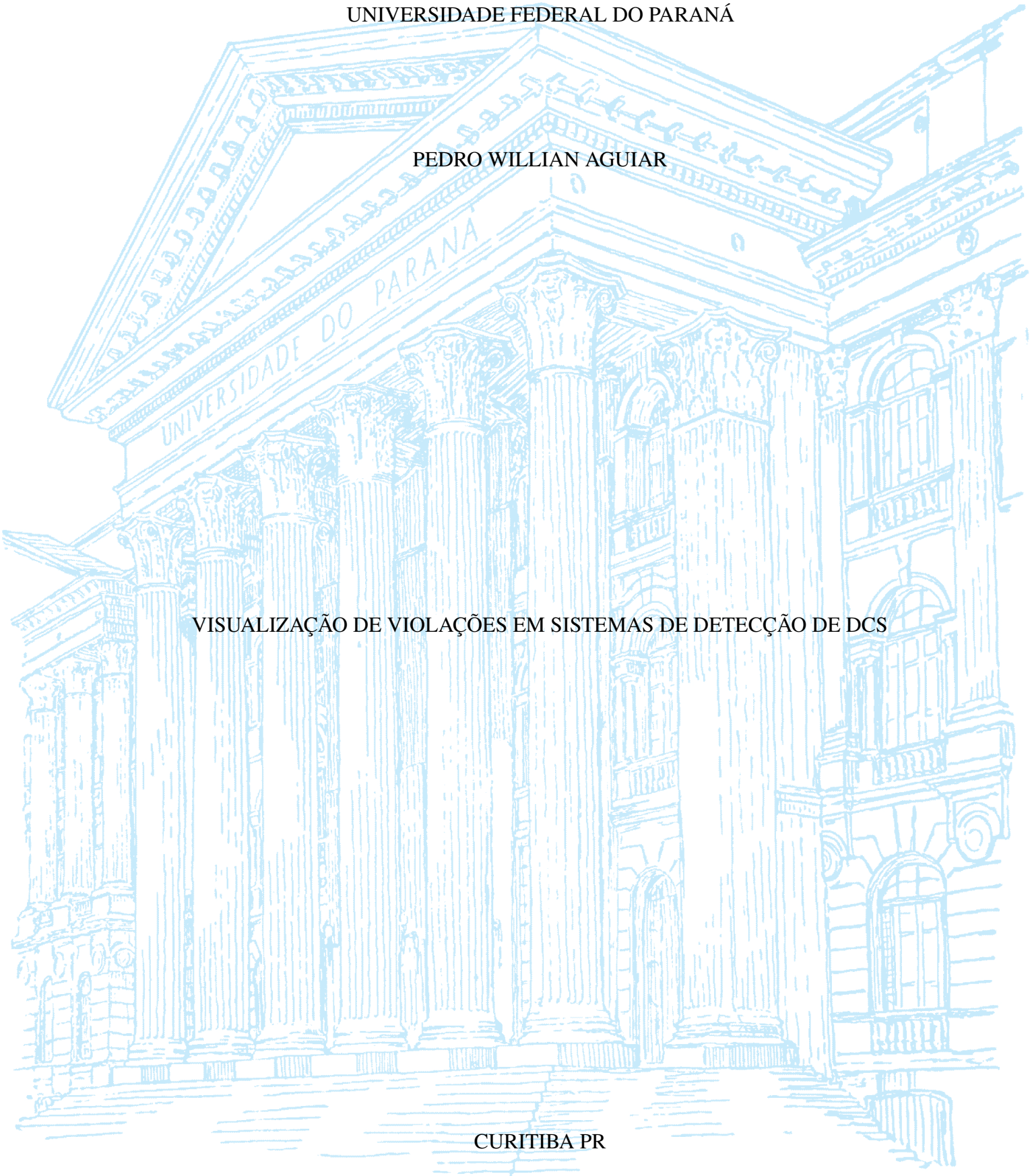
UNIVERSIDADE FEDERAL DO PARANÁ

PEDRO WILLIAN AGUIAR



VISUALIZAÇÃO DE VIOLAÇÕES EM SISTEMAS DE DETECÇÃO DE DCS

CURITIBA PR

2025

PEDRO WILLIAN AGUIAR

VISUALIZAÇÃO DE VIOLAÇÕES EM SISTEMAS DE DETECÇÃO DE DCS

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Banco de Dados*.

Orientador: Eduardo Cunha de Almeida.

CURITIBA PR

2025

Dedico esse trabalho à minha família, meus amigos, professores e comunidade acadêmica da UFPR.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, Pedro e Izabel, que são os pilares da minha vida. Obrigado pelo amor incondicional, pelo suporte constante e por todo o sacrifício e dedicação para que eu pudesse chegar até aqui. Tudo o que sou e conquistei devo a vocês.

À minha namorada e companheira de vida, Bianca. Obrigado pelo amor, pela escuta atenta nos momentos difíceis e, acima de tudo, pela paciência infinita durante todo este processo. Sua presença tornou a caminhada mais leve.

Às minhas irmãs, Karina e Jacqueline, por sempre acreditarem no meu potencial, pelo carinho e por estarem sempre presentes. À minha sobrinha Luiza, que trouxe felicidade e brilho para as nossas vidas com sua alegria contagiante.

Ao meu orientador, Professor Eduardo, expresso minha profunda gratidão. Obrigado pelo apoio, pelas dicas, pelo incentivo constante e pela orientação precisa que foram fundamentais para a conclusão deste trabalho.

Estendo meus agradecimentos a todos os professores que tive ao longo da graduação, que foram fontes de inspiração e aprendizado, moldando não apenas o meu conhecimento técnico, mas também minha visão profissional.

Por fim, aos meus amigos — seria injusto nomear apenas alguns — obrigado pela amizade, pelas risadas que aliviaram a tensão e pelo apoio inestimável durante esse período.

