M e, usando esse modelo, inferir o valor numérico do rótulo y, é chamado de regressão. Caso o rótulo y seja categórico, será denominado de classificação, pois nesse caso será inferido a categoria, classificando a amostra naquela categoria. Ilustrando um cenário de regressão, a Figura 6 apresenta as amostras (cada um dos pontos) de um conjunto de dados posicionadas de acordo com o valor do seu respectivo rótulo juntamente com um modelo de regressão (a reta) que foi obtido a partir dos dados.

Diferentemente do anterior, o aprendizado não-supervisionado não exige que os dados sejam rotulados. Essa abordagem é adequada ao cenário no qual não há uma massa de dados em que suas amostras estejam associadas a um rótulo. Nesses casos, busca-se construir um modelo que faça uma representação mais informativa do conjunto de dados disponível usando uma técnica como a de agrupamento (BISHOP, 2007). Um exemplo desse tipo de cenário é quando se deseja descobrir os diferentes perfis de usuários que assistem uma grande quantidade de filmes diferentes e, então, criar um sistema de recomendação para novos filmes. Como não é possível conhecer, a priori, os diferentes perfis existentes entre os usuários, fica inviável a

Figura 6 – Conjunto de dados rotulado associado a um modelo de regressão.

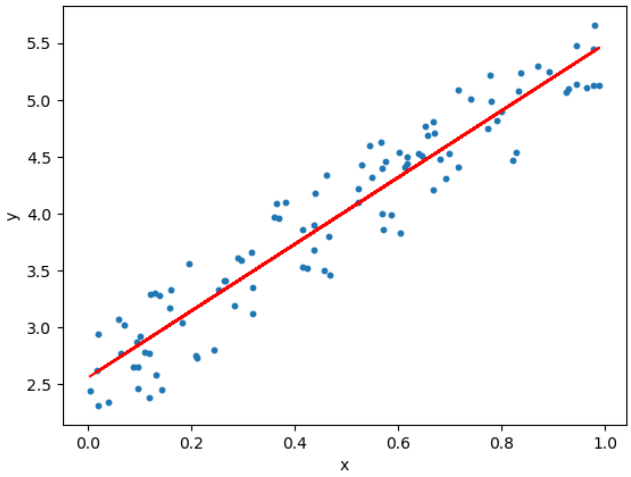
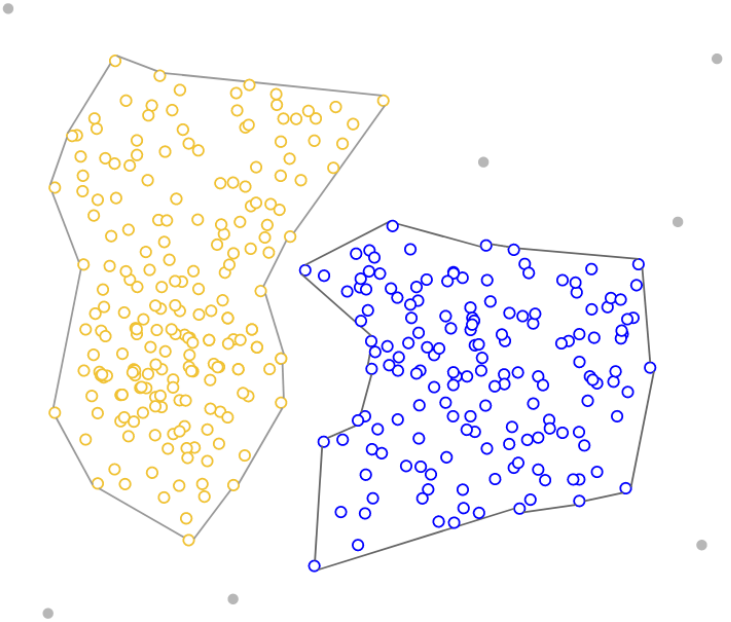


Figura 7 – Conjunto de dados dividido em dois grupos.

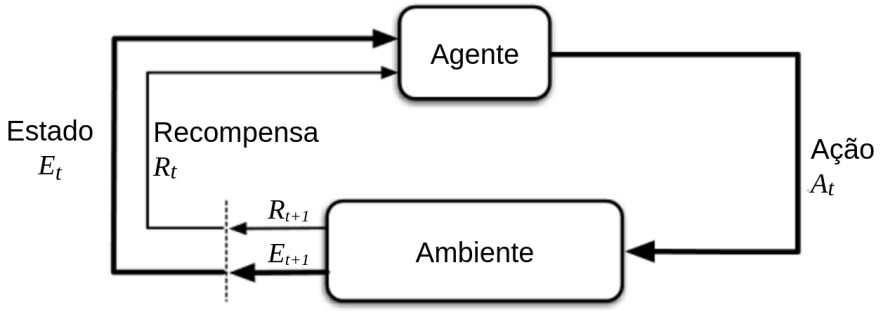


Fonte: (DEVELOPERS, 2019).

utilização do aprendizado supervisionado. De maneira pictográfica, pode-se observar na Figura 7 um conjunto de dados que foi separado em diferentes perfis ou grupos, como é definido na literatura.

O aprendizado por reforço se diferencia dos anteriores devido a não necessidade de se possuir um conjunto de dados X para treinar um modelo M. Para tanto, o processo de aprendizagem ocorre quando um agente realiza ações em um determinado ambiente, o qual possui um estado, e esse estado pode ser modificado de acordo com a ação tomada, como pode ser observado na Figura 8 (SUTTON *et al.*, 1998). A cada ação realizada é associada uma

Figura 8 – Fluxo da aprendizagem por reforço.



Fonte: (SUTTON *et al.*, 1998).

recompensa que será positiva, caso a ação ajude no processo de aprendizagem, ou negativa, quando a ação não auxilia na obtenção do objetivo proposto ao algoritmo de aprendizagem. Conclui-se, portanto, que o aprendizado do modelo M será baseado nas ações tomadas pelo agente.

Considerando agora o contexto desta dissertação, temos que a nossa pesquisa modela o Problema 1 como uma regressão, visto que as consultas SQL serão transformadas para uma representação compatível com o modelo de aprendizagem de máquina utilizado e servirão para formar o conjunto de dados de treinamento X e o rótulo y será a cardinalidade associada às consultas. Mais detalhes dos aspectos relativos à modelagem do Problema 1 em um problema de aprendizagem supervisionada serão apresentados no Capítulo 4. A função f que se espera que o modelo M gerado a partir da modelagem das consultas SQLs capture é tanto a distribuição unidimensional $\mathcal{T}_{R_i.A_j}$, para consultas que envolvem somente um único atributo A_j de uma relação R_i, bem como a distribuição multidimensional, quando mais de uma relação estiver envolvida nas consultas.

Existem várias técnicas de aprendizagem de máquina que são aplicáveis para o contexto de regressão. Durante o percurso dessa pesquisa, pelo menos duas técnicas foram testadas, a saber: *Máquinas de Aprendizado Extremo* (MAE) (HUANG *et al.*, 2012) e GBDT (BREIMAN *et al.*, 1984; FRIEDMAN, 2001), em especial o *LightGBM* (KE *et al.*, 2017). Não obstante, tendo em vista que esse último apresentou resultados melhores em relação ao primeiro, optou-se por utilizar o *LightGBM* no contexto desta dissertação.

Conforme veremos mais adiante, o modelo que é utilizado pelo *LightGBM* é o GBDT, que é capaz de modelar funções lineares assim como também as não lineares. Além do mais, o *LightGBM* se destaca devido às suas características particulares que fazem com que o seu processo de treinamento demande poucos recursos computacionais necessários para tal

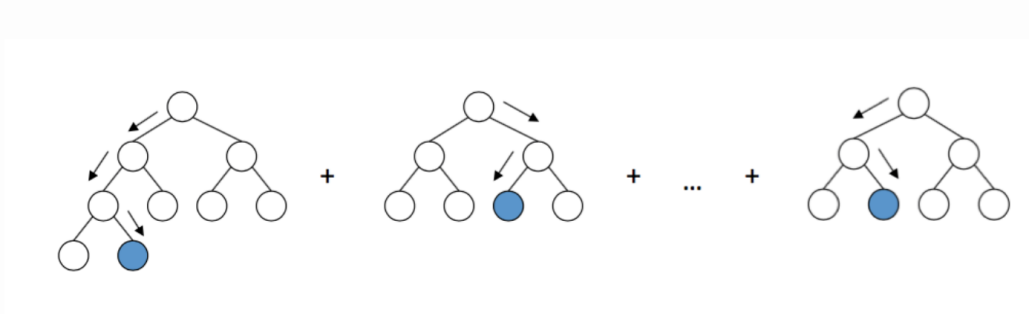
tarefa. Inclusive, esse modelo foi escolhido para o método proposto nesse trabalho devido a essa característica, principalmente. Como estamos em um cenário de otimização em tempo de compilação, isto é, antes de realmente executar a consulta submetida pelo usuário, então o *Tempo de Resposta* (TR) de uma consulta Q será o *Tempo de Otimização* (TO) juntamente com o *Tempo de Execução* (TE) real, ou seja: $TR = TO + TE$. Portanto, estamos em um cenário restrito de recursos, pois se escolhermos um modelo que represente bem as características dos dados subjacentes e, consequentemente, produza boas estimativas de cardinalidade, mas que tenha um alto custo associado tanto para construção quanto para predição, não será conveniente no contexto em que estamos inseridos, visto que nesse caso fictício teremos que o TO impactará de maneira negativa diretamente no TR, o que não é desejável. Nota-se, também, que a quantidade de memória utilizado pelo modelo não pode ser muito grande, uma vez que nesse caso o modelo irá concorrer com o armazenamento dos dados reais, gerando um novo gargalo nos SGBDs. Com isso, optamos pelo *LightGBM* para construir a solução do problema que atacamos nesta dissertação.

2.3.1 LightGBM

O *LightGBM* é uma nova implementação do algoritmo de aprendizagem conhecido como *Gradient Boosting Decision Tree* (GBDT) com o objetivo de reduzir a complexidade computacional, pois esse algoritmo é conhecido por ter dificuldades de eficiência e escalabilidade quando lida com um conjunto massivo de dados. Veremos a seguir que as técnicas *Gradient-based One-Side Sampling* (GOSS) e *Exclusive Feature Bundling* (EFB) incorporadas ao GBDT tornam possível a construção de um modelo M representativo treinado em um tempo de treinamento menor que as outras implementações (KE *et al.*, 2017). Antes, uma explicação do algoritmo base GBDT.

O GBDT é um algoritmo de aprendizagem que segue o paradigma de aprendizagem conhecido como *ensemble*. Nessa abordagem, um conjunto de estimadores base, ilustrado na Figura 9, são treinados para resolver a mesma tarefa com o intuito de que cada um desses estimadores gere uma solução e, posteriormente, utiliza-se de alguma estratégia (média, votação majoritária, etc) para combinar essas soluções em uma única solução final para a tarefa definida (SOLLICH; KROGH, 1996; MARQUÉS *et al.*, 2012). A motivação por trás dessa ideia é que a combinação das predições de vários estimadores base fracos (i.e., um estimador é dito ser fraco quando suas predições são um pouco melhores que as de um estimador aleatório) gera um modelo

Figura 9 – Ilustração de um conjunto de estimadores base usado no GBDT.



Fonte: (ROGOZHNIKOV, 2016).

robusto, poderoso, mais representativo (JANSEN, 2018). Por conta disso, os modelos *ensemble* ganharam atenção na comunidade científica ao apresentarem um bom poder de predição em diferentes tipos de aprendizagem, como a regressão, que é o tipo de tarefa em que o método proposto está modelado (HASHEM; SCHMEISER, 1995).

Para treinar um modelo usando o GBDT, utiliza-se o Algoritmo 1. Para tanto, são fornecidos como entrada o conjunto de treinamento X com seus respectivos rótulos y juntamente com o número de estimadores base num_est, taxa de aprendizagem λ (i.e., quanto um estimador base vai aprender em cada etapa) e uma função de custo diferenciável L. Essa função L será utilizada no processo de otimização que é responsável por ajustar os estimadores base de modo que estes sejam capazes de aprender a tarefa que lhes foi imposta. Isto é, essa etapa de otimização visa diminuir o valor da função custo L, tendo em vista que o custo é definido como sendo o erro de aprendizagem, ou seja, vislumbra-se que os estimadores base sejam otimizados a fim de que o seu erro de predição seja diminuído. É válido lembrar que o objetivo dos estimadores base é ser capaz de predizer melhor que um estimador aleatório. Na prática, duas funções de custo são muito utilizadas, sendo uma para regressão e outra para classificação, a saber: *Erro Quadrático Médio* (EQM) e Perda Logarítmica. No próximo capítulo, veremos que é possível utilizar o EQM no método proposto, além de uma outra função que será descrita nele. Em relação ao número de estimadores e à taxa de aprendizagem, essas são duas entradas que precisam ser ajustadas por meio de um *Grid Search*, por exemplo, e, por conta disso, são denominadas de hiper-parâmetros no contexto de aprendizagem de máquina.

Com as entradas fornecidas, o algoritmo já dá início ao processo de aprendizagem na linha 3. O processo de *boosting* utiliza as informações do estimador base anterior para ajustar um novo modelo. Todavia, como não existe tal estimador no início do processo (M_{lista} é declarado vazio na linha 2), então é necessário que se faça uma suposição inicial a fim de que a equação

abaixo seja minimizada. No caso de se utilizar o EQM, essa suposição será o valor médio dos rótulos, por exemplo. Nesta equação, o γ representa o fator de contribuição que um estimador base terá para o processo de aprendizagem. Vale relembrar que os novos estimadores base levam em consideração o processo de aprendizagem dos anteriores, ou seja, a contribuição de um novo estimador para a aprendizagem é dependente do modo que foi realizado a aprendizagem até aquele ponto, sendo esta a razão pela qual um fator de contribuição γ se faz necessário.

$$\arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma) \tag{2.10}$$

Com um modelo $M_{anterior}$ já calculado, adiciona-se este na lista mantida na variável M_{lista} (linha 4) e daí em diante as linhas 5-10 serão responsáveis por executar o processo de aprendizagem dos outros estimadores base de modo a gerar o modelo final treinado M. Inicialmente, na linha 6, é avaliado o erro do $M_{anterior}$ a fim de que ele seja útil no processo de otimização do novo estimador base E. Nessa fase, é realizado o cálculo do gradiente da função de custo através do uso de derivada parcial, como mostrado na Equação 2.11, e com essa informação em mãos tem-se o conhecimento da direção em que a função de custo possui tendência de crescimento. Tendo em vista o objetivo de minimização, então é adicionado o sinal de negativo para pegar a direção inversa.

$$r = -\frac{\partial L(y_i, M_{anterior}(x_i))}{\partial M_{anterior}(x_i)} \tag{2.11}$$

De posse dessas informações, efetua-se o processo de otimização fazendo uso da Equação 2.12 cujo objetivo é encontrar um novo fator de contribuição γ e um novo estimador base E que minimizem a função de custo L, ou seja, que incrementem o processo de aprendizagem, vislumbrando produzir um modelo mais representativo. Diferentemente do processo inicial que utiliza a Equação 2.10 e que considera somente os rótulos, nessa já se possui um $M_{anterior}$ treinado, e portanto, também o considera no processo de otimização.

$$\arg\min_{\gamma, E} \sum_{i=1}^{n} L(y_i, M_{anterior}(x_i) + \gamma E(x_i)) \tag{2.12}$$

Em seguida, na linha 8, é calculado o novo modelo $M_{anterior}$ que é formado pelo $M_{anterior}$ atual junto com o novo estimador base E, sendo que este é parametrizado tanto pelo fator de contribuição γ bem como pela taxa de aprendizagem λ fornecido na entrada. Para finalizar o laço, o modelo $M_{anterior}$ é adicionado na lista M_{lista} de modo a preparar o algoritmo para a próxima iteração do laço definido na linha 5 e iniciar todas essas etapas novamente. Ao

final das iterações desse laço, o modelo final M é obtido por meio da seguinte equação, a qual representa o somatório de todos os modelos gerados. Por fim, retorna-se o modelo M treinado.

$$M = \sum_{k=1}^{num\_est} M_{lista}[k] \tag{2.13}$$

Resumidamente, todo o processo realizado durante o processo de treinamento pode ser representado na Equação 2.14. Na etapa k, para calcular o modelo M_k, são utilizados o modelo atual M_{k-1} em companhia do novo estimador base E_k, o qual é parametrizado pela taxa de aprendizagem λ e pelo fator de contribuição γ_k.

$$M_k(X) = \underbrace{M_{k-1}(X)}_{\textbf{modelo atual}} + \underbrace{\lambda\gamma_k E_k(X)}_{\textbf{novo estimador}} = M_{k-1}(X) + \arg\min_{\gamma_k, E_k} \sum_{i=1}^{n} \underbrace{L(y_i, M_{k-1}(x_i) + \gamma_k E_k(x_i))}_{\textbf{Função de custo}} \tag{2.14}$$

Após a explanação do GBDT, adentraremos mais nas características exclusivas do *LightGBM*.

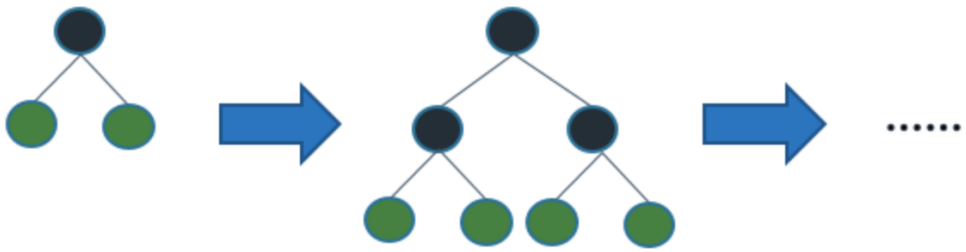
Algoritmo 1: GBDT

Entrada: X, y, num_est, λ, L
Saída: M

início
  $M_{lista} \leftarrow \emptyset$
  $M_{anterior} \leftarrow$ *Estimador base inicial ajustado para X usando a Equação 2.10*;
  $M_{lista}.adiciona(M_{anterior})$
  para $k = 1 \rightarrow k = num\_est - 1$ **faça**
    $r \leftarrow$ *Calcular o erro do* $M_{anterior}$ *usando X por meio da Equação 2.11*;
    $\gamma, E \leftarrow$ *Fator de contribuição e estimador base k ajustados para X pela Equação 2.12 usando r calculado previamente*;
    $M_{anterior} \leftarrow M_{anterior} + \lambda\gamma E$
    $M_{lista}.adiciona(M_{anterior})$
  fim
  $M \leftarrow \sum_{k=1}^{num\_est} M_{lista}[k]$
  retorna M
fim

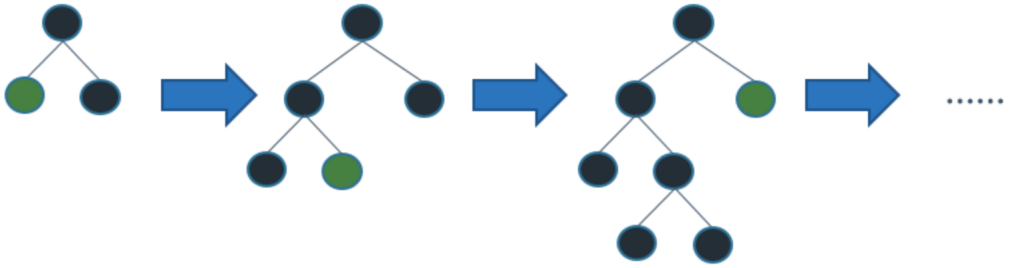
A primeira delas é a forma de construção dos estimadores base (que são árvores de decisão) que é adotada nessa técnica. Enquanto que na maioria dos algoritmos é utilizada a estratégia conhecida como *Level-wise*, no *LightGBM* é usada a abordagem *Leaf-wise* (KE *et al.*, 2017). Na terminologia dos grafos, pode se dizer que cada uma das estratégias se assemelha com as buscas por largura e por profundidade, respectivamente. Na primeira, a *Level-wise*, novos nós serão adicionados nas árvores de decisão de acordo com o nível da estrutura, ou seja, as árvores vão crescer horizontalmente, como é possível observar na Figura 10.

Figura 10 – Estratégia de crescimento *Level-wise*.



Fonte: (MICROSOFT, 2017).

Figura 11 – Estratégia de crescimento *Leaf-wise*.



Fonte: (MICROSOFT, 2017).

Em contrapartida, na *Leaf-wise*, as árvores de decisão vão possuir um desenvolvimento vertical no momento da adição de novos nós, como ilustrado na Figura 11. Essa diferença de abordagem resulta em uma melhor tendência de se alcançar um menor erro de aprendizagem no *LightGBM* (KE *et al.*, 2017). Contudo, pode ocorrer, com uma maior probabilidade, que o modelo gerado seja enviesado, isto é, que ele se ajuste tanto aos dados do conjunto de treinamento que o seu poder de generalização será baixo na hora de fazer predições para aqueles dados que ainda não foram visualizados por ele. Esse problema vai se acentuar, principalmente, quando o conjunto de treinamento for pequeno (KE *et al.*, 2017). Por um lado, é bom que o modelo apresente um menor erro de aprendizagem. Por outro lado, pode acontecer que o seu modelo fique enviesado. Consequentemente, há de se ter mais precaução na hora de treinar modelos usando o *LightGBM*.

O *LightGBM* propôs dois novos algoritmos, GOSS e EFB, que possibilitaram o uso de uma grande quantidade de dados para o treinamento do modelo de maneira eficiente. Em síntese, o GOSS realiza o processo de divisão dos nós das árvores de decisão para que o modelo

treinado faça boas predições, enquanto que o EFB traz uma nova forma de realizar essa tarefa de uma maneira computacionalmente eficiente.

Para finalizar, é necessário notar uma outra característica que as técnicas de GBDT possuem e, por conseguinte, conseguem ser capazes de realizar outros tipos de predições. Dependendo da função de custo que se escolhe para fazer a construção do modelo, é possível definir qual estatística deverá ser predita pelo modelo a ser criado. Geralmente, no processo de aprendizagem, assume-se que o valor médio ou a esperança matemática da função f presente no conjunto de treinamento é a estatística que será aproximada pela função $\hat{f}$ por meio do modelo treinado M. No entanto, há várias estatísticas distintas que podem ser utilizadas, por exemplo os percentis ou quantis da distribuição do conjunto de dados de treinamento. A propósito, veremos que o método apresentado no Capítulo 4 fará uso dessa importante capacidade do *LightGBM* com o intuito de gerar um intervalo de predição, ao invés de gerar somente uma predição em um único ponto, e, a partir desse intervalo, avaliar o nível de incerteza presente na predição de um único ponto (o valor médio). Dessa forma, é possível afirmar se o modelo M é confiável para realizar as predições. A maneira pela qual o *LightGBM* faz a escolha da sua função de custo de modo a possibilitar a predição de quantis é baseada na técnica de Regressão Quantílica (KOENKER; HALLOCK, 2001).

2.4 Conclusão

Este capítulo apresentou o arcabouço necessário para o entendimento das etapas realizadas no âmbito do processamento de consultas nos SGBDs, sendo que algumas definições e notações necessárias para o restante do texto também foram relacionadas. Além disso, as principais fontes de erros presentes nas atuais técnicas de estimar as cardinalidades foram listadas e descritas. Posteriormente, introduzimos uma breve apresentação dos principais aspectos relativos à área de aprendizagem de máquina, tais como conjunto de treinamento e tipos de treinamento. Finalmente, a técnica de aprendizagem de máquina GBDT foi descrita com o intuito de explicar o funcionamento do *LightGBM* que será adotado no método proposto por meio desta dissertação.