RESUMO

Sistemas de detecção de erros de dados, como o FACET (FAst Constraint-based Error DeTector), são rápidos para encontrar violações de Restrições de Negação (DCs), mas seus resultados são insuficientes para a análise humana, informando apenas contagens de erros ou identificadores abstratos (tids). Esta lacuna impede o diagnóstico da causa raiz dos problemas de qualidade de dados. Este trabalho contribui com a apresentação de uma solução de software desenvolvida para resolver este problema. O motor do FACET foi modificado para coletar, durante a detecção, duas formas de metadados: estatísticas agregadas (contagens de violações por tupla) usando *Hash Maps* e uma amostra probabilística imparcial dos pares violadores, via *Reservoir Sampling*. Uma aplicação web, foi desenvolvida para orquestrar a análise. O *backend* executa o FACET modificado, utiliza um módulo de enriquecimento para buscar os dados completos das tuplas amostradas e envia um JSON consolidado para o *frontend*. A interface de visualização apresenta os resultados em um painel interativo, exibindo as estatísticas, os quartis, um *ranking* "Top 10" das tuplas mais problemáticas e as amostras de violação. Os experimentos demonstraram que a ferramenta permite ao analista diagnosticar rapidamente diferentes padrões de erro, validando a solução como uma ponte eficaz entre a detecção e a interpretação de erros.

Palavras-chave: Restrições de Negação. Detecção de Erros. FACET. Visualização de Dados. Análise Exploratória de Dados.

ABSTRACT

Data error detection systems, such as FACET, are fast at finding Denial Constraint (DC) violations, but their results are insufficient for human analysis, reporting only error counts or abstract identifiers (tids). This gap hinders the root cause diagnosis of data quality problems. This work contributes by presenting a software solution developed to address this problem. The FACET engine was modified to collect, during detection, two forms of metadata: aggregate statistics (per-tuple violation counts) using Hash Maps and an unbiased probabilistic sample of violating pairs, via Reservoir Sampling. A web application was developed to orchestrate the analysis. The backend executes the modified FACET, uses an enrichment module to fetch the complete data of the sampled tuples, and sends a consolidated JSON to the frontend. The visualization interface presents the results in an interactive dashboard, displaying statistics, quartiles, a "Top 10" ranking of the most problematic tuples, and the violation samples. The experiments demonstrated that the tool allows the analyst to quickly diagnose different error patterns, validating the solution as an effective bridge between error detection and interpretation.

Keywords: Denial Constraints. Error Detection. FACET. Data Visualization. Exploratory Data Analysis.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

AIQ	Amplitude Interquartil
API	*Application Programming Interface* (Interface de Programação de Aplicações)
AQP	Processamento Aproximado de Consultas
BCNF	*Boyce-Codd Normal Form* (Forma Normal de Boyce-Codd)
CSV	*Comma-Separated Values* (Valores Separados por Vírgula)
DC	*Denial Constraint* (Restrição de Negação)
EDA	*Exploratory Data Analysis* (Análise Exploratória de Dados)
FACET	*FAst Constraint-based Error DeTector*
FD	*Functional Dependency* (Dependência Funcional)
JSON	*JavaScript Object Notation*
SQL	*Structured Query Language* (Linguagem de Consulta Estruturada)
TID	*Tuple Identifier* (Identificador de Tupla)
UCC	*Unique Column Combination* (Combinação Única de Colunas)

SUMÁRIO

1 INTRODUÇÃO

A qualidade dos dados é um pilar fundamental para a tomada de decisões confiáveis, o treinamento de modelos de aprendizado de máquina e a eficácia de processos operacionais em qualquer organização. No entanto, dados do mundo real são notoriamente imperfeitos, contendo uma vasta gama de erros como valores ausentes, inconsistências, duplicatas e imprecisões (Rahm e Do, 2000). O processo de identificação e correção desses erros, conhecido como limpeza de dados (*data cleaning*), é uma tarefa crítica, porém complexa e frequentemente dispendiosa.

Para formalizar a detecção de inconsistências, a comunidade de pesquisa propôs o uso de regras declarativas, com destaque para as Restrições de Negação (DCs – *Denial Constraints*). Estas regras são capazes de capturar uma ampla variedade de erros que envolvem múltiplas tuplas e comparações complexas, superando a expressividade de restrições tradicionais como as Dependências Funcionais (Fan et al., 2008; Chu et al., 2013).

Contudo, a alta expressividade das DCs implica um custo computacional elevado. Para mitigar esse problema, algoritmos de alta performance como o FACET (Pena et al., 2022) foram desenvolvidos. O FACET opera sobre identificadores de tuplas (tids) evitando a manipulação custosa dos dados brutos. Embora essa abordagem seja eficiente para quantificar erros, ela cria uma lacuna na interpretação: saber que existem 50.000 violações é insuficiente se o analista não consegue visualizar quais dados geraram o erro e por que.

A literatura enfatiza que a detecção é apenas o primeiro passo; a compreensão e correção exigem envolvimento humano apoiado por ferramentas de diagnóstico (Chu e Ilyas, 2016; Kandel et al., 2012). Atualmente, a visualização de violações de DCs é difícil porque envolve relacionamentos complexos entre pares de tuplas, não sendo facilmente representável por histogramas ou gráficos simples de distribuição.

Este trabalho propõe resolver essa lacuna estendendo o FACET de uma ferramenta de detecção para uma plataforma de análise.A contribuição central é uma interface de visualização interativa acoplada a um *backend* modificado, que materializa a tupla completa dos dados violadores de restrições fornecendo estatísticas agregadas e amostras detalhadas.

Para validar a proposta, foram conduzidos experimentos utilizando *datasets* reais e sintéticos amplamente adotados na literatura (`Actorkey`, `Flights` e `Tax`), cobrindo diferentes cenários de erro como duplicatas, inconsistências lógicas e anomalias numéricas. Os resultados demonstram como a ferramenta auxilia na identificação de padrões de erro sistêmicos e pontuais.

O restante deste trabalho está organizado da seguinte forma. O Capítulo 2 apresenta os fundamentos teóricos e discute a lacuna de ferramentas de visualização para DCs. O Capítulo 3 descreve a proposta e as perguntas de pesquisa. O Capítulo 4 detalha as modificações no FACET. O Capítulo 5 apresenta a interface web e o fluxo de dados. O Capítulo 6 discute os resultados experimentais. Por fim, o Capítulo 7 apresenta as conclusões.

2 FUNDAMENTAÇÃO

2.1 RESTRIÇÕES DE NEGAÇÃO (DENIAL CONSTRAINTS)

No campo da qualidade de dados, é fundamental dispor de mecanismos robustos para definir e identificar inconsistências. As Restrições de Negação surgem como um formalismo de destaque para modelar regras de negócio e garantir a integridade dos dados. Sua principal vantagem é atuar como uma generalização de outras restrições. Essa generalidade advém de sua estrutura flexível, baseada em uma conjunção de predicados lógicos que pode ser livremente definida. Isso permite que as DCs não apenas expressem as regras de restrições mais especializadas, como as Dependências Funcionais (FDs) e as Combinações Únicas de Colunas (UCCs), mas também capturem regras de negócio complexas que envolvem comparações de ordem ou múltiplas condições entre atributos (Chu et al., 2013).

A ideia fundamental de uma DC é especificar um conjunto de condições, ou predicados, que não podem ser verdadeiros simultaneamente em um banco de dados. Se uma combinação de tuplas (registros) satisfaz todas as condições de uma DC, uma violação é detectada, sinalizando um erro ou um estado inconsistente nos dados (Pena et al., 2022).

2.1.1 Definição Formal

Formalmente, uma Restrição de Negação ϕ é expressa através da lógica de primeira ordem, utilizando a seguinte notação, conforme apresentado por Pena et al. (2022):

$$\phi : \forall t, t' \in r, \neg(p_1 \wedge p_2 \wedge \cdots \wedge p_m)$$

Onde:

- $\forall t, t' \in r$ indica que a regra se aplica a todo par de tuplas (registros) t e t' pertencentes a uma relação r (tabela). Em alguns casos, a regra pode se aplicar a uma única tupla (onde $t = t'$).
- $p_1, p_2, \ldots, p_m$ são predicados, que são comparações lógicas entre atributos das tuplas.
- $\wedge$ representa o operador lógico **"E"** (conjunção).
- $\neg$ representa o operador lógico **"NÃO"** (negação).

Em essência, a fórmula "nega" a possibilidade de que todos os predicados sejam verdadeiros ao mesmo tempo. Uma violação de ϕ ocorre quando a conjunção $(p_1 \wedge \cdots \wedge p_m)$ resulta em verdadeiro para algum par de tuplas, pois isso contradiz a negação imposta pela restrição.

2.1.1.0 Propriedades de uma Restrição de Negação

A definição formal de uma DC vai além de sua fórmula lógica, abrangendo um conjunto de propriedades que buscam garantir sua validade e utilidade. Contudo, um desafio persistente na área de descoberta de DCs é que as definições tradicionais de validade frequentemente resultam na descoberta de um grande volume de restrições "falsa" ou espúrias, que não capturam a semântica real dos dados, com proporções de falsos positivos que raramente ficam abaixo de 95% (Martin et al., 2025).

Tabela 2.1: Exemplo de Tabela de Gerenciamento de Projetos

ID_Projeto	Nome do Projeto	Gerente	Data de Início	Data Fim Prevista	Status	Orçamento
101	Sistema Alpha	Ana	2024-03-15	2024-09-30	Em Andamento	50000.00
102	App Mobile Beta	Carlos	2024-02-01	2024-08-01	Concluído	35000.00
103	Portal Interno	**Beatriz**	**2024-05-20**	**2024-04-30**	**Planejado**	**20000.00**
104	BI Analytics	**Ana**	**2024-06-10**	**2024-12-20**	**Concluído**	**0.00**
101	Sistema Alpha	**Fernanda**	2024-03-15	2024-09-30	Em Andamento	50000.00

Segundo Martin et al. (2025), o conjunto de propriedades tradicionalmente utilizado na literatura para definir uma DC válida inclui:

Satisfatibilidade: A DC deve ser satisfeita por 100% dos pares de tuplas, embora essa condição possa ser relaxada por um fator de aproximação η.

Simetria: A validade da DC para um par de tuplas é invariante à ordem das tuplas.

Minimalidade: A remoção de qualquer predicado da conjunção torna a DC inválida.

Não-trivialidade: A DC não é satisfeita por todos os estados de dados possíveis.

O problema, como apontado por Martin et al. (2025), é que essas propriedades não são suficientes para garantir a "solidez" (*soundness*) semântica. Para endereçar a alta taxa de descoberta de DCs falsas, Martin et al. (2025) propõem uma redefinição dessas propriedades, introduzindo novos conceitos:

Solidez (Soundness): Garante que a satisfação da DC não é alcançada artificialmente por meio de predicados independentes. Para assegurar a solidez, o artigo propõe a propriedade de **Atomicidade**, que determina que nenhum subconjunto de predicados pode ser removido sem alterar o comportamento de algum predicado restante (Martin et al., 2025).

Não-trivialidade (Redefinida): A DC não deve restringir estados de dados que já são impossíveis de ocorrer (por exemplo, restringir uma condição que a própria semântica do banco de dados já impede) (Martin et al., 2025).

Essa redefinição visa aprimorar o processo de descoberta, filtrando restrições logicamente válidas mas semanticamente inúteis, e aproximando os resultados algorítmicos das regras de negócio que os analistas de dados de fato procuram.