3 TRABALHOS RELACIONADOS

O problema abordado neste texto permeia a comunidade científica desde os primórdios da tecnologia relacional. Não obstante, como foi explanado no Capítulo 2, ainda há desafios que precisam ser superados nas técnicas desenvolvidas até aqui para esse problema. Inúmeros trabalhos propostos na literatura abordam o problema da estimação de cardinalidades, do mesmo modo, seletividades. Neste capítulo, vamos descrever as duas técnicas convencionais utilizadas ainda hoje nos SGBDs, histogramas e amostragem. Ademais, também descreveremos os novos trabalhos mais relevantes que propuseram o estudo da incorporação de mecanismos que possibilitam a geração de estimativas levando em consideração a experiência do passado, enfatizando aqueles que utilizam técnicas do ramo da aprendizagem de máquina.

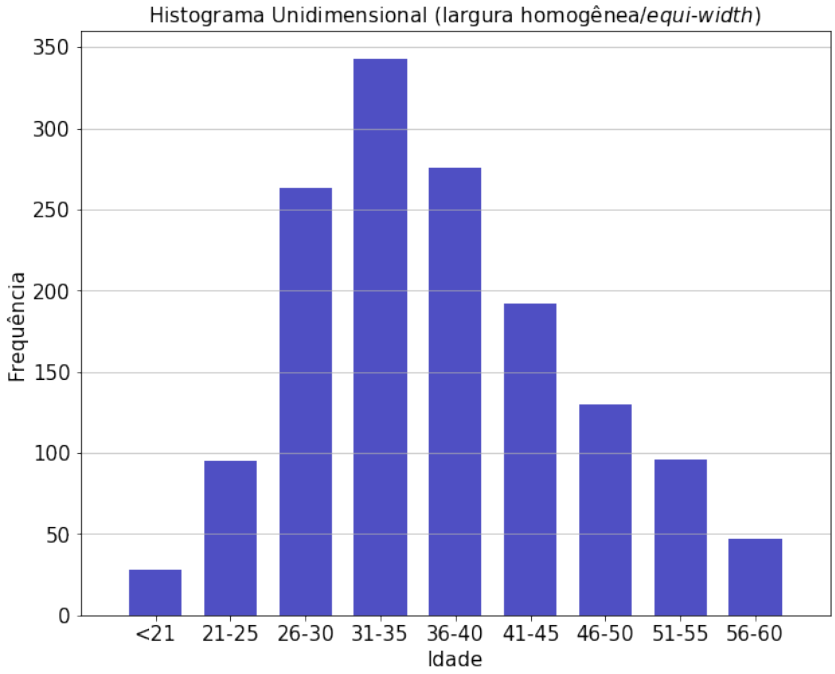
Iniciamos esse capítulo abordando as duas técnicas tradicionais de estimação de cardinalidades, histogramas e amostragem, nas Seções 3.1 e 3.2, respectivamente. Adiante, nas Seções 3.3, 3.4 e 3.6, detalharemos três importantes trabalhos que estão indiretamente relacionados ao contexto abordado neste texto, haja vista que esses não tratam da incorporação das técnicas aprendizagem de máquina, todavia apresentam uma nova abordagem para incorporar a experiência no processo de estimação. Aqueles trabalhos presentes na literatura que estão diretamente relacionados serão apresentados nas Seções 3.5, 3.7, 3.8 e 3.9. No final, é levantada uma discussão sobre os pontos fortes e as limitações de cada um dos trabalhos relacionados e, por conseguinte, é realizada uma análise comparativa entre esses trabalhos, vislumbrando situar o trabalho proposto nesta dissertação.

3.1 *The Optimization of Queries in Relational Databases*

O primeiro trabalho relacionado descreveu a técnica de histogramas (KOOI, 1980) que é muito utilizada nos SGBDs devido à sua simplicidade e desempenho. Basicamente, um histograma tenta aproximar a distribuição dos dados unidimensional $\mathcal{T}$ por particionamentos de intervalos disjuntos dos dados, tendo duas maneiras de realizar essa divisão: *equi-width* e *equi-height*. Na primeira abordagem, os dados são divididos em faixas que possuem a mesma quantidade de valores, i.e., mesma largura, e para cada faixa é associada a respectiva frequência de valores que ocorrem naquela faixa, veja um exemplo de histograma desse tipo, que mapeia a distribuição de dados do atributo Idade de um BD, na Figura 12. Esta imagem apresenta a frequência dos respectivos intervalos de Idade. Por exemplo, é possível concluir que existem

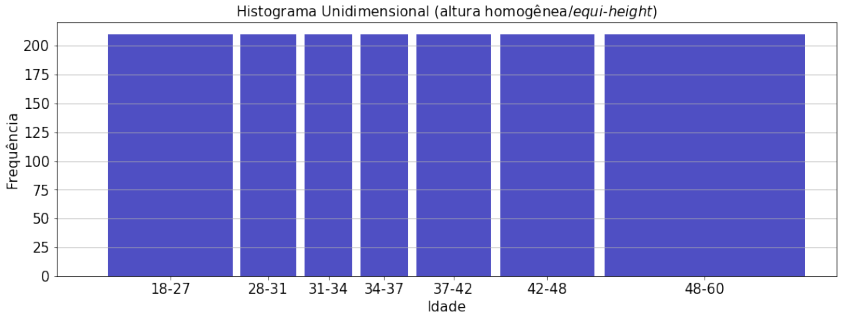
mais de 300 tuplas cujo valor no atributo Idade está entre 31 e 35. Ademais, observe que há cinco valores em cada grupo de Idade.

Figura 12 – Exemplo de histograma unidimensional *equi-width* do atributo Idade.



Sob outra perspectiva, surgem os histogramas *equi-height* que agrupam os dados em faixas de tal sorte que todas as faixas possuam a mesma frequência, i.e., mesma altura. Ou seja, a quantidade de valores dentro de uma faixa será variável, pois dependerá de se alcançar a frequência desejada para cada uma das faixas. Um exemplo desse tipo de histograma, também para o atributo Idade, é apresentado na Figura 13. A primeira observação pertinente é que todas as classes de frequência de Idade possuem a mesma frequência, neste caso, 200. Todavia, veja que a divisão das classes não segue uma quantidade fixa de valores, por exemplo a primeira classe tem 10 valores (18-27) ao mesmo tempo que a última tem 13 valores (48-60).

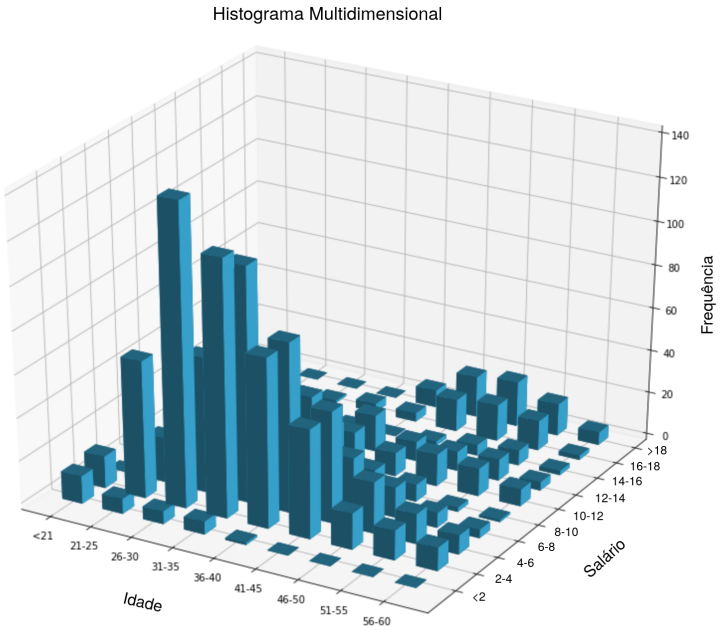
Figura 13 – Exemplo de histograma unidimensional *equi-height* do atributo Idade.



Os histogramas também podem ser construídos para aproximar a distribuição dos

dados multidimensionais $\mathcal{T}$, no caso temos os conhecidos histogramas multidimensionais. Por exemplo, na Figura 14, é ilustrado um histograma *equi-width* desse tipo para os atributos Idade e Salário. Observe que a diferença entre os histogramas uni e multidimensionais é a capacidade de aproximar a distribuição de dados de somente um ou mais atributos. O histograma apresentado na imagem é composto por três dimensões: duas referentes aos valores dos atributos mapeados e uma que apresenta a frequência desses valores. Veja que utilizando essa estrutura é possível perceber que há uma correlação entre esses dois atributos. Por exemplo, é possível concluir que quanto maior a idade, maior é o valor do salário. Um outro exemplo de informação importante que se pode retirar é que existem mais pessoas com idades entre 21 e 45 e que apresentam salário até o valor máximo de 14. Caso esse histograma não estivesse disponível no SGBD, não seria possível chegar a essa informação através da hipótese *Independência de Valores de Atributos* (IVA).

Figura 14 – Exemplo de histograma multidimensional *equi-width* dos atributos Idade e Salário.



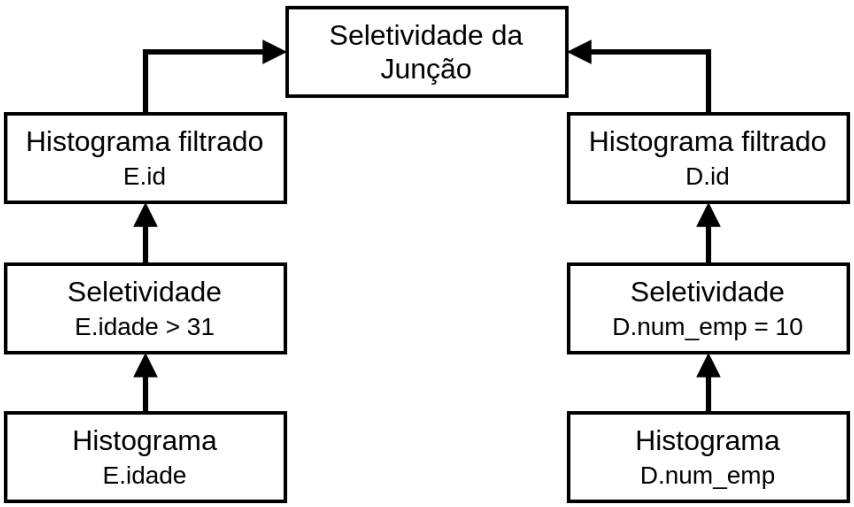
Para calcular as estimativas de cardinalidade de uma consulta Q que envolve tanto operações de seleção e junção, necessita-se que existam histogramas para os atributos que estão envolvidos na consulta. Satisfeito esse pré-requisito, o histograma correspondente a cada atributo é primeiramente consultado de forma a calcular a seletividade dos predicados de seleção que envolvem aquele atributo. Depois, os histogramas dos atributos envolvidos são filtrados pela seletividade dos predicados e estes são usados para que a seletividade da junção seja obtida. Isto ocorre porque é possível assumir a premissa IVA. Por exemplo, considere a seguinte consulta

SQL:

```
SELECT * FROM E, D WHERE E.id = D.id AND E.idade > 31 AND D.num_emp = 10;
```

A Figura 15 mostra todo o processo para gerar as estimativas de cardinalidade da consulta especificada acima. Observe na imagem que os histogramas base são consultados para então chegar ao histograma modificado pela seletividade dos predicados $E.idade > 31$ e $D.num\_emp = 10$. Posteriormente, eles são utilizados para calcular a seletividade da junção cuja condição é $E.id = D.id$.

Figura 15 – Exemplo de uso da técnica de histogramas para estimar as cardinalidades de uma consulta.



3.2 *Practical Selectivity Estimation Through Adaptive Sampling*

A outra técnica que recebeu grande atenção por parte da literatura foi a amostragem (LIPTON *et al.*, 1990), que surgiu para ser uma evolução da técnica de histogramas. Essa técnica realiza a estimação de cardinalidade por meio da coleta de uma pequena amostra randômica retirada dos dados e, então, escala as características presentes nessa amostra para toda a base de dados. Esse processo é realizado até que se alcance uma acurácia aceitável. Por esse motivo, essa abordagem apresenta, essencialmente, uma melhor acurácia quando comparada a de histogramas. Para explicar o processo de geração de estimativas de cardinalidade dessa técnica, considere o exemplo de consulta SQL a seguir:

```
SELECT * FROM R, S, T WHERE R.id = S.id AND S.id = T.id;
```

Inicialmente, por meio de uma técnica de amostragem estatística, selecionam-se amostras randômicas e independentes das relações R e S, respectivamente. Inferem-se as características de cada uma dessas amostras e, por fim, é realizado o cálculo da estimativa de cardinalidade da operação de junção $R \bowtie S$. Usando um valor superior limite, é calculado a possível acurácia dessa estimativa. Caso não satisfaça um parâmetro definido pelo DBA, o processo de estimação é iniciado novamente. Isto se repete até o momento em que esse parâmetro é satisfeito e, então, é iniciado o processo de estimação da junção $S \bowtie T$ já considerando o resultado obtido. De maneira análoga, é realizado a amostragem da relação T com a finalidade de conhecer as características de T e, então, calcular a estimativa de cardinalidade. Depois de ter satisfeito o parâmetro da acurácia, as estimativas calculadas são injetadas no otimizador de consultas para que este gere um novo PE considerando as estimativas já obtidas.

3.3 *LEO - DB2's LEarning Optimizer*

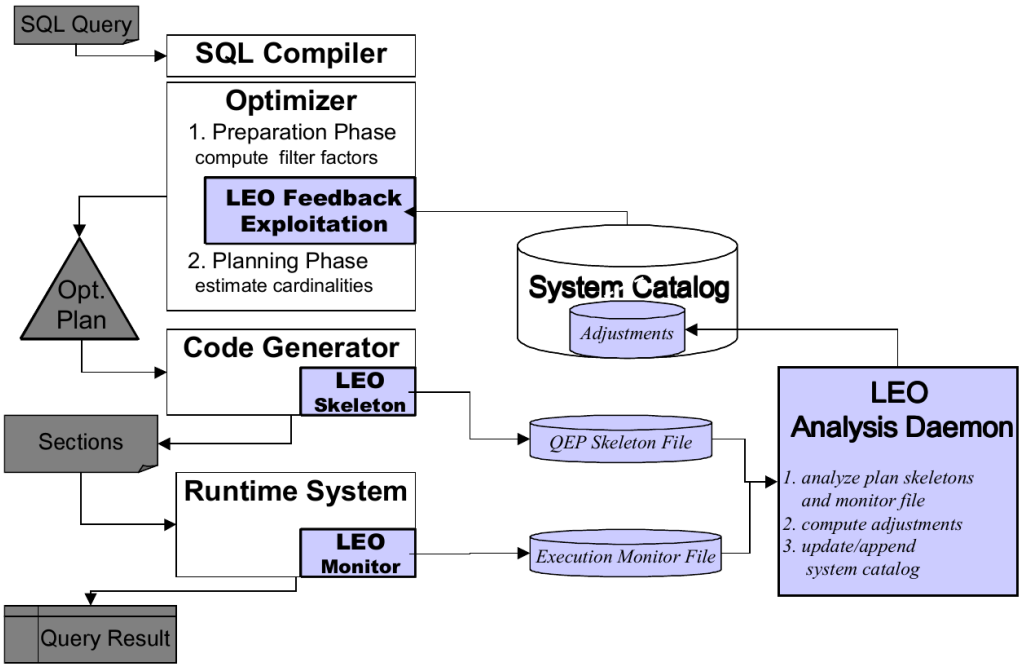
Esse trabalho relacionado apresentou o *DB2's LEarning Optimizer* (LEO) (STILLGER *et al.*, 2001), que foi um dos primeiros a ser desenvolvido tendo em vista construir um mecanismo para gerar as estimativas de cardinalidade considerando também os resultados das consultas anteriores. Isto é, um método que realiza o monitoramento das cardinalidades presentes nos PEs no tempo de execução com o propósito de obter as cardinalidades reais e, consequentemente, injetá-las nas próximas consultas que envolvam as cardinalidades já adquiridas anteriormente. Explorar esse valioso conhecimento obtido através das consultas já executadas traz à tona os possíveis erros que ocorreram no processo de otimização dessas consultas, possibilitando a identificação da parte na qual acontece uma imprecisão nas estimativas. Dessa forma, torna-se viável a construção de uma maneira de abordar e corrigir esses erros nas consultas futuras. Assim, o impacto dos erros de estimação ocorridos no processo de otimização será menor, tornando o desempenho dos SGBDs mais eficiente.

Para viabilizar a incorporação do conhecimento adquirido das consultas anteriores no processamento das consultas atuais, os autores do LEO propuseram a adição de quatro componentes na arquitetura dos SGBDs, os quais estão ilustrados na Figura 16 tendo como cor azul. Os outros componentes que aparecem nessa imagem são aqueles já tradicionais que fazem parte da maioria dos SGBDs, como apresentado na Seção 2.2.1.

De maneira resumida, os quatro componentes possuem as seguintes funcionalidades:

1. *LEO Skeleton*. Encontra-se no módulo de geração de código de execução dos PEs e é

Figura 16 – Arquitetura do LEO proposta no trabalho (STILLGER *et al.*, 2001).



Fonte: (STILLGER *et al.*, 2001).

responsável por capturar e armazenar esses planos de execução em um formato próprio (denominado de *skeleton*) em um arquivo especial. Como os SGBDs já possuem um módulo de geração de código, basta uma modificação para que esse componente seja implementado;

2. *LEO Monitor*. Interage com o módulo de execução dos SGBDs com a finalidade de monitorar as cardinalidades reais geradas pelos PEs executados. Do mesmo modo que o componente anterior, necessita-se de algumas modificações no módulo de execução para que esse monitoramento seja realizado;
3. *LEO Analysis Daemon*. Esse componente é autônomo no sentido de que pode ser efetuado separadamente do processo servidor dos SGBDs e é capaz de analisar tanto os *skeletons* quanto as cardinalidades reais associadas aos *skeletons* para que os ajustes necessários sejam armazenados no catálogo dos SGBDs e, por fim, sejam disponibilizados para o próximo componente. No caso do atual componente em específico, é necessária a adição de um novo módulo realmente na arquitetura dos SGBDs;
4. *LEO Feedback Exploitation*. É integrado ao módulo de otimização de consultas com a intenção de injetar as cardinalidades adquiridas anteriormente no processo de otimização das novas consultas. Mais uma vez, esse componente pode ser implementado ao se fazer algumas alterações no módulo de otimização.

Os componentes supracitados são independentes entre si, conquanto a orquestração entre eles ocorre de maneira sequencial visando formar um mecanismo de aprendizagem. Primeiramente, os PEs são capturados e armazenados em um arquivo *QEP Skeleton File* pelo *LEO Skeleton*. À medida que os PEs são executados, as suas cardinalidades reais são monitoradas pelo *LEO Monitor*, isto é, formam-se pares $(exp, card)$ que vão associar uma expressão relacional exp com a sua respectiva cardinalidade $card$, os quais serão armazenados no arquivo *Execution Monitor File*. Subsequentemente, ambos os arquivos que armazenam os *skeletons* e as cardinalidades serão analisados no *LEO Analysis Daemon* a fim de que sejam detectados os ajustes necessários para as próximas consultas. Estas, por sua vez, farão uso desse conhecimento por meio do *LEO Feedback Exploitation*, o qual utilizará o conhecimento adquirido para injetar as cardinalidades nos próximos planos a serem executados.

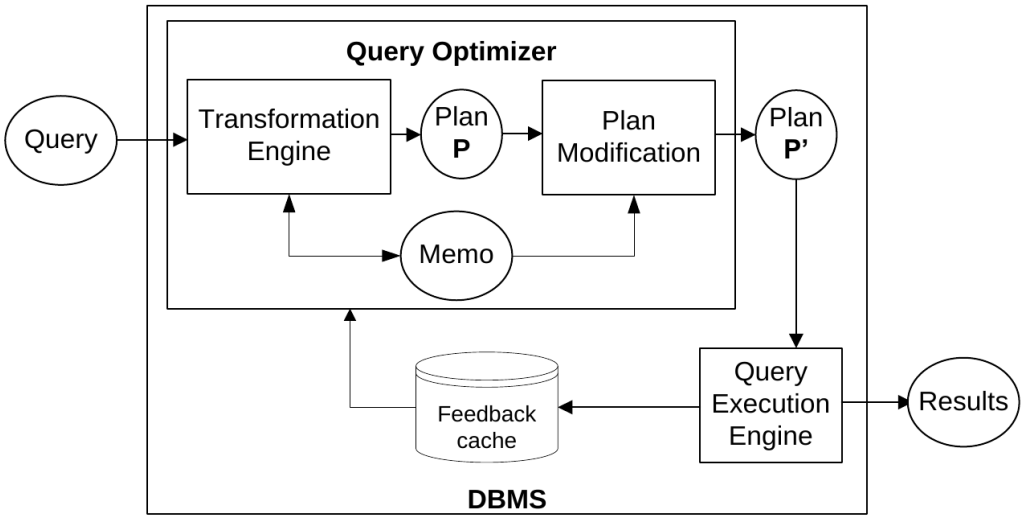
3.4 *A Pay-as-you-go Framework for Query Execution Feedback*

O *Pay-as-you-go* (CHAUDHURI *et al.*, 2008) é uma proposta de evolução para o mecanismo apresentado no trabalho relacionado descrito na seção anterior. Muito embora o LEO tenha trazido uma melhora na acurácia das estimativas de cardinalidade, os autores do atual trabalho relacionado identificaram que era possível melhorar as estimativas com um pequeno acréscimo no custo de executar as consultas. Ou seja, o *Pay-as-you-go* também reside na ideia de utilizar o conhecimento das cardinalidades adquiridas nas consultas já executadas no processamento das consultas futuras. Porém, novos mecanismos de monitoramento, denominados de monitoramento proativo, foram propostos com vistas a possibilitar que o otimizador de consultas influencie no processo de decidir quais cardinalidades devem ser obtidas a partir da execução de um PE. Esses mecanismos serão responsáveis por transformar um PE escolhido para executar uma dada consulta Q em um novo $\overline{PE}$ de modo que esse novo plano seja executado para gerar o resultado de Q ao mesmo tempo em que as cardinalidades que parecem ser mais proeminentes sejam coletadas.

A arquitetura da abordagem *Pay-as-you-go* é apresentada na Figura 17 e tem a intenção de ilustrar as modificações necessárias na arquitetura tradicional dos SGBDs para que essa técnica seja implementada. Basicamente, dois novos componentes são essenciais:

1. *Feedback cache*. Esse componente é utilizado para armazenar as cardinalidades reais vistas durante a execução dos PEs de consultas já efetuadas. Ou seja, é nesse componente que são guardados os pares $(exp, card)$. Um paralelo pode ser feito com o componente

Figura 17 – Arquitetura do *Pay-as-you-go* proposta no trabalho (STILLGER *et al.*, 2001).



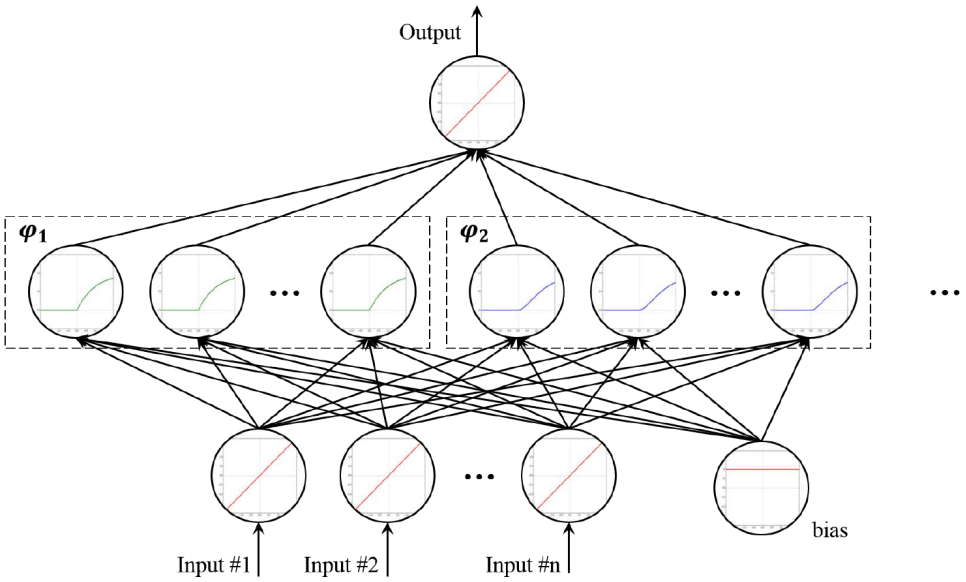
Fonte: (STILLGER *et al.*, 2001).