2.1.2 Exemplo Prático de Aplicação

Para ilustrar o funcionamento das DCs, considere a Tabela 2.1, que armazena informações sobre projetos em uma empresa de desenvolvimento de software.

A seguir, definimos três regras de negócio e suas respectivas DCs para identificar os erros presentes na Tabela 2.1.

2.1.2.0 Regra 1: O ID de um projeto deve ser único.

Esta é uma restrição de chave primária, mas pode ser expressa como uma DC para encontrar duplicatas.

- **DC_1:** $\forall t, t' \in \text{Projetos}, \neg(t.\text{ID\_Projeto} = t'.\text{ID\_Projeto} \wedge t \neq t')$

2.1.2.0 Regra 2: A data de fim prevista de um projeto não pode ser anterior à sua data de início.

Esta é uma regra de consistência temporal fundamental.

- **DC_2:** $\forall t \in$ Projetos, $\neg(t.\text{Data\_Fim\_Prevista} < t.\text{Data\_Inicio})$

2.1.2.0 Regra 3: Projetos com status "Concluído"devem ter um orçamento maior que zero.

Esta é uma regra de negócio que assume que todo projeto finalizado teve algum custo.

- **DC_3:** $\forall t \in$ Projetos, $\neg(t.\text{Status} = \text{'Concluído'} \wedge t.\text{Orcamento} = 0.00)$

2.1.3 Combinações Únicas de Colunas (UCC)

A descoberta de Combinações Únicas de Colunas (UCC), do inglês *Unique Column Combination*, é um problema central no perfilamento de dados, essencial para garantir a qualidade e otimizar o desempenho de consultas (Heise et al., 2013). Uma UCC é definida como um conjunto de um ou mais atributos cujos valores, quando combinados, são únicos para cada registro em uma relação. Dito de outra forma, não existem tuplas duplicadas ao se projetar os dados apenas nessas colunas.

A importância das UCCs se estende desde a identificação de chaves candidatas em modelagem de dados até a otimização de consultas e detecção de anomalias. No entanto, o processo de descoberta é um desafio notório, sendo um problema NP-difícil devido ao número exponencial de combinações de colunas a serem verificadas (Heise et al., 2013). Uma UCC é dita mínima se nenhum de seus subconjuntos próprios também for uma UCC, representando assim a forma mais concisa de uma chave única.

Um conceito derivado e de grande interesse é o de Combinação Única de Colunas Mínima (mUC). Uma UCC é considerada mínima se nenhum de seus subconjuntos próprios também for uma UCC. São essas mUCs as mais informativas, pois representam as chaves mais concisas e fundamentais da tabela.

2.1.4 Dependências Funcionais (FD)

A dependência funcional representa uma restrição de integridade entre dois conjuntos de atributos, aqui denominados X e Y, dentro de um mesmo esquema de relação. Representada pela notação $X \rightarrow Y$, a DF impõe a regra de que, para qualquer par de tuplas em uma tabela, se os valores dos atributos em X forem idênticos, os valores dos atributos em Y também deverão ser, obrigatoriamente, idênticos. Isso significa que o conjunto de atributos X determina funcionalmente o conjunto Y. Conforme Elmasri e Navathe (2016), tal dependência é uma propriedade intrínseca do esquema da relação, definida com base na semântica dos dados, e não pode ser simplesmente inferida a partir de uma instância particular da tabela.

O propósito central das dependências funcionais reside na análise e no aprimoramento do projeto de bancos de dados. Elas servem como a principal ferramenta para a normalização de esquemas relacionais, um processo que visa a organização dos atributos em tabelas para minimizar a redundância de dados e evitar as conhecidas anomalias de inserção, atualização e exclusão. A avaliação das DFs de uma relação permite verificar se ela atende aos critérios das formas normais, como a Segunda Forma Normal (2FN), a Terceira Forma Normal (3FN) e a Forma Normal de Boyce-Codd (BCNF) (Elmasri e Navathe, 2016).

A relação entre dependências funcionais e chaves é direta. Uma chave candidata de uma relação, por definição, determina funcionalmente todos os outros atributos dessa mesma relação.

Portanto, se K é uma chave candidata da relação R, a dependência funcional $K \rightarrow R$, onde R compreende todos os atributos da relação, é sempre considerada válida.

2.2 DESCOBERTA E TRATAMENTO DE VIOLAÇÕES

A abordagem para o tratamento de violações em sistemas de banco de dados, conforme detalhado por Elmasri e Navathe (2016), é multifacetada, abrangendo desde o projeto preventivo do esquema até mecanismos de detecção em tempo de execução e protocolos de recuperação de falhas. A estratégia não se limita a um único mecanismo, mas a um conjunto de técnicas que operam em diferentes estágios do ciclo de vida dos dados.

A primeira linha de defesa contra inconsistências ocorre na fase de projeto do esquema relacional. Nesta etapa, utilizam-se ferramentas formais, com destaque para as dependências funcionais, para analisar a qualidade do agrupamento de atributos. Este formalismo é a base para o processo de normalização, através do qual o esquema é decomposto para satisfazer formas normais como a Terceira Forma Normal e a Forma Normal de Boyce-Codd. O objetivo principal da normalização é projetar tabelas que minimizem a redundância e, consequentemente, previnam as anomalias de inserção, atualização e exclusão que surgem de um projeto inadequado (Elmasri e Navathe, 2016).

Em tempo de execução, os sistemas de gerenciamento de banco de dados empregam mecanismos explícitos para impor as regras definidas. A linguagem SQL oferece restrições de integridade como `PRIMARY KEY`, `UNIQUE` e `FOREIGN KEY` para validar os dados no momento em que são inseridos ou modificados. Quando uma operação viola uma dessas restrições, a ação padrão do sistema é rejeitá-la. Além disso, gatilhos (*triggers*) e asserções (`CREATE ASSERTION`) podem ser implementados para monitorar operações de forma mais complexa e executar lógicas personalizadas caso uma condição de violação seja detectada (Elmasri e Navathe, 2016).

Além dos mecanismos de imposição em tempo real, a detecção de violações em um conjunto de dados existente, uma tarefa central do perfilamento de dados, requer algoritmos específicos, cuja complexidade varia conforme o tipo de restrição.

Para restrições mais simples, como as Combinações Únicas de Colunas e as Dependências Funcionais, a detecção pode ser realizada de forma eficiente através de operações de agregação em SQL. No caso das UCCs, a violação é identificada ao agrupar os dados pelas colunas da restrição e filtrar os grupos com mais de um registro, utilizando a expressão `GROUP BY ... HAVING COUNT(*) > 1`. De forma análoga, para uma Dependência Funcional (X→Y), o método consiste em agrupar os dados pelos atributos do determinante X e verificar se algum grupo contém mais de um valor distinto para o dependente Y, geralmente com a cláusula `HAVING COUNT(DISTINCT Y) > 1` (Elmasri e Navathe, 2016).

A detecção de violações para DCs, por sua vez, é um desafio computacionalmente mais intenso. A abordagem de força bruta, que compara cada par de tuplas do banco de dados (equivalente a uma auto-junção ou *self-join*), possui uma complexidade quadrática ($O(n^2)$) e se torna inviável para grandes volumes de dados. Para superar essa limitação, algoritmos otimizados como o FACET foram desenvolvidos. Tais algoritmos evitam a comparação exaustiva através de estratégias como o particionamento dos dados com base em predicados de igualdade e a reordenação inteligente dos predicados restantes. Ao avaliar primeiro as condições mais seletivas, o sistema consegue "podar" drasticamente o espaço de busca, analisando apenas um subconjunto relevante de pares de tuplas e tornando a detecção de erros complexos viável em larga escala (Pena et al., 2022).

2.2.0.1 *Exemplos de Violações de Restrições de Negação*

Para ilustrar como os algoritmos de detecção identificam violações de DCs na prática, aplicamos as seguintes regras à Tabela 2.1:

1. Regra de Unicidade: Uma restrição de chave primária expressa como uma DC.
   - DC_1: $\forall t, t' \in$ Projetos, $\neg(t.\text{ID\_Projeto} = t'.\text{ID\_Projeto} \wedge t \neq t')$
   - *Análise da Violação:* Os registros com `ID_Projeto = 101` violam esta DC. Ao comparar a primeira e a última linha (t e t'), temos que t.ID_Projeto = t'.ID_Projeto (ambos são 101) e $t \neq t'$ (os gerentes são diferentes). Como a conjunção de predicados é verdadeira, a regra é violada, apontando um erro grave de integridade.
2. Regra de Consistência Temporal: Uma regra fundamental de lógica de dados.
   - DC_2: $\forall t \in$ Projetos, $\neg(t.\text{Data\_Fim\_Prevista} < t.\text{Data\_Inicio})$
   - *Análise da Violação:* O registro com `ID_Projeto = 103` viola esta DC. Neste caso, t.Data_Fim_Prevista (2024-04-30) é menor que t.Data_Inicio (2024-05-20). O predicado é verdadeiro, o que constitui uma violação da regra, indicando um erro de digitação ou de lógica no planejamento do projeto.
3. Regra de Negócio Condicional: Uma regra que assume que todo projeto finalizado teve um custo.
   - DC_3: $\forall t \in$ Projetos, $\neg(t.\text{Status} = \text{'Concluído'} \wedge t.\text{Orcamento} = 0.00)$
   - *Análise da Violação:* O registro com `ID_Projeto = 104` viola esta DC. Os predicados t.Status = 'Concluído' e t.Orcamento = 0.00 são ambos verdadeiros. A violação aponta para uma inconsistência: ou o status do projeto está incorreto, ou o orçamento não foi devidamente registrado.

2.3 O SISTEMA FACET PARA DETECÇÃO DE ERROS

O FACET (*FAst Constraint-based Error DeTector*) é um sistema de alto desempenho projetado especificamente para a tarefa de detecção de erros em dados, que se manifestam como violações de DCs (Pena et al., 2022). O sistema foi desenvolvido para superar as limitações de desempenho e consumo de memória de abordagens anteriores, que frequentemente se mostravam ineficientes ao lidar com as consultas complexas, especialmente as junções não-equi, necessárias para validar as DCs em grandes volumes de dados.

Para alcançar sua eficiência, o FACET introduz um conjunto de inovações que otimizam cada etapa do processo de detecção de violações. As contribuições do sistema podem ser compreendidas através de suas principais estratégias de otimização.

2.3.1 Planejamento de Execução Baseado em Esboços de Coluna

Uma das principais inovações do FACET é a sua abordagem para o planejamento da ordem de avaliação dos predicados. Diferentemente de sistemas anteriores que dependiam de amostras dos dados para estimar a seletividade de um predicado, o FACET utiliza esboços de coluna (*column sketches*), como o HyperLogLog, para obter estimativas de cardinalidade muito mais precisas. Essa precisão permite ao sistema construir um *pipeline* de execução mais robusto, priorizando os predicados mais seletivos (aqueles que mais filtram os dados) e mitigando a escolha

de planos de avaliação sub-ótimos. A estratégia consiste em avaliar primeiro os predicados de igualdade, seguidos pelos de desigualdade e, por fim, os de não-igualdade, ordenando-os internamente com base na cardinalidade das colunas para otimizar o uso de memória (Pena et al., 2022).