LEO Monitor do trabalho descrito anteriormente. Para que esse componente seja integrado na arquitetura, basta uma modificação no módulo de execução de consultas;

2. *Plan Modification*. Com a finalidade de aplicar as técnicas do mecanismo proativo, esse componente transformará um PE em um novo plano que tornará possível a obtenção de cardinalidades adicionais, ou seja, aquelas que não seriam obtidas durante a execução do PE agora é viável obtê-las por meio da execução do novo plano. Esse componente requer que um novo módulo que interage com o módulo de processamento de consultas dos SGBDs seja criado.

No que diz respeito aos mecanismos do monitoramento proativo, os autores propuseram um conjunto para ser aplicado em predicados de seleção, assim como um conjunto para a operação de junção. Um exemplo do primeiro conjunto é o *Prefix Counting*. Considere um predicado de seleção $\mathcal{P} = A < 10 \ \& \ B = 20 \ \& \ C < 30 \ \& \ D = 40$ de uma consulta $\mathcal{Q}$ qualquer. Usando essa técnica, é possível adicionar um contador para cada prefixo de $\mathcal{P}$ do qual se deseja obter a cardinalidade. Isto é, suponha que precisamos conhecer a cardinalidade do prefixo $A < 10$. Basta então associarmos um contador para essa expressão e a cada vez que uma tupla satisfizer o predicado presente no prefixo, o contador correspondente será incrementado. Em suma, com o pequeno *overhead* do contador adicional, possibilitou-se que a cardinalidade de uma expressão, que não aparecia diretamente na consulta submetida, fosse obtida.

Figura 18 – Arquitetura de uma rede neural aumentada com múltiplas entradas proposta em (LIU *et al.*, 2015). Essa arquitetura é utilizada para gerar o estimador principal desse trabalho relacionado.



Fonte: (LIU *et al.*, 2015).

3.5 *Cardinality Estimation Using Neural Networks*

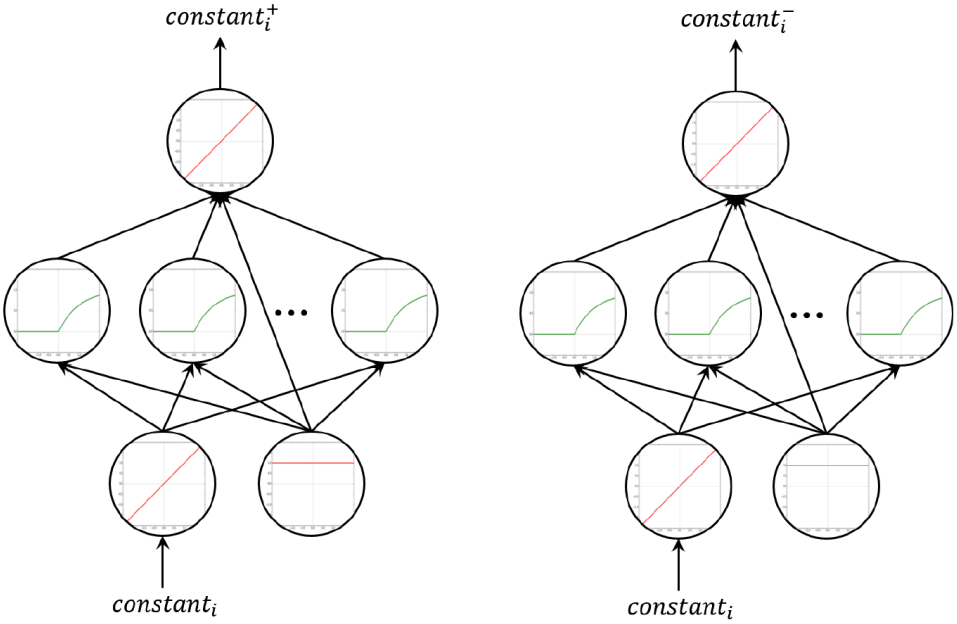
Até agora, os trabalhos relacionados propuseram abordagens para gerar as estimativas de cardinalidade que não envolvem o uso de técnicas da área de aprendizagem de máquina. Daqui em diante, exceto o trabalho da próxima seção, serão relacionados os trabalhos que envolvam algum modelo de aprendizagem de máquina, sendo que iniciamos com o trabalho pioneiro (LIU *et al.*, 2015) que apresentou um novo estimador de cardinalidade baseado no uso de redes neurais para aprender a seletividade dos predicados de seleção presentes nas consultas SQL. Isto é, os autores modelaram o problema da estimação de cardinalidade como sendo um problema de aprendizagem supervisionada com vistas a obter um modelo M capaz de detectar a distribuição dos dados $\mathcal{T}$ e, a partir disso, estimar o quão seletivos são os predicados.

Considere uma consulta Q com predicados de seleção $\mathcal{P} = p_1 \ \& \ p_2 \ \& \ ... \ \& \ p_n$ envolvendo somente uma relação R_i, em que cada predicado de seleção p_k no atributo A_j segue o seguinte formato: $l_k \leq R_i.A_j \leq u_k$. Antes da explicação da técnica proposta, é importante salientar que predicados que envolvem a operação de igualdade também estão contemplados nesse formato, pois a seguinte equivalência é válida:

$$R_i.A_j = v \iff v \leq R_i.A_j \leq v \tag{3.1}$$

Para tornar possível a entrada de vários predicados de seleção no modelo M de

Figura 19 – Duas redes neurais adicionais são usadas para estimar os valores de v^+ e v^-.



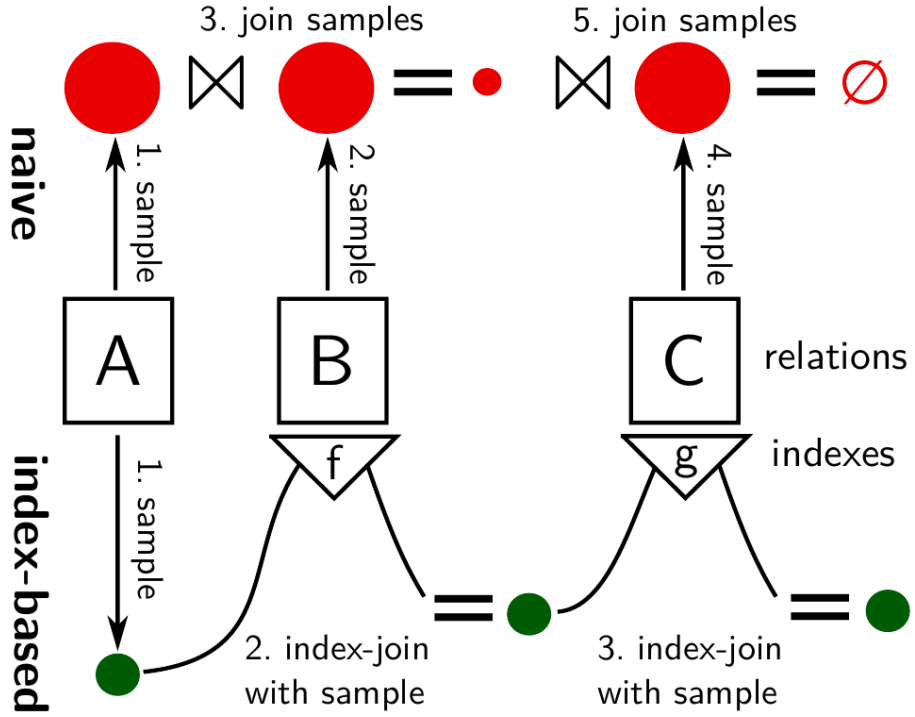
Fonte: (LIU *et al.*, 2015).

aprendizagem de máquina, os autores propuseram a arquitetura mostrada na Figura 18 de uma rede neural aumentada de três camadas que aceita múltiplas entradas. É possível perceber na imagem apresentada que a proposta dos autores envolveu o uso de duas novas funções, φ_1 e φ_2, para possibilitar que a rede aceitasse uma entrada com vários valores. Com isso, tornou-se possível a aplicação desse modelo M para a predição da seletividade de P. Todavia, à luz da limitação de que somente as operações não restritas podem estar presentes nos predicados (i.e., $<$ e $>$ não podem ser utilizados), os autores optaram por usar mais duas redes neurais adicionais, que seguem a arquitetura apresentada na Figura 19, para contornar essa situação baseados no uso da equivalência abaixo:

$$l_k < R_i.A_j < u_k \iff l_k^+ \leq R_i.A_j \leq u_k^- \tag{3.2}$$

Os valores l_k^+ e u_k^- denotam, respectivamente, o menor valor maior que l_k em $\mathcal{V}_{R_i.A_j}$ e o maior valor menor que u_k em $\mathcal{V}_{R_i.A_j}$. Dessa forma, predicados que utilizam operadores estritos podem ser reescritos de tal sorte que todos os predicados presentes nas consultas usam somente operadores não estritos, viabilizando o uso da rede neural aumentada para múltiplas entradas de maneira genérica desde que exista um mecanismo que encontre os valores v^+ e v^- para um dado valor v. Esse mecanismo é formado pelas duas redes neurais simples apresentadas na Figura 19.

Figura 20 – Abordagem padrão e abordagem proposta em (LEIS *et al.*, 2017) para a amostragem de duas operações de junção.



Fonte: (LEIS *et al.*, 2017).

3.6 *Cardinality Estimation Done Right: Index-based Join Sampling*

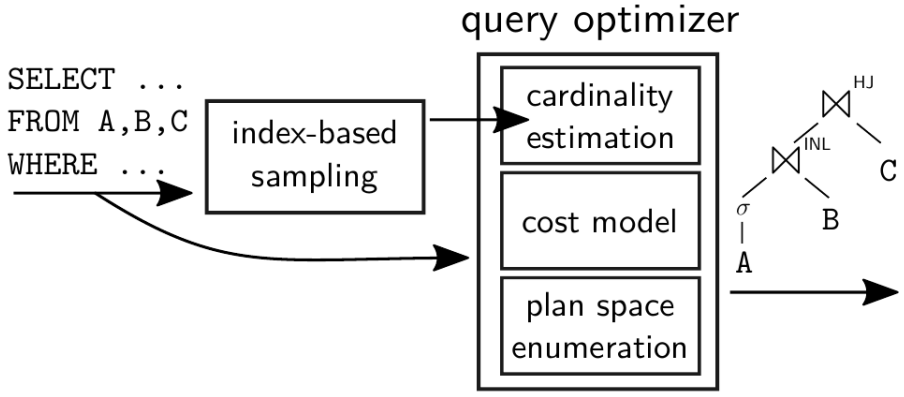
Este trabalho apresenta uma estratégia de amostragem que é baseada no uso de índices para produzir as amostras e, por fim, verificar e corrigir os possíveis erros das estimativas de cardinalidades. Ou seja, é apresentado em (LEIS *et al.*, 2017) uma alternativa à estratégia de amostragem padrão, descrita na Seção 3.2, que visa gerar uma maior acurácia nas estimativas de cardinalidade por meio do uso das estruturas de índices. Na Figura 20, é retratado o processo que ocorre quer seja na abordagem padrão, quer seja na proposta para uma operação de junção.

Dada uma consulta Q com as seguintes operações de junção $R_{i1} \bowtie R_{i2} \bowtie R_{i3}$ cuja condição geral é apresentada na Equação a seguir:

$$R_{i1}.A = R_{i2}.B = R_{i3}.C \tag{3.3}$$

Na abordagem padrão, são gerados os conjuntos de amostras relativas às relações R_{i1}, R_{i2}, R_{i3} obtidos por meio de uma amostragem independente e aleatória de cada uma das relações. Em seguida, esses conjuntos são utilizados para obter as estimativas da primeira operação de junção, no caso a que envolve as relações R_{i1} e R_{i2}. Por fim, o resultado dessa operação é fornecida como entrada para a próxima operação de junção, a qual envolverá a relação R_{i3} para gerar o resultado final. Por outro lado, na abordagem proposta no atual trabalho

Figura 21 – Integração da técnica proposta em (LEIS *et al.*, 2017) na arquitetura dos SGBDs.



Fonte: (LEIS *et al.*, 2017).

relacionado, um conjunto de amostras da relação R_{i1} é alcançado inicialmente. Depois, é usado o índice relativo ao atributo B da relação R_{i2} com o propósito de averiguar as amostras dessa última relação e, então, gerar o resultado intermediário. Este, seguidamente, será fornecido como entrada para a estrutura de índice do atributo C da relação R_{i3} com a mesma intenção e, consequentemente, gerar o resultado final referente as duas operações de junção da consulta Q. Uma ilustração desse processo pode ser visualizado na Figura 20, sendo que a parte superior mostra o processo da abordagem padrão, enquanto que o da abordagem proposta se encontra na parte inferior.

Finalmente, a Figura 21 mostra como esse processo de amostragem via estruturas de índices pode ser integrado na arquitetura dos SGBDs. Basicamente, a principal mudança necessária é a capacidade de que sejam injetadas cardinalidades no módulo de estimação, visto que algumas estimativas serão geradas pela técnica proposta anteriormente à execução do processamento de otimização.

3.7 *Selectivity Estimation for Range Predicates Using Lightweight Models*

O trabalho (DUTT *et al.*, 2019) apresentou um novo estimador de cardinalidade baseado no uso de modelos de aprendizagem de máquina para aprender a seletividade dos predicados de seleção por intervalo. Ou seja, aquelas consultas que possuem predicados $\mathcal{P} = p_1 \ \& \ p_2 \ \& \ ... \ \& \ p_k$ em que cada um dos predicados $p's$ tem o seguinte padrão $l_k \leq R_i.A_j \leq u_k$, donde l_k e u_k definem o intervalo de valores desejados. Semelhante ao caso apresentado na Seção 3.5, especificamente por meio da Equação 3.1, também é possível ter predicados cuja operação é a de igualdade.

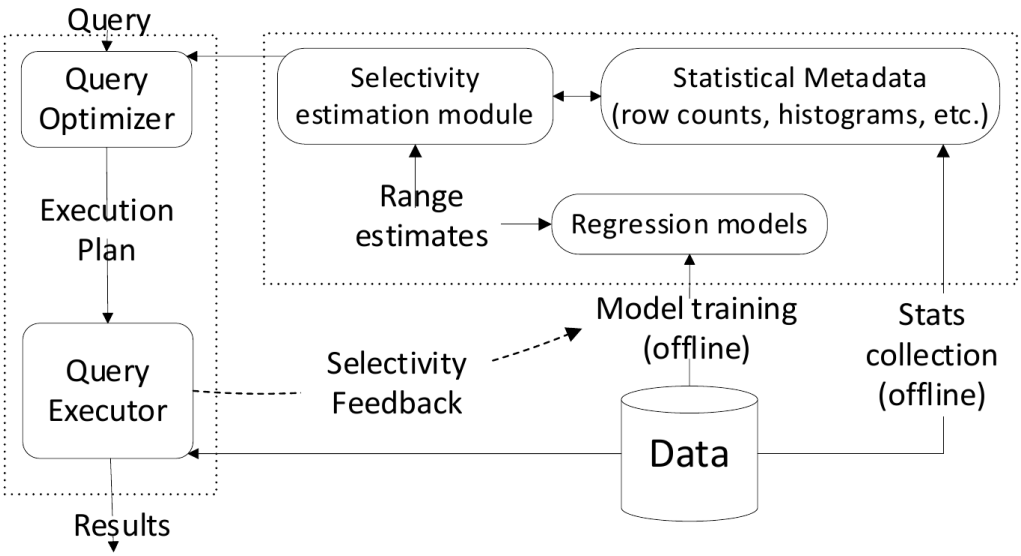
Os autores utilizaram modelos de aprendizagem de máquina que aplicam regressão, tais como as redes neurais e as árvores do tipo *ensemble*. Para isso, realizaram a modelagem das consultas para o problema de aprendizagem de máquina da seguinte forma: considere uma consulta Q com predicados que envolvem os atributos $A_1, A_2, ..., A_n$ de uma conhecida relação R_i. Cada predicado p desses segue o formato apresentado acima. Caso um dos atributos A_j não apareça em nenhum desses predicados, então é criado um novo predicado $\overline{p}$ de tal forma que $\overline{p} = min(\mathcal{D}_{A_j}) \leq R_i.A_j \leq max(\mathcal{D}_{A_j})$. Ou seja, se um atributo não é filtrado por nenhum predicado, então todas as tuplas devem ser incluídas no resultado final sem considerar o valor que aparece no tal atributo. Uma forma de fazer isso é incluir todos os valores possíveis (domínio do atributo) no intervalo de valores desejado. Com todos os predicados construídos, inicia-se o processo de transformar essas informações em uma entrada para o modelo M. Para tal, constrói-se uma tupla $(l_1, u_1, l_2, u_2, ..., l_n, u_m)$ de modo que os dois valores extremos que definem o intervalo de valores de cada um dos predicados, l_k, u_k para o predicado p_k, sejam adjacentes. Ademais, o tamanho dessa tupla será $2 \cdot m$, em que m representa o número de predicados resultantes na consulta Q. Para clarificar, vejamos um exemplo simples. Considere uma relação R com $\mathcal{A} = \{A_1, A_2\}$, em que $\mathcal{D}_{A_1} = \mathcal{D}_{A_2} = \{0, 1, ..., 100\}$. Agora admita que o predicado $\mathcal{P} = 10 \leq R.A_1 \leq 20$ precisa ser avaliado. Dessa forma, sua modelagem será gerada e terá a tupla a seguir como resultado para representá-la: $(10, 20, 0, 100)$. Isto ocorreu devido à adição do predicado $\overline{p}$ para o atributo $R.A_2$.

Condizente com a estratégia adotada neste trabalho, a proposta dos autores é também não excluir o uso das técnicas tradicionais usadas nos SGBDs, por exemplo os histogramas, como pode ser visualizado na arquitetura apresentada na Figura 22. Nesse contexto, as técnicas irão trabalhar em conjunto de maneira orquestrada. Em relação à arquitetura proposta, destaca-se a necessidade de modificar o módulo de execução de consultas para que este repasse os seus resultados para que o novo módulo de treinamento de modelos (na imagem, denominado de *Regression models*) os utilize. De resto, é necessário também que haja uma maneira de injetar as estimativas de cardinalidade no módulo de otimização.

3.8 *Learned Cardinalities: Estimating Correlated Joins with Deep Learning*

Em (KIPF *et al.*, 2019), os autores propuseram uma modelagem do problema de estimativas de cardinalidade para ser utilizado com o modelo de aprendizagem de máquina denominado por eles de MSCN (ZAHEER *et al.*, 2017), demonstrado na Figura 23. A modelagem proposta separa as consultas Q em três conjuntos que serão fornecidos como entrada para o

Figura 22 – Arquitetura proposta para a integração da técnica apresentada no trabalho (DUTT *et al.*, 2019).

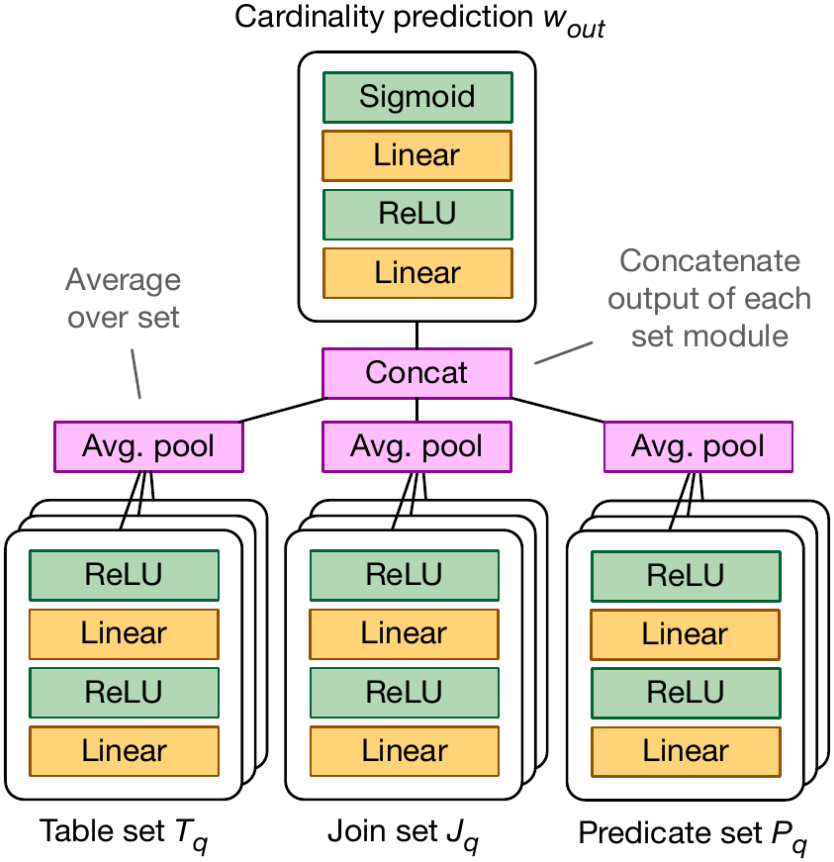


Fonte: (DUTT *et al.*, 2019).

MSCN. Para formar esses conjuntos, as consultas passam por um processo de *featurização*, que veremos logo adiante, juntamente com o uso de uma técnica de amostragem da base de dados para a construção de *bitmaps* que serão úteis no processo de aprendizagem.

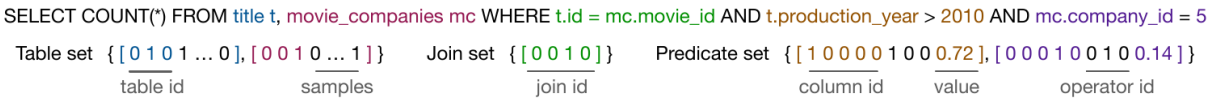
No que diz respeito à modelagem das consultas Q, separam-se as relações envolvidas, os predicados de seleção e os de junção em três conjuntos distintos: conjunto das relações, das junções e dos predicados de seleção. Cada uma das relações do esquema do BD é transformada em um vetor através do uso da técnica de modelagem *one-hot enconding*. Nesse tipo de transformação, um vetor de tamanho n é criado, em que n é a quantidade de relações no BD. Cada uma das entradas desse vetor é um *bit* e é associada a uma relação. Isto é, suponha que há três relações R_1, R_2 e R_3 no esquema. Para a relação R_1, o vetor que o representará será $[1,0,0]$. Por sua vez, o vetor da relação R_2 é $[0,1,0]$. E assim adiante. O conjunto de relações é composto pelas representações *one-hot enconding* das relações que são referenciadas na consulta. Adicionalmente, como algo optativo, na fase de treinamento, é incluído também um *bitmap* que indica quais tuplas de um dado conjunto amostral dos dados satisfazem as condições impostas na consulta que está sendo modelada. Já no que diz respeito ao conjunto das junções, tem-se que ele é formado pelas representações *one-hot enconding* de cada uma das condições de junção que ocorrem na consulta. Por fim, o conjunto dos predicados é constituído por vetores que refletem a estrutura de cada um dos predicados. Os atributos e os operadores que aparecem nos predicados são também caracterizados por meio da técnica *one-hot enconding* e os valores de seleção são normalizados (no próximo capítulo, mais detalhes são abordados sobre esse processo). Um

Figura 23 – Arquitetura do modelo MSCN proposto em (KIPF *et al.*, 2019).