2.3.2 Algoritmos de Refinamento e Estruturas de Dados Adaptativas

O núcleo do FACET é um pipeline de refinamentos, que são operadores lógicos especializados para avaliar eficientemente cada tipo de predicado. Para os predicados de desigualdade (como < ou >), que são computacionalmente intensivos, o sistema escolhe dinamicamente entre três algoritmos distintos: IEJoin, Hash-Sort-Merge (HSM) e um novo algoritmo proposto pelos autores, o Binning-Hash-Sort-Merge (BHSM). A escolha é baseada nas características dos dados, como a cardinalidade das colunas, garantindo que a abordagem mais eficiente seja sempre utilizada (Pena et al., 2022).

Complementarmente, o FACET emprega estruturas de dados híbridas para armazenar os resultados intermediários. Em vez de uma representação fixa, o sistema adapta a estrutura conforme a necessidade da operação: para refinamentos de igualdade, que apenas necessitam armazenar e ler identificadores de tuplas (*tids*), ele utiliza arrays de inteiros simples; para operações que exigem muitas uniões ou diferenças de conjuntos, como nos refinamentos de desigualdade, ele utiliza *bitmaps* compactados para se beneficiar de operações *bitwise* de alta velocidade (Pena et al., 2022).

2.3.3 Otimização para Múltiplas Restrições

O FACET também é projetado para otimizar a detecção de erros quando múltiplas DCs são avaliadas simultaneamente. Ele organiza os refinamentos de diferentes DCs em uma estrutura de árvore (*trie*), o que permite compartilhar e reutilizar os resultados de predicados comuns a mais de uma restrição. Adicionalmente, o sistema pode executar as árvores de predicados de forma paralela, acelerando a detecção mesmo para DCs que não compartilham predicados, aproveitando assim os recursos de hardware modernos (Pena et al., 2022).

2.4 LACUNAS NA VISUALIZAÇÃO DE QUALIDADE DE DADOS

Enquanto a detecção de erros via algoritmos avançou significativamente com sistemas como o FACET, a visualização desses erros não acompanhou o mesmo ritmo. Ferramentas tradicionais de *Data Profiling* focam majoritariamente em estatísticas de coluna única (histogramas, contagem de nulos, valores mínimos e máximos) (Kandel et al., 2012).

No entanto, violações de Restrições de Negação são inerentemente relacionais e complexas: um erro não é apenas um valor atípico, mas uma contradição lógica entre dois ou mais registros baseada em múltiplos atributos. Ferramentas existentes de limpeza de dados, como o *Metanome*, são excelentes para execução de algoritmos de descoberta e detecção, mas oferecem interfaces limitadas para a exploração interativa dos dados brutos que constituem as violações. Geralmente, o resultado é apresentado em listas estáticas ou arquivos de log, transferindo para o analista o ônus de consultar o banco de dados manualmente para entender o contexto do erro.

Essa carência de ferramentas visuais específicas para DCs justifica a necessidade de uma solução que integre a eficiência da detecção com técnicas de amostragem e sumarização estatística, permitindo ao usuário navegar do “macro” (estatísticas gerais) para o “micro” (o dado da tupla).

2.5 MEDIDAS DE POSIÇÃO E DISPERSÃO: MÉDIA E QUARTIS

A análise dos resultados experimentais e a interface de visualização proposta neste trabalho utilizam medidas estatísticas fundamentais para sumarizar o comportamento das violações de dados. Para compreender a distribuição dos erros — identificando se são sistêmicos ou pontuais — são empregadas medidas de posição (Média e Mediana) e medidas de dispersão baseadas em Quartis.

2.5.1 A Média Aritmética e suas Limitações

A média aritmética simples ($\bar{x}$) é utilizada para informar a posição típica dos dados, atuando como o "centro de gravidade" da distribuição (Farias, 2020). No contexto deste trabalho, ela representa o número médio de violações por tupla. Matematicamente, para um conjunto de n contagens de violações $(x_1, x_2, \ldots, x_n)$, a média é dada por:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

Apesar de sua ampla utilização, a média apresenta uma limitação crítica para a análise de erros: sua alta sensibilidade a valores discrepantes (*outliers*). Conforme destacado por Farias (2020), poucas tuplas com um número massivo de violações podem elevar a média, distorcendo a percepção da qualidade geral dos dados. Por isso, ela é apresentada na ferramenta em conjunto com medidas mais robustas, como os quartis.

2.5.2 Os Quartis para Análise de Distribuição

Para mitigar a sensibilidade da média e permitir a identificação de padrões de erro (como assimetria), utilizam-se os quartis. Eles são medidas que dividem o conjunto de dados ordenado em quatro partes iguais (Montgomery e Runger, 2018):

- **Primeiro Quartil (Q_1):** Valor abaixo do qual encontram-se 25% das tuplas com menos violações.
- **Segundo Quartil (Q_2 - Mediana):** Divide o conjunto de dados ao meio. Diferente da média, a mediana é uma medida robusta e não é afetada por *outliers*, indicando o cenário central real dos erros.
- **Terceiro Quartil (Q_3):** Valor abaixo do qual estão 75% das observações.

A visualização dessas medidas é essencial para o diagnóstico da causa raiz dos problemas de dados. A relação entre a Mediana e os Quartis permite inferir a **dispersão** e a **assimetria** da distribuição (Tukey, 1977):

1. **Dispersão:** A Amplitude Interquartil ($AIQ = Q_3 - Q_1$) indica a variabilidade dos 50% centrais dos dados. Uma AIQ baixa sugere que a maioria das tuplas possui uma quantidade similar de erros.
2. **Assimetria e Outliers:** Comparando a distância entre a Mediana, o Q_3 e o valor Máximo, é possível identificar se o problema de dados é generalizado ou se está concentrado em uma "cauda longa"de tuplas problemáticas.