Fonte: (KIPF *et al.*, 2019).

Figura 24 – Exemplo de uma consulta SQL baseado no IMDB e sua respectiva modelagem de acordo com a abordagem de (KIPF *et al.*, 2019).



Fonte: (KIPF *et al.*, 2019).

exemplo de uma consulta é ilustrado na Figura 24 acompanhada de sua modelagem.

Após os três conjuntos que representam as consultas estarem construídos, cada um deles é submetido a uma rede neural. Sucessivamente, realiza-se uma média da saída dessas redes através da técnica de *pooling*, obtendo uma representação compacta para a consulta. Esta, por sua vez, é fornecida como entrada para a última rede neural que fornecerá como saída a predição da cardinalidade. A ilustração da Figura 23 esclarece todo esse processo.

Por fim, os autores também propuseram o Algoritmo 2 que é um gerador de consultas com o intuito de gerar um conjunto de treinamento X rotulado para o modelo de predição. Em resumo, o referido algoritmo gera números aleatórios uniformes para realizar a escolha das relações e os predicados de seleção presentes em uma consulta, executa a consulta gerada de

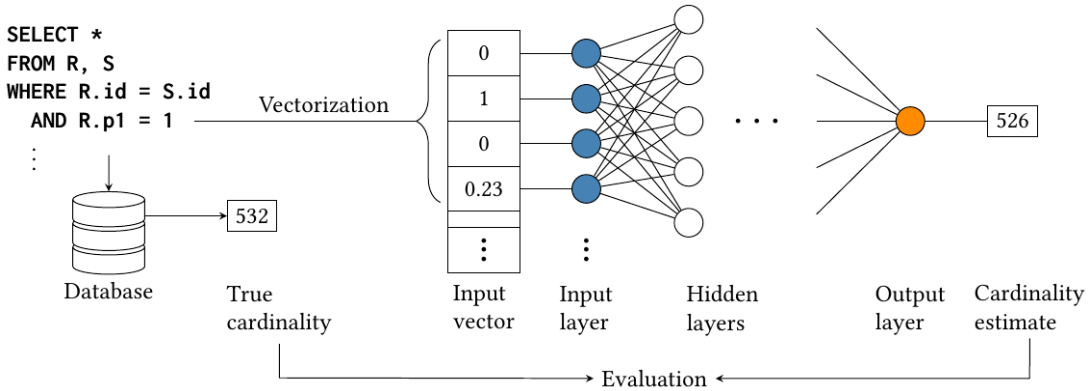
forma a obter sua cardinalidade e, em seguida, a adiciona no conjunto de treinamento que está sendo gerado. Por fim, retorna esse conjunto de treinamento X rotulado para ser utilizado no treinamento do modelo de aprendizagem de máquina.

Algoritmo 2: Gerador de Conjunto de Treinamento. Retirado de (KIPF *et al.*, 2019).

Entrada: Conjunto de relações $\mathcal{R}$, quantidade de consultas n
Saída: Conjunto de dados de treinamento X rotulado

início
  $X \leftarrow \emptyset$
  para $i = 1 \rightarrow i = n$ **faça**
    $R_0 \leftarrow$ *Seleciona-se aleatoriamente uma relação referenciada por outras relações no conjunto* $\mathcal{R}$
    $\mathcal{R}_i \leftarrow R_0$
    $N_{\mathcal{J}} \leftarrow$ *Seleciona-se aleatoriamente uma quantidade de junções entre* 0 *e* 2 *para a nova consulta* Q_i
    $\mathcal{J}_i \leftarrow \emptyset$
    para $j = 1 \rightarrow j \leq N_{\mathcal{J}}$ **faça**
      $R_j \leftarrow$ *Seleciona-se aleatoriamente uma relação referenciada por pelo menos uma relação no conjunto* $\mathcal{R}_i$
      $\mathcal{R}_i \leftarrow \mathcal{R}_i + R_j$
      $\mathcal{J}_i \leftarrow \mathcal{J}_i +$ *condição de junção entre a relação* R_j *e a relação de* $\mathcal{R}_i$
    fim
    $\mathcal{P}_i \leftarrow \emptyset$
    para $j = 1 \rightarrow j \leq |\mathcal{R}_i|$ **faça**
      $N_{\mathcal{P}} \leftarrow$ *Seleciona-se aleatoriamente uma quantidade de predicados entre* 0 *e a quantidade de atributos não chaves na relação* $\mathcal{R}_i[j]$
      para $k = 1 \rightarrow k \leq N_{\mathcal{P}}$ **faça**
        $P_k \leftarrow$ *Seleciona-se aleatoriamente um atributo não chave de* $\mathcal{R}_i[j]$*, um operador matemático e valores constantes para formar um predicado de seleção*
        $\mathcal{P}_i \leftarrow \mathcal{P}_i + P_k$
      fim
    fim
    $Q_i \leftarrow$ *Formar uma nova consulta usando* $\mathcal{R}_i$*,* $\mathcal{J}_i$ *e* $\mathcal{P}_i$
    $c \leftarrow$ *Executar a consulta* Q_i *e obter sua cardinalidade*
    $X \leftarrow X + (Q_i, c)$
  fim
  retorna X;
fim

Figura 25 – Modelagem do problema de estimar as cardinalidades por meio de aprendizagem supervisionada.



Fonte: (WOLTMANN *et al.*, 2019).

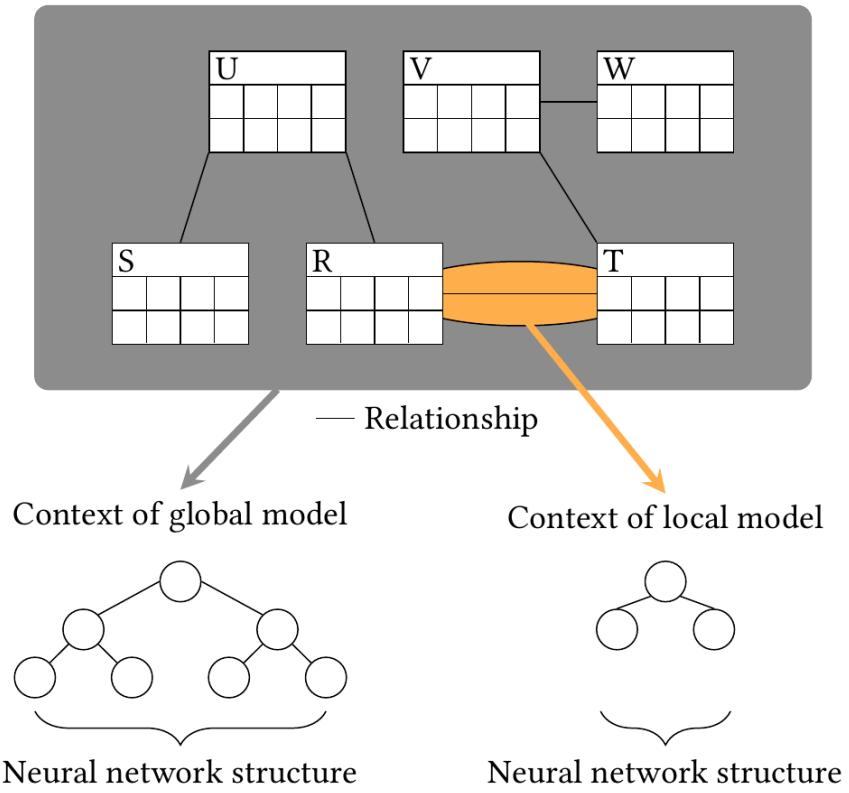
3.9 *Cardinality Estimation with Local Deep Learning Models*

O último trabalho relacionado (WOLTMANN *et al.*, 2019) trouxe uma abordagem bastante similar com o método do trabalho apresentado na seção anterior. Ou seja, as estimativas de cardinalidades são obtidas por meio das predições de um modelo M de aprendizagem de máquina treinado com as representações das consultas Q, observe esse processo na Figura 25. As duas principais diferenças para o trabalho anterior são o modelo subjacente e a modelagem das consultas.

Enquanto que o MSCN foi utilizado no trabalho anterior com o intuito de aprender as distribuições de dados de todo o esquema de um BD específico, os autores do presente trabalho relacionado optaram por utilizar uma única rede neural para cada relacionamento, como demonstrado na Figura 26. Os autores definiram o seu mecanismo proposto como sendo uma abordagem local, pois o modelo criado lida somente com as distribuições das relações locais (específicas). Por outro lado, o trabalho apresentado em (KIPF *et al.*, 2019) utilizou uma abordagem global, visto que todas as distribuições das relações devem ser capturadas pelo modelo treinado.

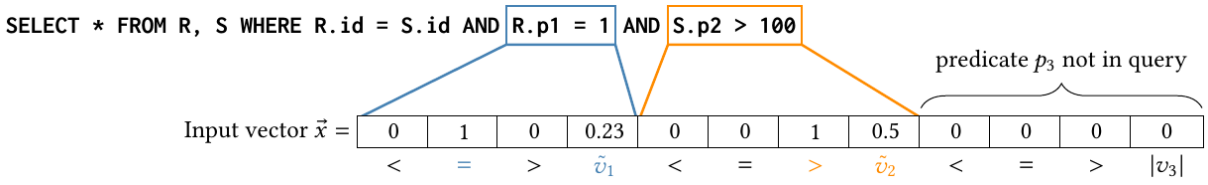
Por consequência dessa mudança de abordagem global para local, a modelagem das consultas é diferente entre ambos os trabalhos. Como no atual trabalho as relações que se deseja aprender a distribuição dos dados $\mathcal{T}$ já estão definidas (i.e., cada modelo M é treinado para um conjunto específico de relações), então não há a necessidade da entrada fornecida para o modelo de aprendizagem de máquina possuir informações sobre as relações envolvidas na consulta. Em contrapartida, o mecanismo de otimização de consulta deve ser capaz de definir qual o modelo deve ser utilizado a partir das relações presentes na consulta submetida, por isso vários modelos

Figura 26 – Ilustração da diferença entre a abordagem global e a local.



Fonte: (WOLTMANN *et al.*, 2019).

Figura 27 – Exemplo de uma consulta SQL e sua respectiva modelagem de acordo com a abordagem de (WOLTMANN *et al.*, 2019).



Fonte: (WOLTMANN *et al.*, 2019).

podem ser utilizados no momento de gerar as estimativas de cardinalidade para uma consulta. Com isso, a entrada deve refletir somente os predicados de seleção, os quais são representados do mesmo modo que o usado no trabalho relacionado anterior, como pode ser visualizado no exemplo apresentado na Figura 27.

3.10 Discussão

Nesta seção, os trabalhos relacionados descritos nas seções anteriores desse capítulo serão resumidos em quatro quesitos: o problema abordado, a estratégia de estimação utilizada, quais operações relacionais são suportadas, e finalmente, o critério de lidar com a incerteza das estimativas. Ademais, os aspectos relativos às vantagens e às desvantagens da utilização de cada

técnica serão discutidos.

O trabalho proposto em (KOOI, 1980) usa os histogramas para calcular a seletividade das operações relacionais e, assim, gerar as estimativas de cardinalidade. Existem dois tipos de histogramas: unidimensional e multidimensional. Não obstante, devido ao seu alto custo de geração e manutenção, os histogramas multidimensionais não são muitos utilizados na prática, uma vez que esse custo de criá-los e mantê-los não supera a vantagem por eles disponibilizada, que é conhecer a distribuição dos dados de mais de um atributo simultaneamente. Já em relação aos unidimensionais, destaca-se que são utilizadas por muitos dos SGBDs mais aceitos na academia e na indústria, inclusive no qual será utilizado nos experimentos deste trabalho, o PostgreSQL, devido à sua simplicidade e eficiência. Dentre as suas limitações, é relevante pontuar que essa técnica necessita assumir a premissa IVA, recaindo naquelas fontes de erros supracitadas. Por fim, nessa técnica também não há uma maneira de medir se as estimativas geradas estão representativas para os valores reais de cardinalidade.

Os autores de (LIPTON *et al.*, 1990) propuseram usar uma técnica de amostragem para identificar e corrigir as possíveis imprecisões nas seletividades obtidas por meio dos histogramas. Ou seja, uma amostra da base de dados reais é obtida para obter as características desses dados e, assim, gerar novas estimativas mais próximas dos valores reais. Todavia, o processo de amostragem demanda um alto custo associado devido à necessidade de se acessar os dados armazenados nos SGBDs, mesmo nos casos em que a quantidade de dados seja pequena. Além do mais, muitas vezes há a necessidade de usar uma grande porção de dados dos SGBDs para evitar que a amostra seja enviesada. Com isso, essa técnica não foi muito adotada na prática dos SGBDs comerciais. A proposta de (LEIS *et al.*, 2017) foi de utilizar as estruturas de índices dos atributos para fazer a amostragem dos dados, dessa forma diminuindo o custo dessa técnica. Mesmo com essa atenuação, o custo ainda é alto se comparado com a técnica que usa os histogramas.

Baseados na ideia de identificar os erros de estimação ao se verificar os dados reais, dois outros trabalhos, um denominado de LEO (STILLGER *et al.*, 2001) e outro de *Pay As You Go* (CHAUDHURI *et al.*, 2008), propuseram um método que realiza o monitoramento das cardinalidades presentes nos PEs no tempo de execução com vistas a obter as cardinalidades reais e, consequentemente, injetá-las nas próximas consultas que envolvam as cardinalidades já vistas anteriormente. Esses trabalhos apresentaram uma melhora no desempenho dos SGBDs, mas eles sofrem devido ao fraco poder de generalização, isto é, a melhora só é possível naquelas

consultas que já foram realizadas algumas vezes. Naquelas em que nunca foram realizadas, os problemas mencionados anteriormente continuam.

Já os trabalhos relacionados (LIU *et al.*, 2015), (DUTT *et al.*, 2019), (KIPF *et al.*, 2019) e (WOLTMANN *et al.*, 2019) estão diretamente relacionados ao método proposto, pois todos eles aplicaram alguma técnica de aprendizagem de máquina para o contexto de otimização de consultas. (LIU *et al.*, 2015) propôs o uso de redes neurais para aprender a seletividade dos predicados que envolviam os atributos de uma única relação. De maneira análoga, (DUTT *et al.*, 2019) aplicou esse modelo para estimar a seletividade dos predicados que usavam condições por intervalo de valores. Ambos melhoraram as estimativas, mas não apresentaram uma maneira de lidar com as junções. Já (KIPF *et al.*, 2019) e (WOLTMANN *et al.*, 2019) propuseram uma abordagem que lida com ambas as operações relacionais. Os dois trabalhos também apresentaram uma melhora nas estimativas de cardinalidade geradas. Entretanto, nenhum desses trabalhos apresentaram uma maneira de gerenciar a confiabilidade dessas estimativas.

Por fim, na Tabela 4 é apresentada uma comparação dos trabalhos relacionados com o trabalho desta dissertação. O método proposto por esse trabalho tem o intuito de predizer as cardinalidades para ambas as operações relacionais de seleção e junção usando o *LightGBM* como estratégia para gerar as estimativas. Por apresentar uma estratégia semelhante com o método proposto, o trabalho (KIPF *et al.*, 2019) será usado como base em conjunto com a técnica do PostgreSQL na avaliação experimental. Além disso, optamos também pela escolha desse modelo tendo em vista a proposição de uma maneira de gerenciar a incerteza das estimativas de cardinalidade geradas por meio do modelo que iremos comentar mais no próximo capítulo.

Tabela 4 – Comparação entre os trabalhos relacionados e o trabalho desta dissertação.

Trabalho	**Problema Abordado**	**Estratégia de Estimação**	**Suporte a Operações**	**Incerteza**
(KOOI, 1980)	Seletividade	Histograma	$\sigma, \bowtie$	
(LIPTON *et al.*, 1990)	Seletividade	Amostragem	$\sigma, \bowtie$	✓
(STILLGER *et al.*, 2001)	Cardinalidade	LEO	$\sigma, \bowtie$	
(CHAUDHURI *et al.*, 2008)	Cardinalidade	*A Pay As You Go*	$\sigma, \bowtie$	
(LIU *et al.*, 2015)	Seletividade	Rede Neural	σ	
(LEIS *et al.*, 2017)	Seletividade	Amostragem	$\sigma, \bowtie$	✓
(DUTT *et al.*, 2019)	Seletividade	Rede Neural	σ	
(KIPF *et al.*, 2019)	Cardinalidade	Rede Convolucional	$\sigma, \bowtie$	
(WOLTMANN *et al.*, 2019)	Cardinalidade	Rede Neural	$\sigma, \bowtie$	
Este trabalho	Cardinalidade	*LightGBM*	$\sigma, \bowtie$	✓

3.11 Conclusão

Os trabalhos relacionados ao tema desta dissertação mais relevantes da literatura foram apresentados nesse capítulo. Foram listadas as principais características desses trabalhos e, na seção anterior, foi realizado uma comparação entre esses trabalhos e o proposto nesta dissertação com o objetivo de situar o contexto no qual o método proposto se encaixa. Nesse sentido, o presente trabalho se propõe a melhorar as estimativas de cardinalidade através do uso do *LightGBM* e, além disso, prover uma maneira de gerenciar a incerteza dessas estimativas de cardinalidade. Por fim, vale lembrar que o mecanismo proposto em (KIPF *et al.*, 2019) juntamente com a técnica de histogramas usada no PostgreSQL serão utilizados como os métodos bases de comparação.

4 REFINAMENTO DAS ESTIMATIVAS DE CARDINALIDADE

O presente capítulo é dedicado aos aspectos atinentes à estratégia proposta por meio desta dissertação. A abordagem apresentada visa corrigir as imprecisões das estimativas de cardinalidade a partir do uso de modelos de aprendizagem de máquina para gerar essas estimativas. Especificamente, a proposta adota o *LightGBM* como sendo o modelo a ser utilizado.

No início, a Seção 4.1 apresenta uma visão macro acerca do mecanismo proposto. Já na Seção 4.2, é apresentada a organização da arquitetura para que a técnica proposta seja integrada aos SGBDs. Depois, na Seção 4.3, são abordados os aspectos necessários à modelagem do problema de estimação de cardinalidades para o contexto de aprendizagem de máquina. Posteriormente, é apresentada uma breve descrição do processo de aprendizagem e evolução do método proposto nas Seções 4.4 e 4.5, respectivamente. No final, discute-se como o gerenciamento da incerteza das estimativas geradas é realizado.

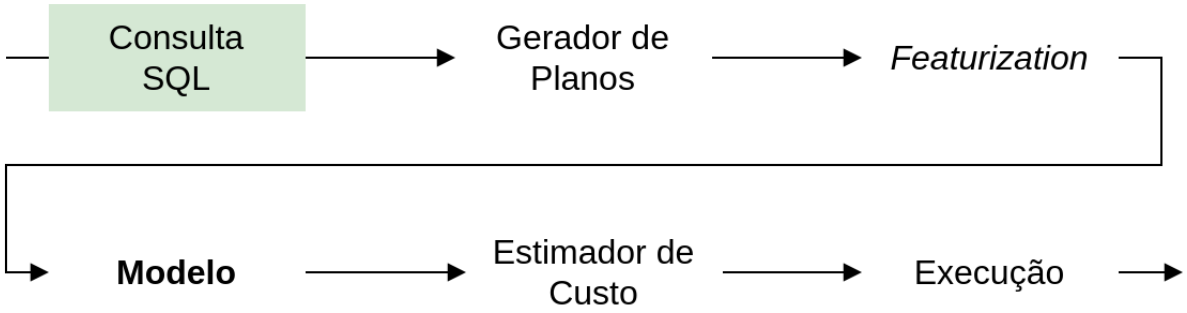
4.1 Visão Geral

Iniciando com o ponto de vista mais geral apresentado na Figura 28, a técnica proposta nesse trabalho visa utilizar um modelo *M* treinado por meio do algoritmo *LightGBM* para gerar as estimativas de cardinalidade. A Figura apresenta os passos nos quais uma consulta SQL vai percorrer para que sua resposta seja produzida pelo SGBD. No início, vários PEs equivalentes à consulta SQL são gerados. Como estes planos não podem ser diretamente repassados ao modelo preditivo, como explicado mais a frente, um passo intermediário de modelagem precisa ser realizado. Esse passo está representado pela fase de *Featurization*. Com essa transformação feita, as representações dos planos são repassadas para o modelo a fim de que sejam geradas as predições de cardinalidade. De posse dessas estimativas, o custo de cada plano é estimado e aquele que apresentar o menor custo é executado para que o resultado seja produzido.

4.2 Arquitetura

Na visão de alto nível apresentada anteriormente, partiu-se do pressuposto de que um modelo *M* já treinado é utilizado para predizer as estimativas de cardinalidade dos PEs gerados a partir de uma consulta SQL. No entanto, há uma fase anterior de treinamento do modelo que é necessária, assim como acontece nos SGBDs que têm como base a estratégia baseada

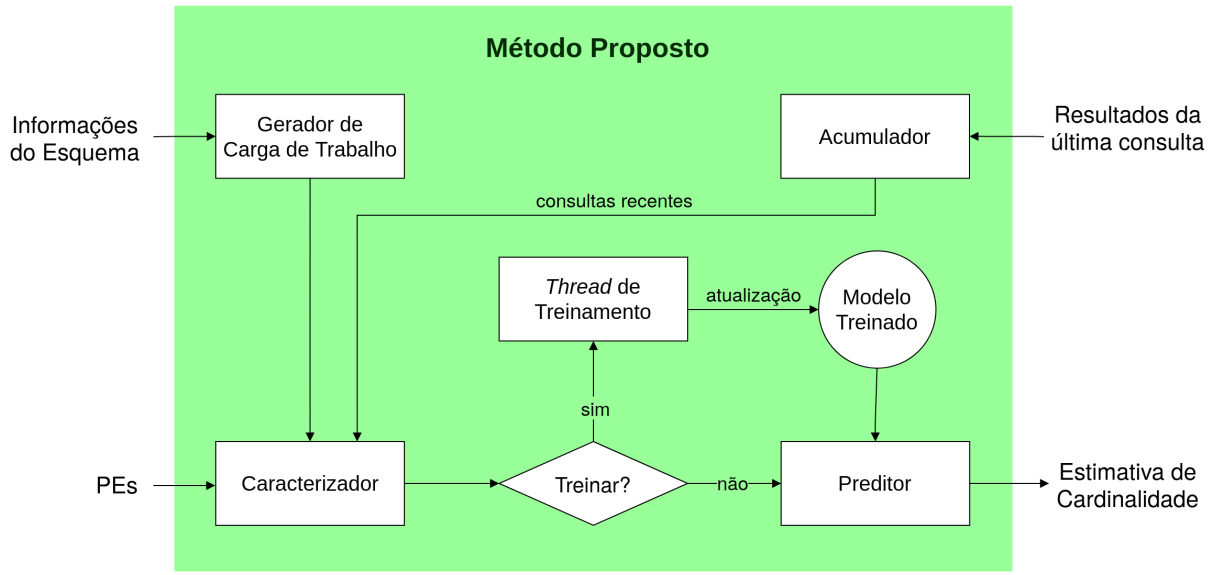
Figura 28 – Visão geral da abordagem do método proposto.



em histogramas para estimar as cardinalidades das consultas impostas a eles. Nestes sistemas, realiza-se uma fase anterior à execução das consultas que é responsável por determinar quais os histogramas são criados e para quais atributos. Após essas estruturas estarem prontas para uso, o otimizador de consultas as utiliza para estimar as cardinalidades das consultas. Na técnica proposta nesse trabalho, também há essa fase, mas com algumas diferenças, como pode ser visualizado na Figura 29 na qual são apresentados dois fluxos de dados. O primeiro é relativo à fase de treinamento do modelo. Ao contrário de criar algumas estruturas, um conjunto de consultas é utilizado para treinar o modelo M. Caso haja um histórico de consultas já executadas armazenado no SGBD, esse histórico vai formar o conjunto que vai ser usado para a fase atual. Caso contrário, um conjunto de consultas vai ser gerado por meio de um mecanismo aleatório usando as informações do esquema do BD para o qual as consultas são submetidas. Por sua vez, essas consultas são modeladas e executadas com o propósito de obter as suas cardinalidades reais e, com isso, treinar o modelo M. Sucessivamente, no segundo fluxo, é demonstrado que as consultas submetidas pelos usuários já transformadas em PEs equivalentes podem utilizar o modelo M para realizar as estimativas. É importante esclarecer que enquanto o modelo não estiver treinado, os métodos convencionais de estimativas de cardinalidade como os histogramas, por exemplo, são utilizados nas estimativas. Neste fluxo, há a presença de um módulo Acumulador responsável por armazenar o resultado dessas consultas executadas de modo a possibilitar que estas sejam utilizadas em uma nova fase de treinamento do modelo M.

Depois dessa apresentação da organização do método proposto de maneira independente, a Figura 30 ilustra a organização do método proposto já integrado com a arquitetura tradicional de um SGBD. Esta arquitetura é dada em duas partes: uma verde (aquela já apresentada) e uma azul (do SGBD). Como a primeira já foi descrita acima, vamos focar na parte azul e, principalmente, na interação entre os componentes. A primeira observação importante é que os elementos referentes ao processo de gerar um modelo de aprendizagem de máquina e usar esse

Figura 29 – Arquitetura somente do método proposto.

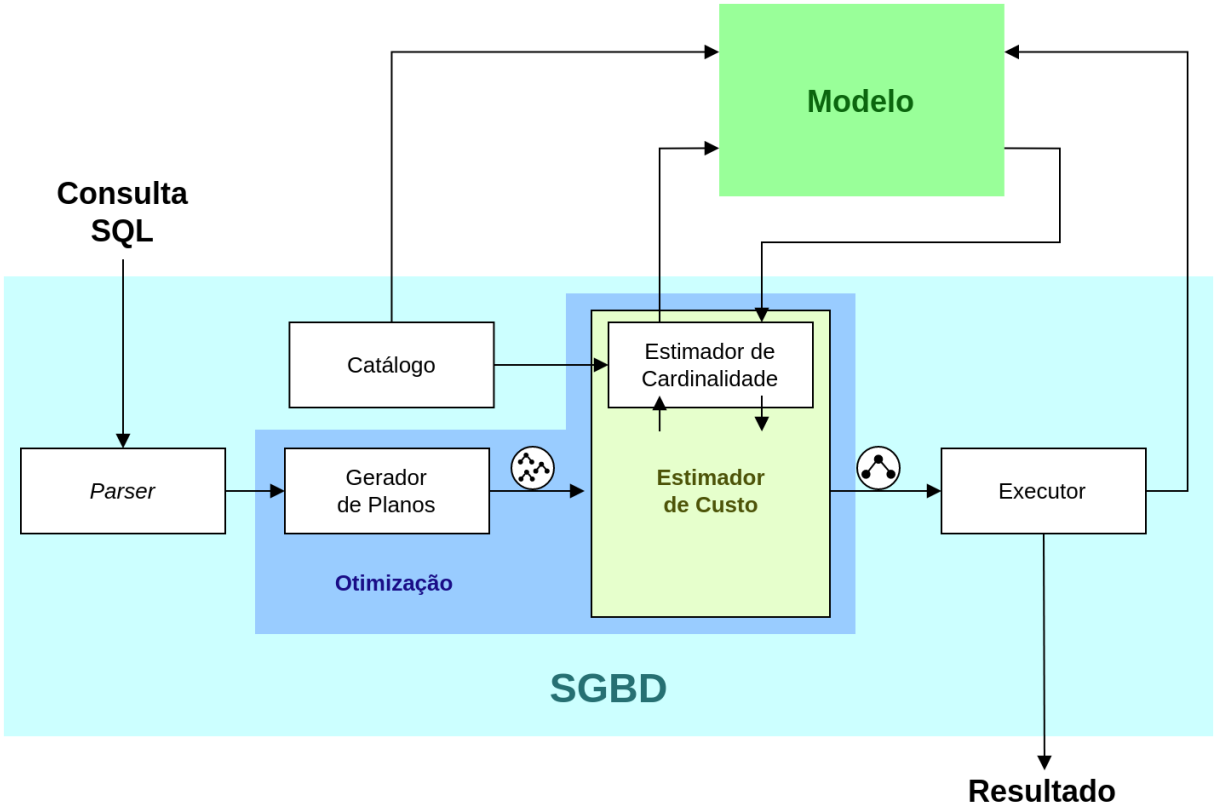


modelo para gerar as estimativas de cardinalidade estão situados fora da fronteira dos SGBDs, isto é, esses módulos são implementados de modo independente da arquitetura dos SGBDs, sendo que a interação entre o sistema e eles ocorre através de uma interface bem definida que iremos comentar logo abaixo. Já a parte azul apresenta os módulos presentes na arquitetura dos SGBDs. As setas representam o fluxo de dados entre os módulos, tanto da parte azul quanto da verde, para que todo o mecanismo funcione. Por exemplo, o Estimador de Cardinalidade precisa se comunicar com o módulo Catálogo com o intuito de obter informações sobre o esquema do BD que está sendo utilizado atualmente.

Acerca da interface entre as duas partes da imagem, é necessário ressaltar que ela foi adotada tendo em vista a diminuição do impacto das mudanças necessárias para a arquitetura dos SGBDs tradicionais que usam a técnica de histogramas, por exemplo. Isto é, no momento de realizar a integração entre o SGBD e o mecanismo proposto, é uma boa abordagem que poucas mudanças nos módulos dos SGBDs sejam necessárias para que esses sistemas estejam apropriados para receber a técnica proposta. Dessa forma, deixar os módulos responsáveis pelo processo de aprendizagem de máquina fora do âmbito dos SGBDs auxilia nesse processo, visto que somente a implementação de uma nova interface nos SGBDs precisa ser feita.

Como pode ser observado na Figura 30, há três pontos de interação entre o modelo de aprendizagem e o SGBD. São estes três pontos que precisam ser implementados na arquitetura dos SGBDs que adotam a técnica de histogramas. A primeira interação necessária é entre o módulo de Catálogo e o do modelo. Essa interação é primordial tendo em vista a necessidade

Figura 30 – Proposta de arquitetura para integrar o método proposto em um SGBD.



do modelo conhecer as informações atinentes ao esquema do BD. O módulo Estimador de Cardinalidade precisa interagir com o modelo também para que as estimativas de cardinalidades sejam calculadas. Por fim, o módulo Executor precisa se comunicar com o módulo do modelo para que o resultado das consultas sejam armazenadas, vislumbrando o uso destas no treinamento do modelo.

4.3 Modelagem

O problema de estimar as cardinalidades das operações relacionais em um PE não pode ser diretamente aplicado no contexto das técnicas de aprendizagem de máquina, em particular no algoritmo *LightGBM*, visto que os PEs formam uma estrutura hierárquica, enquanto que os modelos de aprendizagem de máquina recebem como entrada um objeto linear (por exemplo, um valor numérico). Portanto, surge a necessidade de realizar a modelagem desse problema em um formato no qual é possível ser utilizado o modelo gerado pelo *LightGBM*. No contexto de aprendizagem de máquina, esse processo de transformação é conhecido como featurização (que advém do inglês, *featurization*). Basicamente, as principais características do

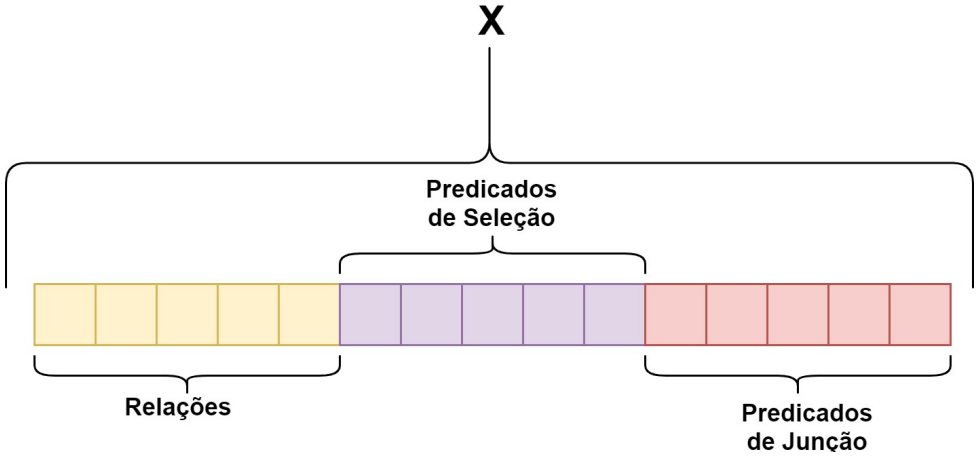
objeto que está sendo modelado são extraídas para que as *features* que são fornecidas ao modelo sejam criadas.

A modelagem a ser adotada é dependente da maneira como o problema é enquadrado no contexto de aprendizagem de máquina, razão pela qual o primeiro passo para definir qual modelagem empregar é escolher qual o tipo de problema de aprendizagem de máquina que o problema de estimar as cardinalidades é contextualizado. Devido ao objetivo ser predizer os valores numéricos das cardinalidades das consultas e a construção de um conjunto de dados de treinamento rotulados ser possível, então adotamos o método de regressão, ou seja, uma abordagem supervisionada para o problema de gerar as estimativas. Isto posto, agora é possível assumir um processo de transformar consultas SQLs em uma entrada linear para o modelo M.

O caminho que adotamos para modelar o problema é similar à estratégia descrita em (KIPF *et al.*, 2019). Todavia, conforme descrito no capítulo anterior, naquele trabalho os autores utilizaram o conceito de *multi-set* e, portanto, realizaram a modelagem usando três conjuntos separados, sendo cada um para representar as relações, os predicados de seleção e os de junções de uma dada consulta Q, respectivamente. Ou seja, três modelos são treinados para cada um desses conjuntos e, posteriormente, as suas representações são agrupadas em um outro modelo, o qual realiza a tarefa de predizer a cardinalidade.

No método proposto, um modelo M de aprendizagem de máquina é treinado com o intuito de predizer as cardinalidades das consultas, ou seja, todas as informações de uma consulta Q são agrupadas em uma única entrada para o modelo, diferente da abordagem do trabalho citado acima. Sendo assim, esse modelo tem que aproximar a distribuição dos dados $\mathcal{T}$ para que consiga realizar as predições com uma boa acurácia. Desta forma, necessita-se pensar em quais informações das consultas precisam ser repassadas ao modelo M de tal sorte que ele seja capaz de realizar sua atividade de predição. De maneira lógica, é indiscutível que as relações envolvidas nas consultas devem prover informações importantes para o modelo M, uma vez que elas armazenam os dados do BD e a cardinalidade das consultas dependerão deles. Analogamente, os predicados de seleção trazem importantes dicas para que o modelo M faça as predições, porque eles são capazes de alterar a cardinalidade das consultas, dependendo do fator de seletividade que lhes é associado, assim como os predicados de junção, uma vez que uma junção pode ser decomposta em um produto cartesiano seguido por uma operação de seleção. Portanto, essas três informações são extraídas das consultas e precisarão ser transformadas em uma maneira que possa ser utilizada no modelo preditivo.

Figura 31 – Composição do vetor que é utilizado para representar as consultas para o modelo M.



Conforme explicado no início dessa seção, os modelos de aprendizagem de máquina lidam com valores numéricos. Portanto, cada consulta SQL é transformada em um vetor de valores numéricos que tem a estrutura apresentada na Figura 31, em que as primeiras posições são utilizadas para retratar as relações que foram utilizadas na consulta. Nas posições seguintes, os predicados de seleção são representados e as posições finais do vetor correspondem aos predicados das operações de junção que estão presentes na consulta.

Sequencialmente, é descrito o processo de modelagem das três partes importantes das consultas, tendo início pelas relações. Estas podem ser visualizadas, no contexto de aprendizagem de máquina, como sendo categorias distintas nas quais os dados estão organizados. Uma técnica para esse tipo de modelagem bastante conhecida na literatura de aprendizagem de máquina é a *one-hot enconding*. Em resumo, essa técnica transforma um valor categórico em um vetor numérico que pode ser fornecido para o modelo M. Dadas n categorias, um vetor de valores binários (i.e., *bits*) de tamanho n é associado a cada uma das categorias, sendo que todas as entradas desse vetor é 0, exceto uma entrada que é definida em 1. A posição deste valor é determinada pela ordem das categorias, ou seja, a categoria associada à posição i tem um valor 1 unicamente na posição i do vetor, pois as outras são determinadas para o valor 0. Por exemplo, considere que há três categorias: A, B e C. O vetor $[1,0,0]$ representa a categoria A, $[0,1,0]$ a B e assim por diante. Em síntese, cada relação em uma consulta é representada por esse vetor e eles, por sua vez, são concatenados nas posições iniciais do vetor que representa a consulta total. Propomos uma variante para essa técnica visando diminuir o uso de memória e da dimensão do vetor resultante da consulta. Ao contrário de pegar os vetores de cada relação e concatená-los diretamente no vetor final, antes realizamos uma operação OU, bit a bit, entre os vetores, para então obter um único vetor de tamanho n e este é utilizado para representar as relações no vetor

representativo da consulta. Com o exemplo apresentado na Seção 4.3.1 adiante deixará claro essa alternativa e a sua diferença para a anterior.

Em relação à modelagem dos predicados de seleção, primeiramente é necessário analisar as informações presentes em cada predicado. De acordo com o definido na Seção 2.2.2, os predicados possuem o seguinte formato: $R_i.A_j\ \theta\ v$, em que $\theta \in \{<, \leq, =, \neq, >, \geq\}$ e v representa um valor constante. Ou seja, há três informações que precisam ser modeladas, a saber: o atributo $R_i.A_j$ envolvido na seleção, o operador matemático θ e o valor numérico v. Semelhante à transformação das relações, os atributos também são considerados categorias e, portanto, são representados por um vetor que segue a técnica *one-hot enconding*. É importante esclarecer que a versão alternativa proposta para representar as relações não vale para os atributos, porque mais de um predicado envolvendo o mesmo atributo e com um valor constante v diferente pode ocorrer em uma mesma consulta, o que inviabiliza o uso dessa variante. Esse problema não ocorre na modelagem das relações, pois aparecer uma ou mais vezes vai ser detectado por meio das condições de junções. Novamente, os operadores matemáticos também são representados por vetores advindos da técnica *one-hot enconding*. Por fim, os valores v's não precisam passar por uma transformação, pois já são numéricos. No entanto, é necessário fazer uma normalização desse valor porque o domínio dos atributos pode ter um intervalo de valores com grande amplitude, isto é, podem existir muitos valores entre o valor mínimo e o máximo. É de conhecimento na literatura de aprendizagem de máquina que possuir valores com magnitudes diferentes pode causar problemas na hora de treinar um modelo. Para evitar isso, realizamos a normalização min-max, a qual segue a seguinte equação:

$$v\_normalizado = \frac{v - v\_min}{v\_max - v\_min} \tag{4.1}$$

Essencialmente, a normalização do valor v envolve subtrair o valor mínimo (v_min) e depois dividir o valor resultante da operação anterior pela diferença entre os valores máximo (v_max) e mínimo. Dessa forma, os valores dos predicados estão no intervalo $[0, 1]$ ou $[-1, 1]$, dependendo se há valores negativos envolvidos. Formalizado o processo de transformar o atributo $R_i.A_j$, o operador matemático θ e o valor v, os predicados são representados por meio de um vetor para representar o atributo, um outro vetor para representar o operador e um valor numérico normalizado. Então, os predicados de seleção são representados pela concatenação desses valores em um único vetor, o qual é adicionado nas posições intermediárias no vetor resultante da consulta.

Finalmente, temos a modelagem das condições de junção. Segundo apresentado na Seção 2.2.2, essas condições são formatadas da seguinte maneira: $R_{i_1}.A_{j_1} = R_{i_2}.A_{j_2}$, para uma operação de junção $R_{i_1} \bowtie R_{i_2}$ envolvendo os atributos A_{j_1} e e A_{j_2} das respectivas relações. Ou seja, as consultas envolvem somente os conhecidos *equi-joins*. Por conseguinte, a informação importante dessa parte é o relacionamento entre as relações que está sendo declarado por meio dos atributos envolvidos na condição. Assim, é utilizado um vetor *one-hot encoding* para cada relacionamento de junção. Para tanto, é necessário fazer a observação que há dois tipos de relacionamento de junção entre as relações, a saber: aqueles que envolvem os atributos que são chaves primárias e estrangeiras e aqueles relacionamentos muitos para muitos (m-n), ou seja, os que não têm a ligação de chave primária-estrangeira. No primeiro caso, é fácil conhecer esses relacionamentos através das restrições de integridades que estão armazenadas no catálogo dos SGBDs. Em contrapartida, pode haver uma explosão combinatorial dos possíveis relacionamentos de junções m-n dos atributos e, portanto, esses não são conhecidos a priori totalmente. Com isso, esse tipo de relacionamento é conhecido à medida em que o modelo M de aprendizagem de máquina vai evoluindo e modelando os relacionamentos envolvidos nas consultas da carga de trabalho. Sumariamente, as consultas que envolvem junções entre chaves primárias e estrangeiras são sempre estimadas pelo modelo M treinado. Os outros relacionamentos são estimados somente quando o modelo M já tiver sido treinado para reconhecê-los. Caso contrário, o estimador padrão do SGBD é utilizado em seu lugar. Posto isto, temos então que as condições de junção são representadas por um vetor obtido através da técnica *one-hot enconding*. Este vetor é adicionado nas posições finais do vetor representativo da consulta de forma a representar a condição de junção associada. Com isso, todo o vetor que vai ser usado para representar uma consulta Q está construído por meio do processo descrito até agora e pode ser dado como entrada para o modelo M com o intuito de obter a sua estimativa de cardinalidade.

Até o presente momento, somente uma parte da modelagem do problema foi apresentado, aquela que envolve extrair as informações das consultas para formar o conjunto de atributos (*features*), o vetor numérico obtido por meio do mecanismo descrito acima, que é repassado ao modelo de aprendizagem de máquina. No entanto, observe que o problema de estimar as cardinalidades foi contextualizado usando a abordagem de aprendizagem supervisionada. Ou seja, ainda falta considerar o rótulo que é utilizado.

Uma vez que o objetivo de aprendizagem do modelo M é predizer as cardinalidades das consultas, então a cardinalidade deve ser utilizado como o rótulo associado a cada vetor

representante das consultas. Por já ser um valor numérico, a cardinalidade, teoricamente, não iria requerer nenhuma transformação. Assim como o valor constante *v* que ocorre nos predicados de seleção, a cardinalidade passa por uma transformação tendo em vista evitar que problemas ocorram no momento de treinar o modelo preditivo M.

Para ser usado como rótulo, a cardinalidade passa por duas transformações. A primeira é a transformação logarítmica que é, basicamente, a aplicação da função *log* na cardinalidade *c*, como apresentado na Equação abaixo:

$$c\_log = log(c) \tag{4.2}$$

Essa transformação é necessária devido à diferença de cardinalidade que pode ocorrer entre duas consultas Q_1 e Q_2. Por exemplo, suponha que, em uma mesma base de dados, Q_1 possui um predicado de seleção muito seletivo, ou seja, $c(Q_1) = 10$, e que Q_2 é uma consulta com predicado de seleção pouco seletivo e retorna muitas tuplas, isto é, $c(Q_2) = 10000$. Se considerarmos como rótulo a cardinalidade sem nenhuma transformação, a amplitude da diferença entre as cardinalidades de Q_1 e Q_2 é de 9990, um valor muito alto que pode causar um enviesamento na distribuição dos dados de cardinalidade, e com isso, impactar o treinamento do modelo de predição, pois este poderá reconhecer o viés, dando uma maior importância para a consulta Q_2 no seu treinamento, por exemplo. Já considerando a transformação logarítmica, a magnitude da diferença é bem menor (veja a Equação a seguir), evitando o problema citado acima.

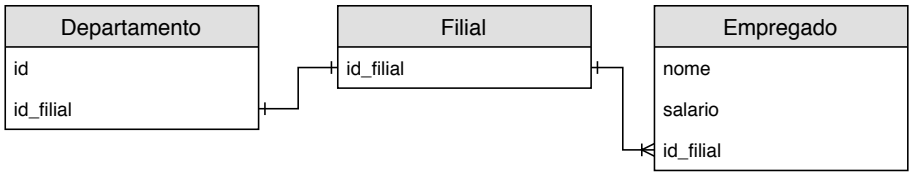
$$log(c(Q_2)) - log(c(Q_1)) = log(10000) - log(10) = 4 - 1 = 3 \tag{4.3}$$

A segunda transformação é a normalização min-max dos valores *c_log*, representada na Equação 4.4, a mesma realizada para os valores constantes que aparecem nos predicados de seleção. Esta transformação visa assegurar que os rótulos terão valores no intervalo $[0, 1]$, garantindo que não o processo de aprendizagem do modelo M ocorra livre de problemas.

$$rótulo = c\_log\_normalizado = \frac{c\_log - c\_log\_min}{c\_log\_max - c\_log\_min} \tag{4.4}$$

Com essas transformações feitas, o rótulo de cada vetor representativo de cada consulta é obtido. Em termos de aprendizagem de máquina, vale observar que o modelo M vai ter a tarefa de predizer um rótulo a partir das características presentes em um vetor numérico, independentemente da origem desses dados, isto é, o modelo vai realizar sua predição baseado nas informações que foram passadas a ele durante a etapa de treinamento. Ou seja, o modelo M

Figura 32 – Exemplo de um esquema de um BD de uma empresa.



vai predizer valores de rótulos que não irão representar diretamente a cardinalidade das consultas representadas pelos vetores fornecidos a ele. Consequentemente, há a necessidade de aplicar as operações matemáticas inversas às transformações que foram realizadas nas cardinalidades para gerar os rótulos. A Equação seguinte realiza essa conversão de rótulo para a cardinalidade predita.

$$\hat{c} = 10^{c\_log\_desnormalizado} = 10^{c\_log\_min+(c\_log\_max-c\_log\_min)\cdot rótulo} \tag{4.5}$$

Fundamentalmente, a normalização min-max é desfeita ao somar o valor *c_log_min* ao resultado da multiplicação do *rótulo* com a amplitude dos valores *c_log* (*c_log_max* − *c_log_min*), obtendo assim o valor *c_log_desnormalizado*. Para desfazer a transformação logarítmica, aplica-se a função exponencial no valor *c_log_desnormalizado* e, assim, a predição de cardinalidade *c* é recuperada.

Enfim, com todo processo de modelagem descrito acima, agora é possível criar um conjunto de dados de treinamento *X* rotulado com o objetivo de iniciar o treinamento do modelo preditivo *M*, o qual é abordado mais adiante no texto. Mas antes, é apresentado uma ilustração do processo descrito por meio do uso de um exemplo.

4.3.1 Exemplo Ilustrativo

Para exemplificar o processo de modelagem de consultas SQL, considere a Figura 33. Nela, é apresentada uma consulta SQL que envolve as relações Empregado e Filial de um BD de uma empresa. Ou seja, esse BD pretende guardar as informações relativas às entidades de Departamento, Empregado e Filial, como mostrado no esquema da Figura 32. Basicamente, a consulta SQL mostrada retorna o nome dos empregados e o id da sua respectiva filial na qual está inserido, sendo que somente interessa aqueles empregados cujo salário é maior que 5000.