Na ferramenta desenvolvida, essas métricas fundamentam o painel de estatísticas e validam a necessidade do *ranking* "Top 10", permitindo ao analista focar nas tuplas que extrapolam significativamente o comportamento do terceiro quartil.

2.6 AMOSTRAGEM POR RESERVATÓRIO (RESERVOIR SAMPLING)

A Amostragem por Reservatório (em inglês, *Reservoir Sampling*) é uma técnica projetada para selecionar uma amostra aleatória uniforme de tamanho fixo, sem reposição, a partir de um fluxo de dados de tamanho desconhecido. Este método é essencial para cenários que exigem processamento em uma única passagem sem conhecimento prévio do volume total de dados (Lee et al., 2007).

2.6.1 Funcionamento do Algoritmo

O algoritmo opera mantendo um "reservatório" com capacidade para k itens. Conforme detalhado por Kepe et al. (2015), o processo ocorre em duas etapas lógicas:

1. **Preenchimento Inicial:** Os primeiros k elementos do fluxo são armazenados diretamente no reservatório.
2. **Substituição Probabilística:** Para cada i-ésimo elemento subsequente (onde $i > k$), a inclusão no reservatório ocorre com probabilidade $\frac{k}{i}$. Se o elemento for selecionado, ele substitui aleatoriamente um dos itens já presentes.

Esse mecanismo assegura que, a qualquer momento do processamento, cada elemento visto até então tenha tido a mesma probabilidade (k/n) de compor a amostra final, garantindo a uniformidade estatística (Lee et al., 2007).

2.6.2 Aplicação na Visualização de Violações

Neste trabalho, a aplicação do *Reservoir Sampling* é estratégica para viabilizar a análise humana sobre os resultados do FACET. A técnica é fundamental para obter estimativas consistentes e imparciais (*unbiased estimates*) da população de erros (Lee et al., 2007).

Ao garantir que a amostra contenha violações de todo o fluxo de dados — e não apenas das primeiras linhas processadas — a ferramenta permite que o analista visualize padrões de erro representativos de todo o *dataset*, mantendo o consumo de memória constante e baixo, independentemente do número total de violações detectadas.

3 VISUALIZADOR DE VIOLAÇÕES DE RESTRIÇÕES DE NEGAÇÃO

Conforme discutido na Fundamentação, a detecção de erros em grandes volumes de dados avançou significativamente com algoritmos de alta performance como o FACET, capaz de identificar milhões de violações de DCs em segundos (Pena et al., 2022). No entanto, a eficiência computacional na *contagem* de erros criou um descompasso com a capacidade humana de *análise*.

O resultado padrão de algoritmos, como no FACET, é, tipicamente, um log contendo a contagem total de violações ou uma lista exaustiva de identificadores de tuplas (*tids*). Embora suficiente para métricas de qualidade globais, esse formato é inadequado para o diagnóstico de causas raiz. O analista de dados se vê diante de um "caixa-preta": ele sabe que existem inconsistências, mas não possui ferramentas para explorar *quais* dados as geraram, se o erro é sistêmico (afetando todo o dataset) ou pontual (*outliers*), e qual a severidade do problema.

A literatura de limpeza de dados enfatiza que a detecção é apenas a primeira etapa de um processo iterativo, onde a inspeção visual e o julgamento humano são insubstituíveis (Chu e Ilyas, 2016; Kandel et al., 2012). Este capítulo apresenta a proposta deste trabalho para preencher essa lacuna: uma plataforma de visualização que integra a detecção otimizada do FACET com técnicas de amostragem e estatística descritiva.

3.1 ESTADO DA ARTE E TRABALHOS RELACIONADOS

Para situar a contribuição deste trabalho, é necessário analisar como ferramentas atuais lidam com a apresentação de erros de dados. As abordagens existentes podem ser categorizadas em três grupos principais: Ferramentas de Perfilamento (*Data Profiling*), Ferramentas de ETL/Limpeza e Protótipos de Pesquisa.

3.1.1 Limitações das Ferramentas de Perfilamento

Ferramentas de perfilamento de dados (como *pandas-profiling* ou funcionalidades nativas de BI) são excelentes para gerar estatísticas de coluna única (histogramas, contagem de nulos, valores mínimos e máximos) (Kandel et al., 2012). No entanto, elas falham ao lidar com Restrições de Negação. Como visto no Capítulo 2, uma violação de DC é inerentemente **relacional**: ela não depende do valor de uma célula isolada, mas da contradição lógica entre atributos de duas ou mais tuplas (t_1 e t_2). Um histograma simples não consegue capturar a relação "O voo de ida tem distância X, mas o voo de volta tem distância Y", tornando essas ferramentas ineficazes para os cenários complexos abordados pelo FACET.

3.1.2 Ferramentas de Pesquisa e Metanome

No meio acadêmico, a plataforma *Metanome* é a referência para execução de algoritmos de descoberta e detecção de dependências (Papenbrock et al., 2015). Embora suporte a execução de algoritmos de DCs, seu foco é a avaliação de desempenho (tempo de execução e memória). A saída visual é geralmente limitada a tabelas estáticas ou listas de dependências descobertas, sem funcionalidades de exploração interativa dos dados brutos que violam essas regras. O ônus de investigar o dado recai sobre o usuário, que precisa realizar consultas manuais (SQL) baseadas nos resultados do algoritmo.

3.1.3 A Lacuna de Visualização

Não existem, até o momento, ferramentas integradas que combinem a velocidade de detecção de DCs (como o FACET) com uma interface de diagnóstico visual. A lacuna principal reside na dificuldade de apresentar milhões de pares de erros de forma digerível. A simples listagem é inviável cognitivamente e computacionalmente. A proposta deste trabalho diferencia-se por aplicar técnicas de **sumarização estatística** (quartis e rankings, fundamentados na Seção 2.4) e **amostragem probabilística** (Reservoir Sampling, fundamentado na Seção 2.5) para criar uma representação visual que seja, ao mesmo tempo, leve para o sistema e informativa para o analista.