Inicialmente, o SGBD vai gerar vários PEs alternativos e equivalentes à consulta declarada. Na mesma Figura 33, também é mostrada um desses possíveis PEs. Para transformar esse PE em um vetor de características que pode ser usado no contexto de aprendizagem de máquina, inicia-se modelando as relações.

Como discutido mais acima, as relações são representadas por vetores da técnica *one-hot enconding*. Como há três relações, então os vetores representativos das relações do BD são, respectivamente: $[1,0,0]$, $[0,1,0]$ e $[0,0,1]$. Tendo em vista que somente as duas últimas relações estão envolvidas na consulta, então temos dois vetores a serem considerados: $[0,1,0]$ e $[0,0,1]$. Usando a variante proposta, então se aplica uma operação OU nos *bits* desses dois vetores, obtendo logo em seguida o vetor $[0,1,1]$. Portanto, essa vai ser a representação das relações envolvidas na consulta. Ou seja, o vetor x final que representará a consulta para a entrada do modelo M é iniciado com esse vetor, como pode ser visualizado na Figura 33, na parte em amarelo do vetor indicado.

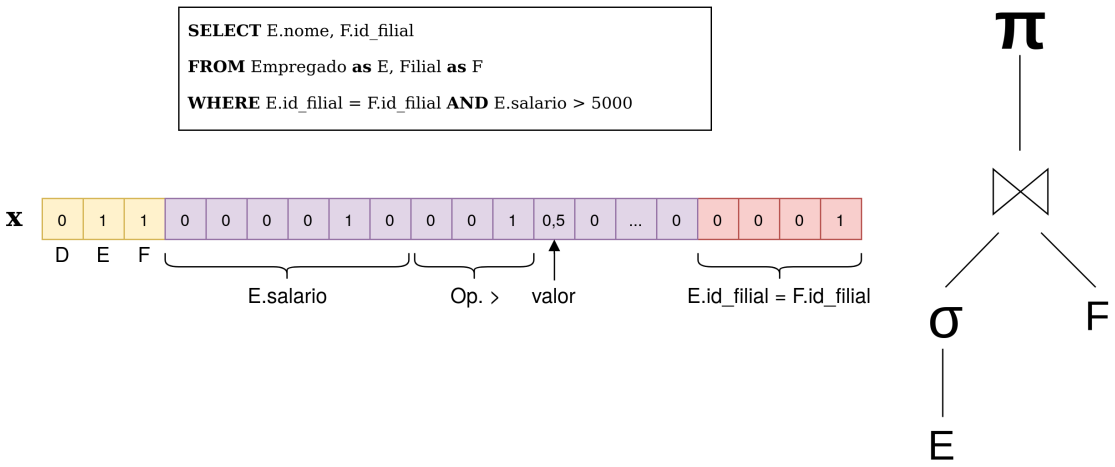
Depois que as relações estão modeladas, temos a modelagem dos predicados de seleção que é constituída por três partes: a representação do atributo, do operador e do valor. Devido à existência de 6 atributos no total, então cada atributo é modelado para um vetor de 6 *bits* com um deles sendo 1, como dito anteriormente. Considerando que o atributo $E.salario$ é o quinto na lista ordenada de atributos, então sua representação é o vetor $[0,0,0,0,1,0]$. Adiante, temos a representação do operador matemático envolvido. Como temos três operadores básicos, então $<,=,>$ são representados por $[1,0,0]$, $[0,1,0]$ e $[0,0,1]$, respectivamente. $\leq$ e $\geq$ podem ser representados ao fazer uma operação OU entre os vetores dos operadores $<,>$ com o do operador $=$. Como o operador que aparece na consulta é o $>$, então o vetor para representá-lo é $[0,0,1]$. Por fim, temos que lidar com o valor da constante, 5000. Esse valor é normalizado de acordo com o processo descrito anteriormente, usando os valores mínimo e máximo que dependem do contexto da base de dados. Como estamos em um exemplo simples, admita que o menor e o maior valor do atributo salário é, respectivamente, $1K$ e $9K$. Desta forma, o valor a ser usado na modelagem é obtido com a seguinte equação:

$$v\_normalizado = \frac{5000 - 1000}{9000 - 1000} = \frac{4}{8} = \frac{1}{2} = 0.5 \tag{4.6}$$

A parte roxa da Figura 33 é composta pela concatenação desses três vetores para formalizar a modelagem do predicado de seleção que aparece na consulta SQL. Devido à consulta usada como exemplo possuir somente um predicado de seleção, então o restante das posições do vetor dos predicados de seleção recebe o valor zero. Essas posições são necessárias tendo em vista a possibilidade de uma consulta SQL qualquer apresentar mais de um predicado de seleção.

Na parte final do vetor, destacada em vermelho, temos a representação da condição de junção da consulta. Como pode ser observado na Figura 32, existem pelo menos dois relacionamentos envolvendo chaves primárias e estrangeiras, além da possibilidade de não

Figura 33 – Exemplo da modelagem de uma consulta SQL usando o método proposto.



aparecer nenhuma junção na consulta. Portanto, quatro *bits* são usados para representar cada possibilidade, sendo que a condição de junção entre as relações E e F foi associada à última posição do vetor da representação *one-hot enconding*. Assim, o vetor $[0,0,0,1]$ compõe a parte final do vetor x representativo totalmente da consulta apresentado na Figura 33. Este vetor já está pronto para ser repassado ao modelo M de aprendizagem de máquina.

4.4 Treinamento

Para que o modelo do *LightGBM* se torne útil e comece a ser capaz de gerar estimativas de cardinalidades com boa acurácia, necessita-se, então, de treiná-lo antes com os denominados dados de treinamento. Duas abordagens são possíveis para essa tarefa. A primeira é gerar uma carga de trabalho sintética baseada na base de dados subjacente para que, dessa forma, o modelo seja treinado com esses dados e consiga capturar algumas características dos dados relevantes para a estimação das cardinalidades. Uma outra possibilidade é usar o estimador padrão para executar um conjunto de consultas da carga de trabalho real. Depois, utilizar essas consultas armazenadas no módulo Acumulador para formar o conjunto de treinamento e, assim, treinar o modelo M. Essa segunda abordagem evita o *overhead* no otimizador de consultas que é necessário para que se obtenha um conjunto de treinamento sintético e é por esse motivo que as técnicas tradicionais fazem sua opção por ela. Em compensação, a primeira abordagem permite que o modelo M seja utilizado logo de início. Na nossa abordagem, optamos pela primeira opção, visto que um modelo sem nenhuma fase de treinamento executada anteriormente realiza predições com muitos erros.

Para gerar o conjunto de dados de treinamento sintético, usamos o Algoritmo 2, apresentado no Capítulo 3, que foi proposto em (KIPF *et al.*, 2019). A única adaptação que fizemos no algoritmo é que o número de junções que pode ocorrer em uma consulta no conjunto de treinamento seja escolhido entre 0 e 4. Na geração desse conjunto, as informações sobre as relações de um BD precisam ser fornecidas como entrada para o algoritmo acima, além da quantidade de consultas que o conjunto deverá ter. Relembrando o funcionamento desse algoritmo, ele gera consultas através de escolhas aleatórias para as relações, os predicados de seleção e as condições de junção. Posteriormente, executa essas consultas e guarda a cardinalidade a elas associada.

4.5 Evolução

À medida em que as consultas da carga de trabalho real são executadas, novas cardinalidades são obtidas e estas são armazenadas no módulo Acumulador, como demonstrado na Figura 30. Além disso, os tipos de consultas presentes na carga de trabalho podem mudar, assim como a base de dados pode sofrer modificações por meio da execução das consultas que modificam os dados, por exemplo as consultas de inserção. Visando agregar essas novas informações em um modelo que já está treinado, três abordagens são possíveis para o método proposto.

A primeira maneira de realizar a evolução do modelo é refazer totalmente a sua fase de treinamento. Isto é, descarta-se o modelo treinado M atual e, então, um novo é treinado sem levar em consideração a estrutura do modelo M. Por um lado, essa abordagem é a mais completa devido ao fato de realizar um ajuste do modelo de aprendizagem de máquina usando todos os dados de treinamento disponíveis naquele momento, como os novos valores mínimo e máximo dos atributos usados no processo de normalização. Por outro lado, retreinar um modelo é uma atividade custosa e dispendiosa, o que pode causar um impacto no desempenho dos SGBDs, além de não ser possível realizar essa atividade em um curto período de tempo entre as atualizações, o que pode levar o método proposto a sofrer com a primeira fonte de erro que é utilizar dados desatualizados para gerar as estimativas de cardinalidade.

Uma segunda abordagem é fazer o modelo M atual evoluir para acomodar as novas informações sem modificá-lo estruturalmente ao invés de descartá-lo. Isto é, o modelo M gerado pelo *LightGBM* é composto por um conjunto de estimadores bases que são responsáveis por gerar as predições de cardinalidade. Nesta abordagem, os novos dados de treinamento que estão no

módulo Acumulador são agregados ao modelo por meio do ajuste dos pontos de quebra, *splits*, dos estimadores, sem adicionar nenhum novo estimador base. Essa abordagem possibilita que novas informações sejam fornecidas ao modelo ao mesmo tempo em que aproveita o conhecimento adquirido anteriormente pelo modelo M. Porém, pode ocorrer que os estimadores bases atuais não consigam generalizar o novo comportamento presente nos novos dados, consequentemente gerando um impacto na qualidade das estimativas de cardinalidade.

Por último, a terceira abordagem é realizar mudanças estruturais no modelo M para que este acomode os possíveis novos padrões presentes nos novos dados. Neste caso, novos estimadores bases são treinados e adicionados ao modelo M treinado pelo *LightGBM*. Esta abordagem supre as dificuldades das anteriores, mas ainda apresenta a dificuldade de lidar com possíveis mudanças no esquema do BD, nos valores mínimos e máximos usados para normalização, por exemplo.

Finalmente, é importante ressaltar que o nosso método proposto realiza a fase de evolução de forma independente do processamento e execução das consultas. Isto é, deve ser implementado uma *thread* independente que é responsável por realizar essa atualização do modelo M. Um processamento em *batch* é realizado para essa atividade, ou seja, o momento de atualizar o modelo é quando uma quantidade n de consultas foram executadas e armazenadas no módulo Acumulador, mas que ainda não foram incorporadas no treinamento do modelo.

4.6 Incerteza

Geralmente, um modelo preditivo M é treinado com o objetivo de realizar predições de um único ponto (valor), o qual segue a tendência do valor esperado (média) do rótulo que está sendo predito, no nosso caso a cardinalidade. Essa estratégia é conhecida como estimativa por ponto, em contraste com a abordagem denominada de estimava por intervalo que tem o intuito de calcular um intervalo de valores no qual o valor real que está sendo predito deve estar presente. Até o momento, a nossa estratégia proposta está baseada na estimativa por ponto. Ou seja, o modelo M que é treinado pelo método proposto prediz o valor esperado da cardinalidade de uma consulta. Entretanto, como esse valor é obtido através de um processo de predição, incorre que há então um risco desse valor não estar representando bem o valor real. Por exemplo, os novos dados que são usados para que o modelo M realize as predições não são necessariamente os mesmos que foram utilizados no seu conjunto de treinamento. Ou seja, existe uma incerteza em volta do valor predito que faz levantar o questionamento de quão confiáveis são as predições

para esses novos dados. No momento em que o modelo M segue a abordagem de estimativa por ponto, fica impossível responder tal questionamento, a menos que seja assegurado que os novos dados são realmente semelhantes ou iguais aos do conjunto de treinamento, dessa forma é possível garantir que as predições são confiáveis utilizando essa estratégia. No cenário de otimização de consultas, não é possível assumir esse fato, visto que o atual estado do BD pode mudar ou a carga de trabalho atual pode vir a ser significativamente diferente daquela utilizada para a fase de treinamento. Ademais, confiar totalmente nas predições do modelo M pode levar a uma perda de desempenho caso as predições de cardinalidade não estejam representativas para a cardinalidade real das consultas.

Propomos, então, uma maneira de quantificar a incerteza das estimativas por meio do uso do conceito de intervalo de predição, o qual define um intervalo de valores nos quais o valor real se encontra de acordo com uma probabilidade conhecida. Por conseguinte, é possível afirmar o quão confiável é uma predição do modelo M. A estratégia proposta utiliza ao invés de um único modelo M, três modelos, $M_{inferior}$, $M_{média}$ e $M_{superior}$, com diferentes objetivos de predição. O modelo $M_{média}$ é o mesmo modelo que foi treinado para dar uma estimativa por ponto. Os outros dois novos são utilizados para realizar uma regressão quantílica. Essa técnica permite que os quantis (ou percentis) de um valor sejam preditos. Nos nossos experimentos, realizamos uma mudança na função de custo para que o seu objetivo de predição seja alterado para a regressão quantílica.

Para definir qual confiança aceitar, dois novos parâmetros são incrementados no SGBD para definir o quantil inferior e o superior, respectivamente. Assim, o modelo $M_{inferior}$ ficará encarregado de realizar predições para o quantil inferior e o $M_{superior}$ para o quantil superior. Com essas informações, é possível afirmar que o valor predito pelo modelo $M_{média}$ está de acordo com o nível de confiança definido pelo DBA. O nível de confiança é a diferença entre o valor do quantil superior e o inferior. Por exemplo, se o nível desejado for de 90%, então os quantis devem ser definidos para 95º e 5º, respectivamente.

Por fim, dada uma nova consulta Q, o SGBD calculará as estimativas de cardinalidade de Q por meio do uso do modelo $M_{média}$ e verificará se elas estão de acordo com a confiança definida pelo DBA através dos modelos $M_{inferior}$ e $M_{superior}$. Caso não estejam, utiliza-se o método padrão do SGBD para calcular as estimativas, por exemplo os histogramas.

4.7 Conclusão

Este capítulo descreveu os aspectos da nossa abordagem para o cálculo de estimativas de cardinalidade das consultas SQL no âmbito dos SGBDs relacionais. Inicialmente, foram apresentados os detalhes envolvendo a proposta de uma arquitetura que um SGBD deve ter para que seja possível a utilização de modelos de aprendizagem de máquina para gerar as estimativas de cardinalidade. Logo em seguida, o processo de modelagem das consultas SQL para um formato que possibilite o uso no contexto de aprendizagem de máquina foi apresentado juntamente com um exemplo ilustrativo. Depois, a forma de treinar e evoluir o modelo foram descritos. Por fim, uma possibilidade de lidar com a incerteza das estimativas foi descrito.

5 AVALIAÇÃO EXPERIMENTAL

Neste capítulo, são apresentados os experimentos realizados com a intenção de validar de maneira experimental a Hipótese de Pesquisa 1, além de mostrar o comportamento do método proposto quando implementado em um SGBD.

Inicialmente, a Seção 5.1 apresenta uma breve introdução do SGBD PostgreSQL. Logo a seguir, na Seção 5.2, são apresentadas algumas ideias relevantes sobre a implementação do método proposto, assim como as modificações feitas no PostgreSQL para fazer a integração desse método. Adiante, a Seção 5.3 detalha as características do ambiente de execução que foi utilizado para executar os experimentos. Por fim, a Seção 5.4 apresenta os resultados experimentais do método proposto sob diversos pontos de vistas.

5.1 PostgreSQL

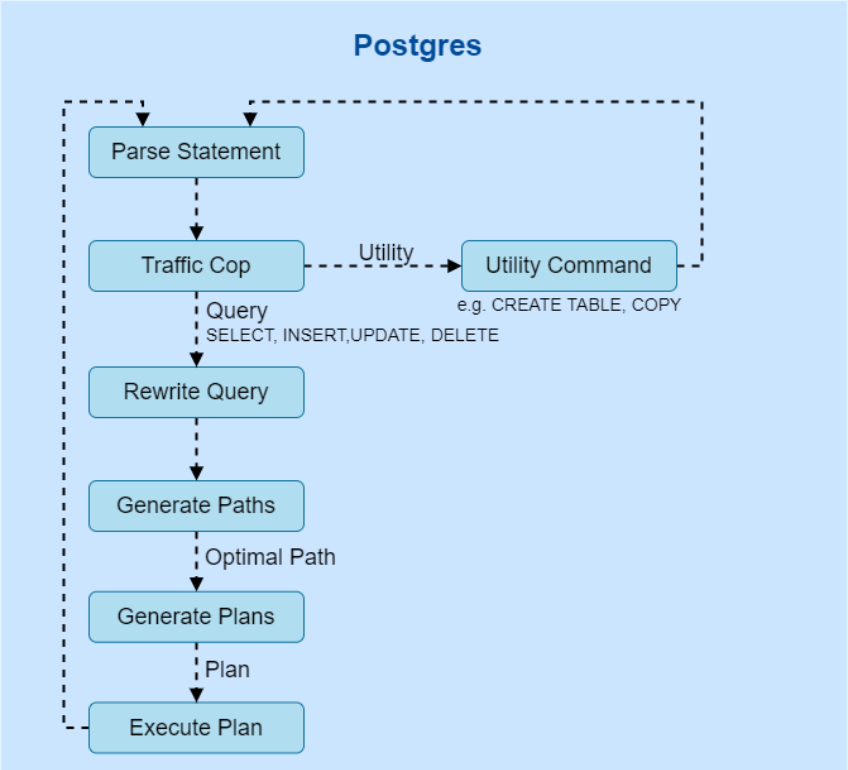
Com o intuito de avaliar o comportamento da nossa estratégia em um SGBD, optou-se por usar o SGBD PostgreSQL tendo em vista sua ampla aceitação quer seja pela comunidade acadêmica, quer seja pela indústria, pois este tem um longo tempo de desenvolvimento e apresenta um padrão de funcionamento.

Brevemente, o PostgreSQL é um SGBD que segue a arquitetura cliente-servidor. Na Figura 34, apresenta-se a maneira pela qual esse sistema trata suas consultas SQL. Nesta imagem, é importante ressaltar duas partes: *Generate Paths* e *Generate Plan*, pois são essas duas que serão usadas no processo de integração do método proposto. O PostgreSQL trabalha com o conceito de *paths* que são PEs mais simples por ainda não poderem ser executados diretamente. O primeiro módulo é responsável por gerar um conjunto de possíveis *paths* que geram o mesmo resultado da consulta. Depois de avaliar o custo de cada um, o *path* com menor custo será expandido em um PE para ser executado e gerar o resultado da consulta.

5.2 Implementação

Em relação aos aspectos de implementação, evidenciou-se que não há necessidade de se implementar o modelo de aprendizagem dentro do código fonte do PostgreSQL. Na prática, para deixar essas duas partes desacopladas, fizemos uso da capacidade de interoperabilidade da linguagem de programação Python versão 3, na qual está implementado o *LightGBM*, com a linguagem C ANSI, na qual o PostgreSQL está implementado, e, então, desenvolvemos dois

Figura 34 – Módulos do SGBD PostgreSQL.



Fonte: Adaptado de (POSTGRESQL, 2012).

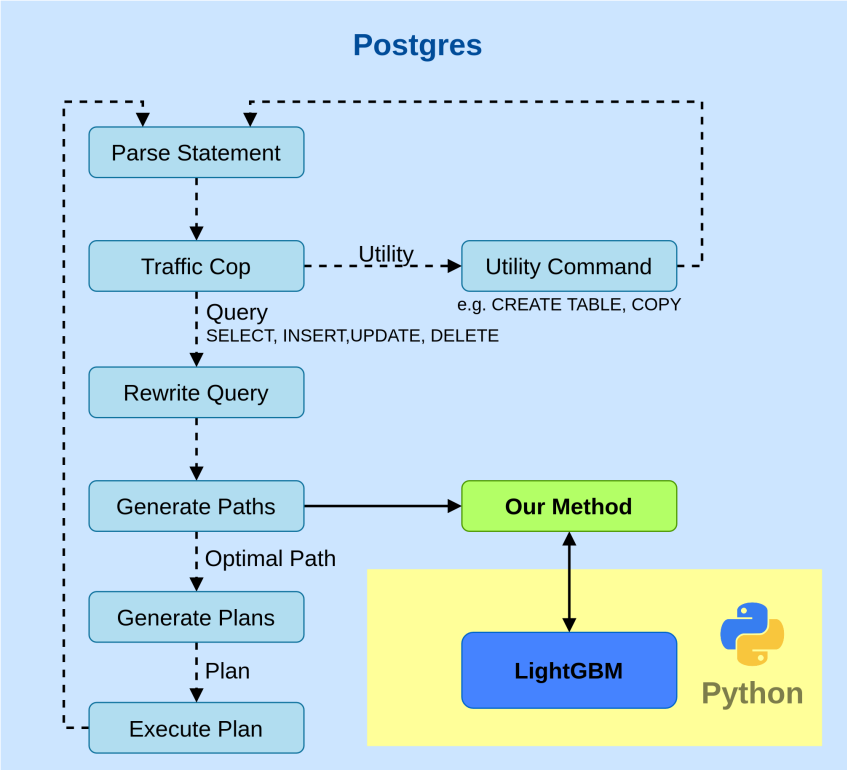
módulos que se comunicam, um em cada linguagem. Muito embora se evite realizar modificações profundas na arquitetura do SGBD, a nossa abordagem ainda precisa modificar o código fonte interno do PostgreSQL.

A primeira modificação que deve ser realizada é a implementação da capacidade de injeção de cardinalidade nos nós dos PEs, no contexto do PostgreSQL, os chamados *paths*. Essa modificação foi facilitada devido à maneira pela qual o PostgreSQL desenvolveu o módulo de otimização de consulta, como apresentado anteriormente.

Posteriormente, necessita-se que o SGBD seja capaz de monitorar as cardinalidades reais das consultas já executadas com o intuito de fornecer um retorno para o modelo de aprendizagem. Para isso, uma relação transparente ao usuário foi implementada de modo a guardar tanto a consulta SQL textualmente quanto a sua cardinalidade real que foi obtida no momento de sua execução.

Por fim, há a etapa de integrar essas implementações em um módulo funcional. Como pode ser visualizado na Figura 35, um novo módulo foi criado no ambiente do PostgreSQL, sendo que este é o responsável por lidar com a comunicação entre as linguagens de programação Python e C ANSI. Ademais, nesse módulo se encontra os mecanismos necessários para capturar os PEs, enviar ao *LightGBM* para que as predições de cardinalidade sejam calculadas e, por fim,

Figura 35 – Módulos do SGBD PostgreSQL modificados para a adição do método proposto.



injetadas nos planos.

5.3 Ambiente de Execução

Uma máquina com um processador Intel Xeon E5-2609 v3 1.9GHz, 15M Cache, 6 núcleos, com a versão do kernel GNU/Linux 4.15.0 e com o sistema operacional *Ubuntu* foi utilizada para realizar os experimentos apresentados na próxima seção. Além disso, a versão do PostgreSQL escolhida foi a 12 estável, pois esta é a versão mais recente do sistema. Com o intuito de evitar erros de sincronização, o paralelismo do PostgreSQL foi desativado.

Já em relação ao lado da linguagem Python, as bibliotecas *Scipy* (JONES *et al.*, 2001), *scikit-learn* (PEDREGOSA *et al.*, 2011) e *lightgbm* (KE *et al.*, 2017) foram utilizadas para auxiliar na implementação do método proposto.

5.4 Resultados Experimentais

Para realizar os experimentos cujos resultados são apresentados a seguir, optamos por utilizar a base de dados conhecida como IMDB. Esta base armazena informações sobre vídeos, atores, diretores e seus relacionamentos e é composta por 21 relações no modelo relacional. De

acordo com (LEIS *et al.*, 2017), em virtude das características presentes nessa base de dados, tais como atributos correlacionados, distribuições não uniformes, entre outras, a geração das estimativas de cardinalidade é mais difícil de ser realizada pelos SGBDs. O tamanho dessa base utilizada é de aproximadamente 9 GB. Em relação à carga de trabalho, 100 mil consultas foram geradas utilizando o algoritmo 2, sendo que 90 mil foram usadas para realizar a fase de treinamento. Já para as fases de validação e teste, respectivamente, foram utilizadas 5000 para cada uma. Por fim, mas não menos importante, os resultados foram obtidos através da execução do mesmo cenário cinco vezes, sendo que os valores das métricas representam a média dos valores obtidos em cada execução.

5.4.1 Escolha dos Hiper-parâmetros

Os hiper-parâmetros de um modelo dizem respeito às variáveis que o definem. Por exemplo, em virtude do *LightGBM* empregar as árvores de decisão em seu algoritmo, um possível hiper-parâmetro é a profundidade máxima dessas árvores, isto é, qual altura máxima que uma árvore pode ter. Em síntese, a lógica por trás dos algoritmos de aprendizagem pode ser modificada de acordo com a escolha dos valores dos hiper-parâmetros. Para encontrar quais são os valores que geram a melhor configuração do modelo para um dado problema, utiliza-se a técnica de busca no *grid* (mais conhecida como *Grid Search*). Nessa técnica, os possíveis valores a serem testados para cada um dos hiper-parâmetros são definidos e, a partir disso, são geradas todas as combinações possíveis entre as possibilidades de todos os hiper-parâmetros. Para cada combinação de valores, um modelo é gerado e é testado através da técnica de validação cruzada. Esta técnica divide o conjunto de treinamento em k partes iguais de maneira aleatória e estas partes serão usadas para realizar tanto a fase de treinamento quanto a fase de testes.

O modelo *LightGBM* possui um grande conjunto de hiper-parâmetros. Por conta disso, encontrar a melhorar configuração é uma tarefa difícil e demorada, visto que o espaço de combinações possíveis dos hiper-parâmetros é grande. Com isso, um subconjunto de quatro hiper-parâmetros foi escolhido, sendo eles: *max_depth*, *num_leaves*, *n_estimators* e *learning_rate*. Conforme descrito acima, o *max_depth* define a profundidade das árvores de decisão. Já *num_leaves* define o número de folhas presentes em cada árvore, dessa forma definindo o nível de complexidade das árvores. O *n_estimators* estabelece a quantidade de árvores de decisão que serão treinadas. O *learning_rate* determina a velocidade em que o modelo incorpora as novas informações vistas durante a cada etapa de treinamento. Enfim, os valores testados para cada um

estão mostrados na Tabela 5.

Tabela 5 – Hiper-parâmetros e seus valores testados no *Grid Search*.

Hiper-parâmetro	**Valores testados**
max_depth	8, 16
num_leaves	32, 64, 128
n_estimators	1000, 5000, 10000
learning_rate	0.01, 0.1

Depois de realizar todas as combinações desses valores, a configuração que apresentou o menor erro, usando a métrica *Q-Error* que será definida a seguir, de treinamento foi $16, 64, 10000, 0.01$, respectivamente. Essa configuração foi usada em todos os modelos.

5.4.2 Acurácia

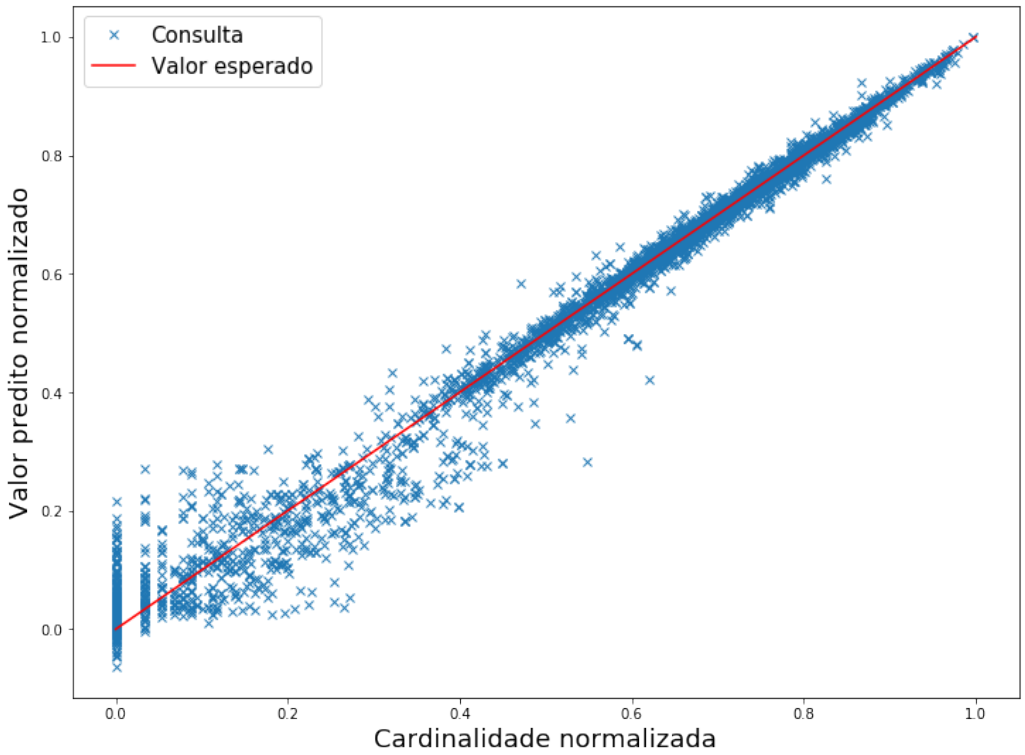
Para avaliar a qualidade das estimativas geradas pelo modelo obtido por meio do *LightGBM*, utilizamos a métrica conhecida como *Q-error*, a qual representa o fator da diferença entre o valor real c e o valor predito $\hat{c}$. Formalmente, a Equação dessa métrica é apresentada abaixo. É necessário comentar que quanto mais próximo de 1 é o valor dessa métrica, melhores são as estimativas, pois no cenário ideal em que $c = \hat{c}$ o valor da métrica seria 1.

$$Q - error(c, \hat{c}) = \frac{max(c, \hat{c})}{min(c, \hat{c})} \tag{5.1}$$

Visando demonstrar se o modelo treinado M está realmente aprendendo a tarefa de regressão submetida a ele, apresentamos o primeiro resultado na Figura 36. Neste gráfico, o eixo das abscissas representa os valores normalizados da cardinalidade das consultas, enquanto que o outro eixo representa os valores preditos pelo modelo. Desta forma, a reta vermelha apresentada mostra como seria se o resultado fosse perfeito, isto é, os valores preditos iguais aos reais. O primeiro fato interessante observável por meio desse gráfico é que o modelo produz predições que seguem a tendência do valor esperado. Mais especificamente, o modelo M prediz com uma melhor exatidão para os maiores valores de cardinalidade, pois observe que entre 0.0 e 0.4 há uma maior variação das consultas em relação à reta do valor esperado. Conclui-se, então, que o modelo M é capaz de aprender as características da base de dados e, com essa informação, predizer os valores de cardinalidade normalizados.

Para finalizar a análise desse gráfico, é importante explicar o porquê de haver várias consultas no ponto 0.0 do eixo horizontal, ou seja, a cardinalidade normalizada de algumas consultas estão sendo mapeadas para o valor 0.0. Isto quer dizer que a cardinalidade dessas

Figura 36 – Predições realizadas pelo modelo *M* treinado. O gráfico também mostra a tendência do valor esperado.

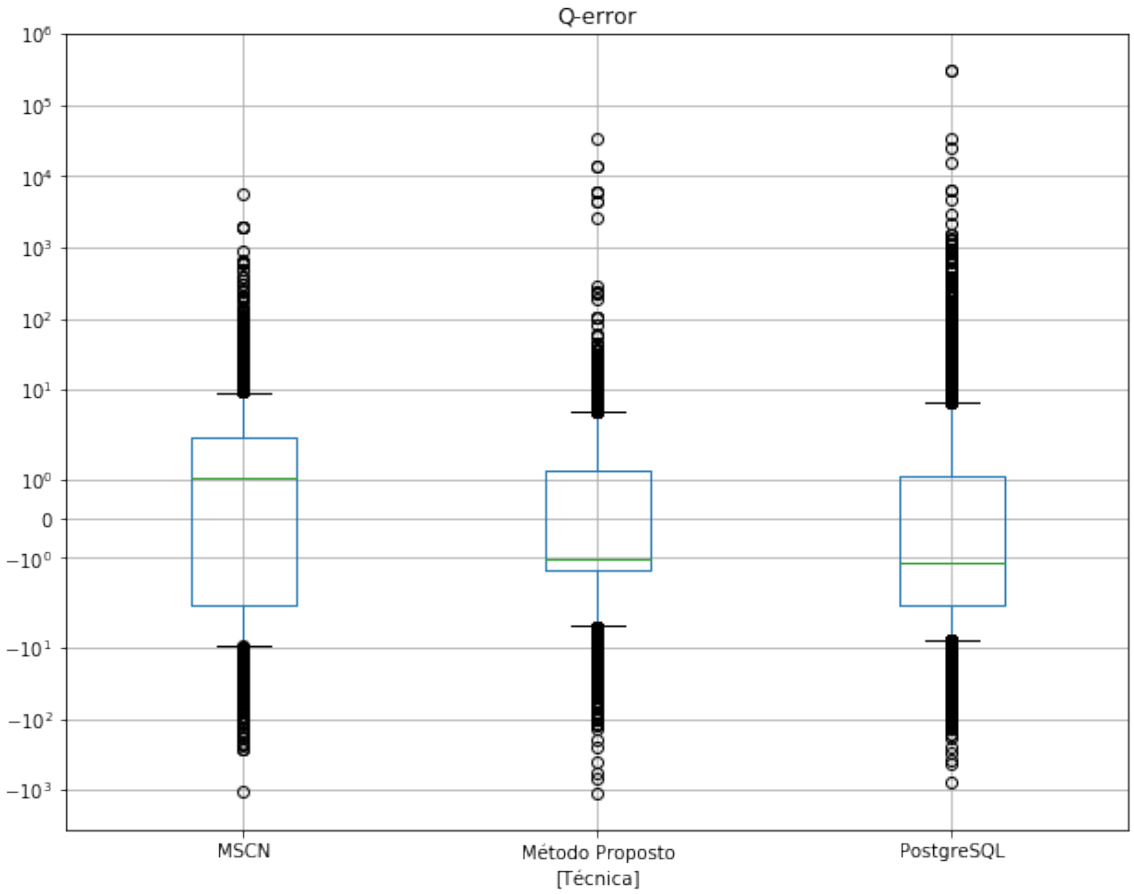


consultas em escala é a menor, portanto seu valor será mapeado para zero quando for realizado a normalização min-max.

Agora partimos para uma análise da qualidade das estimativas de cardinalidade considerando que o modelo prevê bons valores para a cardinalidade normalizada. Para isso, vamos comparar o resultado do método proposto com duas outras técnicas: PostgreSQL e MSCN sem a variante que leva em consideração o processo de amostragem das relações. O primeiro devido ao fato de representar bem a classe das técnicas baseadas em histogramas e esse segundo em virtude dele representar o estado da arte na comunidade científica como sendo uma das propostas mais recentes do uso de técnicas de aprendizagem de máquina para o problema das estimativas de cardinalidade nos SGBDs. A Figura 37 apresenta um gráfico *boxplot* da métrica *Q-Error* para cada uma das referidas técnicas.

No gráfico, os eixos das abscissas e das coordenadas representam, respectivamente, cada uma das técnicas e a métrica *Q-Error* em escala logarítmica. Ademais, vale esclarecer que para gerar essa Figura, dividimos os valores dessa métrica em dois cenários: subestimação e sobre-estimação. No primeiro caso, as estimativas do modelo *M* são valores menores que os reais, ao passo que as estimativas que ocorrem no segundo caso são maiores que os valores reais. Para diferenciar cada cenário, multiplicamos por -1 os valores quando ocorre o caso de subestimação.

Figura 37 – *Boxplot* da métrica *Q-error* de cada uma das técnicas.



Desta forma, os valores negativos representam esse caso. Já os positivos representam o caso de sobre-estimação. A tendência de subestimar ou sobre-estimar as estimativas de cada um dos trabalhos é mostrada por meio da reta verde que divide o intervalo inter-quartil, o qual está representado pela caixa azul.

Como pode ser visto no gráfico mostrado, as estimativas de cardinalidade geradas pelas técnicas baseadas em aprendizagem de máquina, MSCN e o método proposto, são melhores que as geradas pelo PostgreSQL, pois os pontos fora da curva estão mais compactas em um intervalo menor. Por um outro ponto de vista, é possível ver também que o intervalo inter-quartil do método proposto é menor que o das duas técnicas, o que implica que os valores da métrica *Q-Error* estão mais concentradas entre 1 e −1. Portanto, já concluímos que as técnicas baseadas em aprendizagem de máquina produziram estimativas com uma maior acurácia, diminuindo o erro de predição. Agora comparando as duas técnicas de aprendizagem entre si, iniciamos observando que o MSCN apresentou pontos fora da curva em um menor intervalo de valores, considerando o caso de sobre-estimação, a parte superior do gráfico. Isto significa que o método proposto produz maiores valores de *Q-Error*, o que implica que algumas estimativas podem apresentar

uma maior imprecisão no nosso método. Nesse cenário, a nossa proposta está sobre-estimando as estimativas com um fator maior do que os do MSCN. Em contrapartida, o MSCN teve um ponto fora da curva com um erro de subestimação maior. Além disso, o intervalo inter-quantil do método proposto está mais concentrado que o do MSCN. Consequentemente, concluímos que os valores entre os quartis $Q1$ e $Q3$ da nossa técnica possuem uma melhor qualidade que o do MSCN. Por fim, os estimadores do método proposto e do PostgreSQL apresentaram uma tendência de subestimação, visto que a barra verde dentro do intervalo inter-quartil está mais próxima do valor -1, o qual significa que as estimativas de cardinalidade são menores que as reais. Já o MSCN apresentou uma tendência próxima do valor 1, o que significa que sua tendência é sobre-estimar os valores de cardinalidade.

Para chegar a uma conclusão final, observe os valores apresentados na Tabela 6 que apresenta várias medidas estatísticas sobre a métrica de *Q-Error* das três técnicas. Os valores destacados são os menores e, portanto, representam o melhor valor alcançado para cada uma das estatísticas. Os valores apresentados pelo método proposto são melhores, exceto a média e o valor máximo da métrica *Q-Error* no qual o MSCN apresenta resultados melhores. É importante ressaltar a grande diferença existente entre os valores máximos obtidos pelas técnicas de aprendizagem e pelo PostgreSQL. Em conclusão, o método proposto por esta dissertação gera estimativas de cardinalidade que possuem uma maior acurácia que as técnicas tradicionais baseadas em histogramas, assim como também apresenta resultados melhores ou semelhantes com os das técnicas do estado da arte.

Tabela 6 – Resultados do *Q-error* das estimativas de cardinalidade geradas por cada uma das técnicas.

Técnica	**Média**	**Mediana**	**Percentil**			**Máximo**
			90	**95**	**99**	
MSCN	**7,659**	2,000	9,829	20,618	77,337	**6.586,000**
Método Proposto	21,920	**1,243**	**5,000**	**12,245**	**50,690**	32.803,500
PostgreSQL	171,120	1,775	20,352	80,000	964,000	314.187,000

5.4.3 Tempo de Execução

Para dar suporte à comprovação da Hipótese de Pesquisa 1, executamos 5000 consultas, usando tanto a técnica padrão do PostgreSQL quanto a versão modificada desse sistema com o método proposto, e acumulamos o tempo de execução de todas elas em quatro partes: até 1250, 2500, 3750 e 5000 consultas. Os resultados obtidos por meio desse experimento são mostrados

Tabela 7 – Tempo acumulado de execução de 1250, 2500, 3750 e 5000 consultas quando executadas no PostgreSQL não modificado e esse sistema modificado com a nossa proposta.

Abordagem / Tempo acumulado (min)	**≤ 1250**	**≤ 2500**	**≤ 3750**	**≤ 5000**
Método Proposto	277.09389	598.76837	886.04205	1182.96412
PostgreSQL	285.14849	612.49828	906.34336	1210.44504
Diferença	**8.05460**	**1.3,72991**	**20,30131**	**27,48092**

na Tabela 7. A última linha indica a diferença entre o resultado obtido por meio do PostgreSQL sem modificação e com modificação.

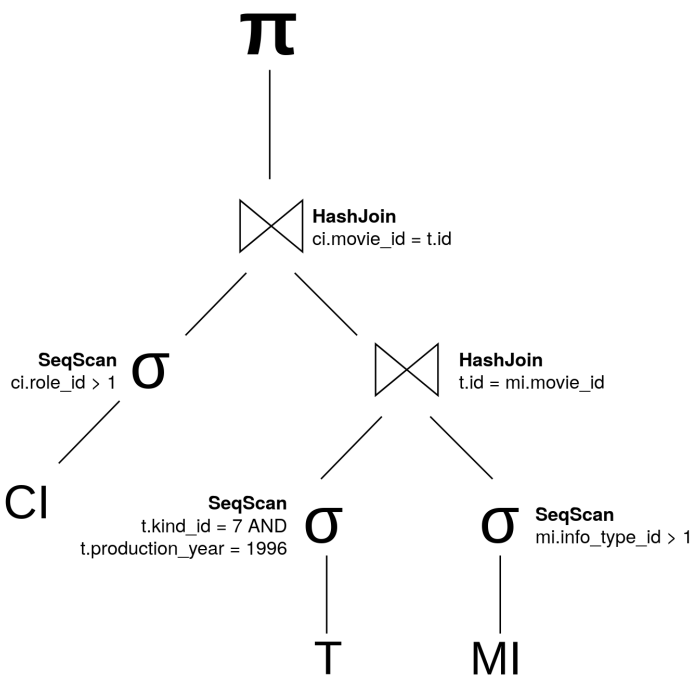
Observando os valores apresentados, chegamos à conclusão de que o tempo de executar as mesmas consultas no PostgreSQL usando o nosso método proposto e o mesmo sistema usando a técnica tradicional, que é baseada em histogramas, é aproximadamente $2,5\%$ menor, o que representa uma economia próxima de 30 minutos. Ou seja, as mesmas consultas demoraram 30 minutos a menos para serem executadas no sistema com o nosso método. Em consequência, é concebível dizer que essa economia está relacionada com a mudança do PE escolhido para executar algumas consultas. Isto é, pelo motivo de que as estimativas de cardinalidades estarem mais acuradas, as estimativas de custo dos PEs equivalentes de uma dada consulta também estão, portanto a escolha do PE com menor custo anterior à execução se mostra a opção correta no tempo de execução. Para comprovar esse fato, escolhemos a consulta que apresentou o maior ganho de desempenho que é apresentada a seguir.

```
SELECT *
FROM cast_info CI, movie_info MI, title T
WHERE CI.movie_id = T.id AND MI.movie_id=T.id AND CI.role_id > 1 AND
      MI.info_type_id > 1 AND T.production_year = 1996;
```

Em síntese, essa consulta possui duas operações de junções e três de seleção. Entre os planos avaliados pelo PostgreSQL usando sua técnica padrão de estimativas, aquele com o menor custo estimado se encontra na Figura 38. Nesse plano, as operações de junção são implementadas pelo algoritmo *Hash Join*, que aplica uma estrutura de dados *hash* para realizar a operação. Para realizar as operações de seleção, realiza-se uma varredura sequencial (*SeqScan*) das três relações CI, MI e T.

Por outro lado, a Figura 39 apresenta o PE escolhido pelo PostgreSQL utilizando o método proposto para calcular as estimativas. Neste, as mesmas operações de junção são implementadas pelo algoritmo *Nested Loop*, que aplica uma iteração nos dados para realizar a

Figura 38 – PE escolhido pelo PostgreSQL usando a sua técnica de estimativas padrão para a consulta com maior ganho de desempenho.



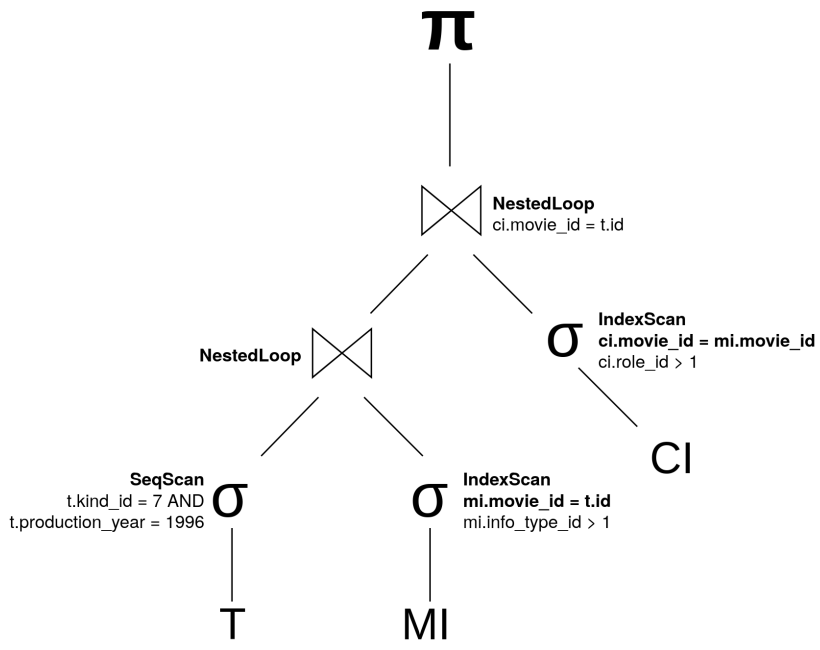
operação. Ao contrário de fazer uma varredura sequencial nos dados para realizar as operações de seleção, esse plano usa as estruturas de índices existentes para realizá-las. De modo geral, ao usar esses índices, as operações de seleção têm um custo menor associado, o que impactou no desempenho melhor desse PE.

Muito embora o desempenho geral usando o PostgreSQL modificado ter sido superior ao sem modificação, é possível observar que o ganho tido foi em torno de no máximo 2,5%. Com o intuito de analisar o porquê dessa situação, vamos analisar a seguinte consulta SQL que apresentou uma piora no seu tempo de execução.

```
SELECT *
FROM cast_info CI, movie_companies MC, title T
WHERE CI.movie_id = T.id AND MC.movie_id=T.id AND
      MC.company_type_id = 1 AND T.kind_id = 1;
```

Esta consulta é composta por duas operações de junção e duas de seleção. A Figura 40 ilustra o PE escolhido quando o PostgreSQL usa seu estimador padrão baseado em histogramas, enquanto que a Figura 41 mostra o PE escolhido quando o método proposto é empregado. Observando os PEs apresentados, é possível detectar que a única diferença entre eles

Figura 39 – PE escolhido pelo PostgreSQL usando o método proposto para a consulta com maior ganho de desempenho.



é a maneira pela qual as operações de junção são realizadas. Na verdade, a ordem de aplicação dessas operações. No primeiro, a ordem escolhida foi $CI \bowtie (MC \bowtie T)$. Em contrapartida, o segundo escolheu a ordem $(CI \bowtie MC) \bowtie T$. Como apresentado na Tabela 2, a operação de junção é associativa, ou seja, essas duas operações são equivalentes.

O plano escolhido usando a técnica padrão do PostgreSQL é esperado ter um menor custo tendo em vista que as duas operações de seleção vão diminuir o trabalho necessário para gerar o resultado da junção entre MC e T. O problema que ocorreu para o PostgreSQL não ter escolhido esse plano quando usou o método proposto foi que as estimativas de cardinalidade para as operações de seleção não foram tão acuradas quanto ao do PostgreSQL usando histogramas. Como ressaltado na apresentação desta técnica, os histogramas produzem estimativas com bastante qualidade para operações unidimensionais, como é o caso apresentado. O seu problema é nos casos em que é necessário estimar seleções em mais de um atributo, pois os histogramas multidimensionais não apresentam uma viabilidade para serem usados na prática. Aprofundar o estudo sobre como superar essa dificuldade do método proposto é uma possível direção para trabalhos futuros, assim como avaliar a possibilidade de se considerar a incerteza no processo de geração das estimativas.

Por fim, a Figura 42 mostra um gráfico de barras cujo intuito é ilustrar a diferença

Figura 40 – PE escolhido pelo PostgreSQL usando a sua técnica de estimativas padrão para a consulta com maior perda de desempenho.

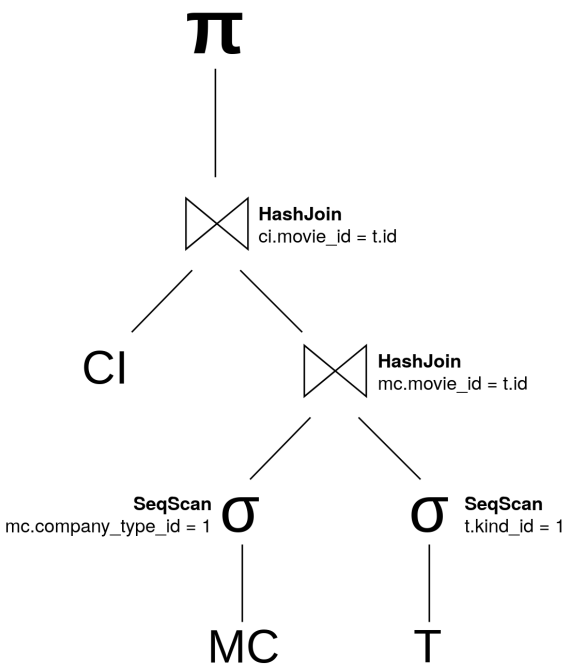


Figura 41 – PE escolhido pelo PostgreSQL usando o método proposto para a consulta com maior perda de desempenho.

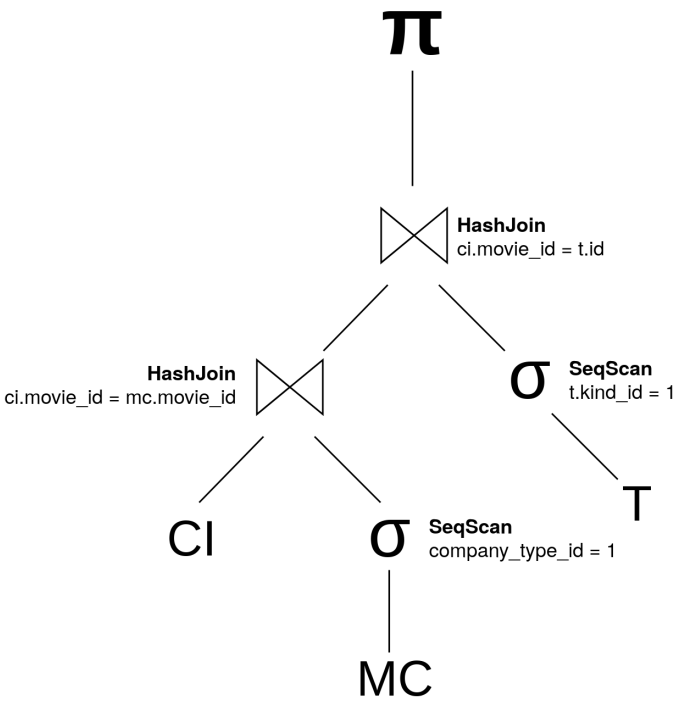
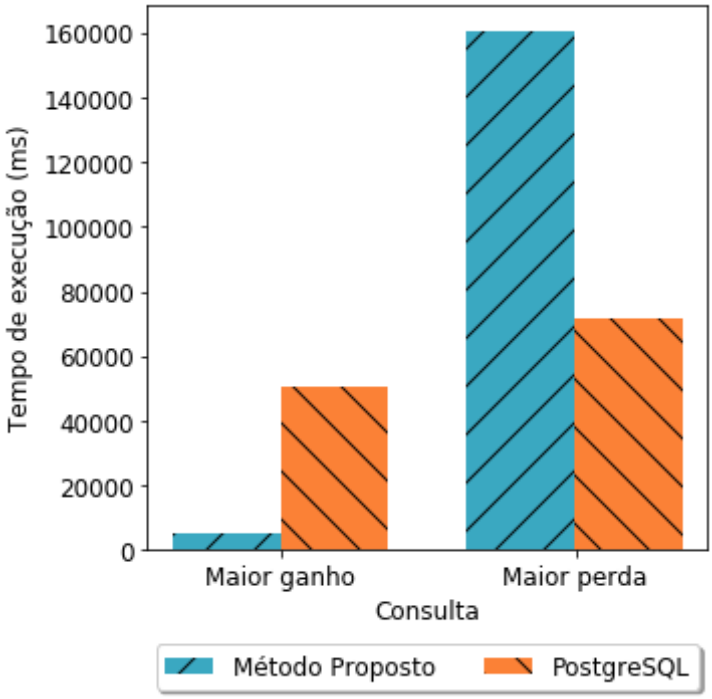


Figura 42 – Diferença entre os tempos de execução das consultas com maior ganho e perda usando a técnica padrão e o método proposto.



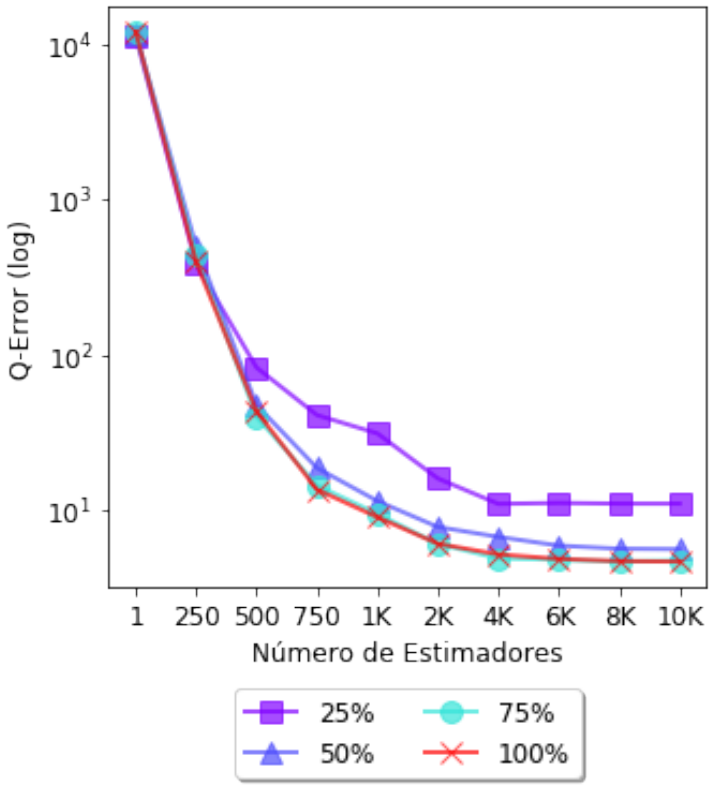
no tempo de execução das duas consultas mostradas acima.

Observando o gráfico, é possível observar que a primeira consulta, a de maior ganho, apresenta um tempo maior que $40K$ quando o PE escolhido é aquele estimado usando a técnica padrão. Quando o método proposto é utilizado, esse tempo cai para uma ordem de aproximadamente $5K$. Em relação à segunda consulta, temos que ela possui um tempo de execução da ordem de $70K$ utilizando a técnica padrão, enquanto que usando o método proposto esse tempo sobe para uma ordem de $160K$. A conclusão que podemos chegar é que o tempo ganho na primeira consulta é diluído pela perda da segunda, o que justifica o ganho geral de desempenho não ter sido superior à $2,5\%$.

5.4.4 Dados de Treinamento Necessários

Para treinar o modelo M utilizado nos experimentos, utilizamos um conjunto de dados de treinamento com um total de $90K$ consultas. Dado que o processo de gerar esse conjunto de consultas é demorado, um questionamento interessante é qual a quantidade realmente necessária para que o modelo seja treinado. A Figura 43 apresenta no eixo horizontal o número de estimadores e no vertical a métrica *Q-Error* em escala logarítmica. Cada curva mostrada nessa Figura representa o *Q-Error* obtido para um dado número de estimadores e uma dada

Figura 43 – Diferença entre o *Q-Error* obtido quando o treinamento do modelo é realizado com um número de estimadores e uma porcentagem do conjunto de dados de treinamento.



porcentagem de dados do conjunto de treinamento usados para treinar o modelo M.

O motivo de usar no eixo horizontal o número de estimadores base é que ele influencia no processo de treinamento. Como pode ser visualizado no gráfico, à medida que esse número vai aumentando, a métrica *Q-Error* vai diminuindo até estabilizar mais ou menos em $4K$. Ou seja, $4K$ estimadores de base seria suficiente para se obter um bom modelo treinado M. Agora analisando as diferentes curvas, veja que o *Q-Error* da curva roxa, que representa que somente 25% do conjunto de treinamento foi utilizado, está maior que 10, enquanto que todas as outras estão abaixo desse número, o que implica que 25% do conjunto de treinamento são poucos dados para treinar M. Aumentando para 50%, o *Q-Error* da curva azul escuro já melhora, mas ainda sim continua distante do cenário da curva vermelha, que representa que todo o conjunto de treinamento foi usado. Já para a curva azul clara, que representa o uso de 75% do conjunto de treinamento, o *Q-Error* obtido é bem próximo daquele que é obtido quando todo esse conjunto é utilizado. Dessa forma, em torno de $60K$ consultas já seriam suficientes para que o modelo M estivesse treinado.

5.5 Conclusão

Neste capítulo, uma avaliação experimental da técnica proposta foi apresentada. Os resultados dos experimentos realizados no PostgreSQL modificado para receber o método proposto foram mostrados com o intuito de testar a acurácia das estimativas de cardinalidade geradas e o impacto destas na escolha dos PEs no momento de executar as consultas. Estes resultados evidenciam que a Hipótese de Pesquisa 1 é verdadeira, mesmo tendo espaço para melhorias futuras.

6 CONSIDERAÇÕES FINAIS

6.1 Principais Contribuições

O tema abordado nesta dissertação diz respeito à maneira pela qual os SGBDs calculam as estimativas de cardinalidade para as consultas ao banco de dados. Especificamente, o objetivo geral do presente trabalho está amparado na literatura de BD que afirma que ainda existe muita melhoria a ser buscada nas técnicas de estimativas de cardinalidade utilizadas no processamento de consulta dos principais SGBDs em uso atualmente. Esta dissertação descreveu as fontes de falhas de estimativas a fim de fortalecer a motivação para o seu problema científico, o que constitui uma primeira contribuição. A principal contribuição desta dissertação foi descrita e detalhada no Capítulo 4 no qual foi apresentada uma nova abordagem para gerar essas estimativas baseada no uso de uma técnica de aprendizagem de máquina conhecido como *Gradient Boosting Machines*. Além disso, no contexto da abordagem proposta, duas contribuições foram apresentadas: (1) uma forma pela qual é realizada a modelagem das consultas para entrada de construção do modelo de aprendizagem e (2) uma arquitetura de integração de uso de técnicas de aprendizagem de máquina com o otimizador de consultas de um SGBD relacional. Realizamos uma experimentação em laboratório da abordagem proposta usando a arquitetura do PostgreSQL e uma implementação de GBM, conhecida como *LightGBM*. Os resultados comprovaram nossa hipótese de melhoria de predição da estimativas de cardinalidade em consultas relacionais. Levando em consideração essa implementação do método proposto em um SGBD e o seu uso ter se mostrado possível, então é plausível afirmar que tanto o objetivo geral quanto os objetivos específicos definidos no início deste trabalho foram alcançados.

6.2 Conclusão

Em conformidade com os resultados obtidos na avaliação experimental do método proposto e mostrados no Capítulo 5, pode-se concluir, então, que a Hipótese de Pesquisa 1 levantada no princípio deste trabalho se mostrou verdadeira considerando que o método proposto trouxe uma solução para o Problema de Pesquisa 1. É possível diminuir a ocorrência de falhas de predição da cardinalidade de consultas nos SGBDs relacionais com a ajuda de técnicas de aprendizagem de máquinas como a GBM. Essas técnicas podem trazer efetividade nas atividades realizadas dentro do âmbito do processamento de consultas dos SGBDs. Especificamente, esses

modelos podem ser utilizados para identificar as características presentes em uma base de dados, e a partir dessas informações, possibilitar que os SGBDs decidam como realizar o processamento das consultas.

6.3 Trabalhos Futuros

O amadurecimento das técnicas e a disponibilidade de ferramentas de aprendizagem de máquina têm facilitado o uso de várias dessas ferramentas quando há igualmente a disponibilidade significativo de dados que permitam o treinamento. Seguindo esta tendência, vários trabalhos na literatura trataram da incorporação dessas técnicas nos SGBDs de maneira enfática, procurando dar-lhes a capacidade de se estruturar de acordo com as características das suas bases de dados e da forma como os dados são processados ao longo do tempo. Nesse cenário, e especificamente no de processamento de consultas, há vários problemas em aberto que ainda precisam ser atacados. Por exemplo, lidar com dados numéricos é uma limitação deste trabalho. Além disso, há também a oportunidade de melhoria do método proposto que foi discutida na Seção 5.4.3. Também relacionado ao método proposto, uma ideia de pesquisa é como incorporar o gerenciamento de incerteza no processo de predição das cardinalidades, ou seja, como fazer o otimizador de consultas levar em consideração também a incerteza, visto que essa já pode ser avaliada como descrito na Seção 4.6.

Um caminho ambicioso para um possível trabalho futuro é investigar como propor uma técnica *online* que se ajusta à medida em que os dados e a carga de trabalho vão sendo modificados. Por fim, a indagação de quais partes da arquitetura de um SGBD, não atrelado somente ao processamento de consultas, podem se beneficiar da utilização de técnicas de aprendizagem de máquina é um outro rumo a ser explorado e pode ser o ponto inicial de uma possível pesquisa acadêmica desafiadora.

REFERÊNCIAS

ALPAYDIN, E. **Introduction to machine learning**. [*S. l.*]: MIT press, 2009.

ASTRAHAN, M. M.; BLASGEN, M. W.; CHAMBERLIN, D. D.; ESWARAN, K. P.; GRAY, J.; GRIFFITHS, P. P.; III, W. F. K.; LORIE, R. A.; MCJONES, P. R.; MEHL, J. W.; PUTZOLU, G. R.; TRAIGER, I. L.; WADE, B. W.; WATSON, V. System R: relational approach to database management. **ACM Trans. Database Syst.**, v. 1, n. 2, p. 97–137, 1976.

BISHOP, C. M. **Pattern recognition and machine learning, 5th Edition**. Springer, 2007. (Information science and statistics). ISBN 9780387310732. Disponível em: http://www.worldcat.org/oclc/71008143. Acesso em: 15 julho. 2019.

BREIMAN, L.; FRIEDMAN, J.; OLSHEN, R.; STONE, C. Classification and regressiontrees, wadsworth statistics. **Probability Series, Belmont, California: Wadsworth**, 1984.

CHAUDHURI, S.; NARASAYYA, V. R.; RAMAMURTHY, R. A pay-as-you-go framework for query execution feedback. **PVLDB**, v. 1, n. 1, p. 1141–1152, 2008.

CODD, E. F. A relational model of data for large shared data banks. **Commun. ACM**, v. 13, n. 6, p. 377–387, 1970.

DEVELOPERS, G. **Clustering in Machine Learning**. 2019. Disponível em: https://developers.google.com/machine-learning/clustering/clustering-algorithms. Acesso em: 7 jan. 2020.

DUTT, A.; WANG, C.; NAZI, A.; KANDULA, S.; NARASAYYA, V. R.; CHAUDHURI, S. Selectivity estimation for range predicates using lightweight models. **PVLDB**, v. 12, n. 9, p. 1044–1057, 2019.

ELMASRI, R.; NAVATHE, S. B. **Fundamentals of Database Systems, 3rd Edition**. [*S. l.*]: Addison-Wesley-Longman, 2000. ISBN 978-0-8053-1755-8.

FRIEDMAN, J. H. Greedy function approximation: a gradient boosting machine. **Annals of statistics**, JSTOR, p. 1189–1232, 2001.

GRAEFE, G. Volcano - an extensible and parallel query evaluation system. **IEEE Trans. Knowl. Data Eng.**, v. 6, n. 1, p. 120–135, 1994.

GRAEFE, G.; GUY, W.; KUNO, H. A.; PAULLEY, G. N. Robust query processing (dagstuhl seminar 12321). **Dagstuhl Reports**, v. 2, n. 8, p. 1–15, 2012.

HASHEM, S.; SCHMEISER, B. Improving model accuracy using optimal linear combinations of trained neural networks. **IEEE Transactions on neural networks**, IEEE, v. 6, n. 3, p. 792–794, 1995.

HUANG, G.; ZHOU, H.; DING, X.; ZHANG, R. Extreme learning machine for regression and multiclass classification. **IEEE Trans. Systems, Man, and Cybernetics, Part B**, v. 42, n. 2, p. 513–529, 2012.

IBARAKI, T.; KAMEDA, T. On the optimal nesting order for computing n-relational joins. **ACM Trans. Database Syst.**, v. 9, n. 3, p. 482–502, 1984.

IOANNIDIS, Y. E.; CHRISTODOULAKIS, S. On the propagation of errors in the size of join results. In: **Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data, Denver, Colorado, USA, May 29-31, 1991.** [*S. l.*: *s. n.*], 1991. p. 268–277.

JANSEN, S. **Hands-On Machine Learning for Algorithmic Trading**. 1st. ed. [*S. l.*]: Packt Publishing, 2018.

JONES, E.; OLIPHANT, T.; PETERSON, P. *et al.* **SciPy: Open source scientific tools for Python**. 2001. Disponível em: http://www.scipy.org/. Acesso em: 8 mar. 2020.

KE, G.; MENG, Q.; FINLEY, T.; WANG, T.; CHEN, W.; MA, W.; YE, Q.; LIU, T.-Y. Lightgbm: A highly efficient gradient boosting decision tree. In: **Advances in Neural Information Processing Systems**. [*S. l.*: *s. n.*], 2017. p. 3146–3154.

KIPF, A.; KIPF, T.; RADKE, B.; LEIS, V.; BONCZ, P. A.; KEMPER, A. Learned cardinalities: Estimating correlated joins with deep learning. In: **CIDR 2019, 9th Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings**. [*S. l.*: *s. n.*], 2019.

KOENKER, R.; HALLOCK, K. F. Quantile regression. **Journal of economic perspectives**, v. 15, n. 4, p. 143–156, 2001.

KÖNIG, A. C. **Query estimation techniques in database systems**. Tese (Doutorado) – Saarland University, Saarbrücken, Germany, 2001.

KOOI, R. P. **The Optimization of Queries in Relational Databases**. Tese (Doutorado) – Case Western Reserve University, Cleveland, OH, USA, 1980. AAI8109596.

LEIS, V.; GUBICHEV, A.; MIRCHEV, A.; BONCZ, P. A.; KEMPER, A.; NEUMANN, T. How good are query optimizers, really? **PVLDB**, v. 9, n. 3, p. 204–215, 2015.

LEIS, V.; RADKE, B.; GUBICHEV, A.; KEMPER, A.; NEUMANN, T. Cardinality estimation done right: Index-based join sampling. In: **CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings**. [*S. l.*: *s. n.*], 2017.

LEIS, V.; RADKE, B.; GUBICHEV, A.; MIRCHEV, A.; BONCZ, P. A.; KEMPER, A.; NEUMANN, T. Query optimization through the looking glass, and what we found running the join order benchmark. **VLDB J.**, v. 27, n. 5, p. 643–668, 2018.

LIPTON, R. J.; NAUGHTON, J. F.; SCHNEIDER, D. A. Practical selectivity estimation through adaptive sampling. In: **Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, USA, May 23-25, 1990.** [*S. l.*: *s. n.*], 1990. p. 1–11.

LIU, H.; XU, M.; YU, Z.; CORVINELLI, V.; ZUZARTE, C. Cardinality estimation using neural networks. In: **Proceedings of 25th Annual International Conference on Computer Science and Software Engineering, CASCON 2015, Markham, Ontario, Canada, 2-4 November, 2015**. [*S. l.*: *s. n.*], 2015. p. 53–59.

MARCUS, R.; PAPAEMMANOUIL, O. Towards a hands-free query optimizer through deep learning. In: **CIDR 2019, 9th Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings**. [*S. l.*: *s. n.*], 2019.

MARQUÉS, A. I.; GARCÍA, V.; SÁNCHEZ, J. S. Exploring the behaviour of base classifiers in credit scoring ensembles. **Expert Syst. Appl.**, v. 39, n. 11, p. 10244–10250, 2012.

MICROSOFT. **LightGBM**. 2017. Disponível em: https://github.com/Microsoft/LightGBM/blob/master/docs/Features.rst. Acesso em: 15 jan. 2020.

MOERKOTTE, G.; NEUMANN, T.; STEIDL, G. Preventing bad plans by bounding the impact of cardinality estimation errors. **PVLDB**, v. 2, n. 1, p. 982–993, 2009.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

POSTGRESQL. **PostgreSQL's backend flowchart**. 2012. Disponível em: https://www.postgresql.org/media/img/developer/backend/flow.gif. Acesso em: 5 fev. 2020.

ROGOZHNIKOV, A. **Gradient Boosting explained [demonstration]**. 2016. Disponível em: http://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html. Acesso em: 15 jan. 2020.

SOLLICH, P.; KROGH, A. Learning with ensembles: How overfitting can be useful. In: **Advances in neural information processing systems**. [*S. l.*: *s. n.*], 1996. p. 190–196.

STILLGER, M.; LOHMAN, G. M.; MARKL, V.; KANDIL, M. LEO - db2's learning optimizer. In: **VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy**. [*S. l.*: *s. n.*], 2001. p. 19–28.

STONEBRAKER, M.; HELLERSTEIN, J. What goes around comes around. **Readings in Database Systems**, v. 4, p. 1724–1735, 2005.

SUTTON, R. S.; BARTO, A. G. *et al.* **Introduction to reinforcement learning**. [*S. l.*]: MIT press Cambridge, 1998. v. 2.

WANG, W.; ZHANG, M.; CHEN, G.; JAGADISH, H. V.; OOI, B. C.; TAN, K. Database meets deep learning: Challenges and opportunities. **CoRR**, abs/1906.08986, 2019.

WEISS, S. M.; KULIKOWSKI, C. A. **Computer Systems That Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1991. ISBN 1558600655.

WOLTMANN, L.; HARTMANN, C.; THIELE, M.; HABICH, D.; LEHNER, W. Cardinality estimation with local deep learning models. In: **Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, aiDM@SIGMOD 2019, Amsterdam, The Netherlands, July 5, 2019**. [*S. l.*: *s. n.*], 2019. p. 5:1–5:8.

ZAHEER, M.; KOTTUR, S.; RAVANBAKHSH, S.; POCZOS, B.; SALAKHUTDINOV, R. R.; SMOLA, A. J. Deep sets. In: **Advances in neural information processing systems**. [*S. l.*: *s. n.*], 2017. p. 3391–3401.

ZUKOWSKI, M. Balancing vectorized query execution with bandwidth-optimized storage. **Journal of Computational Physics - J COMPUT PHYS**, 01 2009.