3.2 PERGUNTAS DE PESQUISA

Diante do cenário exposto, onde existe alta capacidade de detecção mas baixa capacidade de interpretação, este trabalho é guiado pelas seguintes Perguntas de Pesquisa:

Pergunta de Pesquisa 1: Quais modificações algorítmicas são necessárias para estender o *pipeline* do FACET, permitindo não apenas a materialização dos dados de tuplas violadoras, mas também a agregação de metadados estatísticos, sem comprometer seu desempenho?

Pergunta de Pesquisa 2: Como processar o conjunto de violações materializadas para gerar *insights* analíticos escaláveis, utilizando técnicas de amostragem para contornar limitações de memória e transferência de dados?

Pergunta de Pesquisa 3: De que maneira uma interface de visualização interativa pode integrar estatísticas agregadas (média, quartis) e exploração detalhada de amostras para guiar eficientemente o analista na compreensão da magnitude e da causa raiz dos erros?

3.3 PROPOSTA DE TRABALHO

O objetivo deste trabalho é projetar e desenvolver uma solução de software que transforma o FACET de uma ferramenta de "contagem de erros" em uma plataforma de "análise de qualidade". A solução aborda o problema de escalabilidade visual: como mostrar milhões de erros sem travar o navegador e sem sobrecarregar o usuário.

A metodologia para atingir este objetivo divide-se em três pilares, conectando os conceitos teóricos vistos na Fundamentação com a implementação prática:

1. **Instrumentação do Backend (Backend-Driven Statistics):** Em vez de enviar todos os erros para o cliente, o motor de detecção é modificado para calcular estatísticas *in-memory* durante a execução. Isso utiliza os conceitos de *Hash Maps* para contagem de frequência (ranking de tuplas problemáticas) e garante que o processamento pesado ocorra próximo aos dados.

2. **Amostragem Inteligente (Representative Sampling):** Para permitir a inspeção qualitativa, implementa-se o algoritmo *Reservoir Sampling* (Seção 2.5). Isso garante que o usuário receba uma amostra estatisticamente representativa das violações de todo o dataset, e não apenas das primeiras linhas do arquivo, resolvendo o problema de viés de leitura.

3. **Visualização Diagnóstica (Interactive Dashboard):** O desenvolvimento de uma interface que traduz os números abstratos em diagnósticos visuais. A interface utiliza as medidas de dispersão (quartis e outliers, Seção 2.4) para indicar se o erro é uma anomalia isolada ou um problema sistêmico, permitindo que o analista priorize suas ações de limpeza.

Essa abordagem integrada visa validar a hipótese de que, ao fornecer contexto (amostras) e sumarização (estatísticas) instantaneamente após a detecção, reduz-se drasticamente o tempo necessário para entender e corrigir problemas de qualidade de dados complexos (Pena et al., 2022).

4 ALTERAÇÃO DO FACET

Para superar o desafio de visualização de violações de Restrição de Negação, foi implementado uma estratégia de coleta de dados em tempo real que opera simultaneamente ao processo de detecção. Esta abordagem não armazena todas as violações, mas sim duas formas de informação compacta e de alto valor analítico:

1. Amostragem Probabilística: Uma amostra representativa e de tamanho fixo dos pares violadores, que permite a inspeção qualitativa dos tipos de erro.

2. Estatísticas Agregadas: Contadores que rastreiam a frequência de violações por tupla, permitindo identificar os registros mais problemáticos.

4.1 AMOSTRAGEM PROBABILÍSTICA VIA RESERVOIR SAMPLING

O núcleo da estratégia de amostragem é o algoritmo *Reservoir Sampling*. Esta técnica é ideal para cenários de *streaming*, onde o volume total de dados (neste caso, o número de violações) é desconhecido a priori. O algoritmo garante que, ao final do processo, cada par violador detectado tenha tido a mesma probabilidade de ser incluído na amostra final, tornando-a estatisticamente imparcial e representativa.

No sistema FACET, este processo foi adaptado para lidar com o fluxo de pares de tuplas que violam restrições de negação. O funcionamento do algoritmo implementado pode ser descrito da seguinte forma:

1. Um "reservatório" de amostras com um tamanho máximo fixo k (definido como 100 no sistema) é inicializado.

2. Os primeiros k pares violadores detectados são diretamente armazenados no reservatório.

3. Para cada violação subsequente i (onde $i > k$), o sistema decide se deve incluí-la na amostra com uma probabilidade de k/i.

4. Se a decisão for positiva, um dos k pares existentes no reservatório é substituído, de forma aleatória e uniforme, pelo novo par violador.

Essa lógica teórica é traduzida diretamente na implementação de código deste trabalho. O tamanho do reservatório k corresponde à constante `MAX_SAMPLES` (100), enquanto o número de violações vistas i é representado pela variável `counter`. O código primeiro verifica a fase de preenchimento (passo 2) através da condição `violationSamples.size() < MAX_SAMPLES`. Após o reservatório estar cheio, a decisão probabilística k/i (passo 3) é simulada computacionalmente através do teste `rand.nextDouble() < (double) MAX_SAMPLES / counter`. A função `rand.nextDouble()` gera um número aleatório uniforme entre 0.0 e 1.0; ao compará-lo com o resultado de k/i, o código garante que o teste tenha exatamente a probabilidade k/i de ser verdadeiro. Caso o teste seja positivo, a substituição aleatória (passo 4) é realizada selecionando um índice com `rand.nextInt(MAX_SAMPLES)` e atualizando essa posição no reservatório.

Conforme o número de violações i (ou `counter`) aumenta, a probabilidade de substituição diminui, mas nunca chega a zero, garantindo que violações detectadas em qualquer estágio do processo tenham chance de serem amostradas.

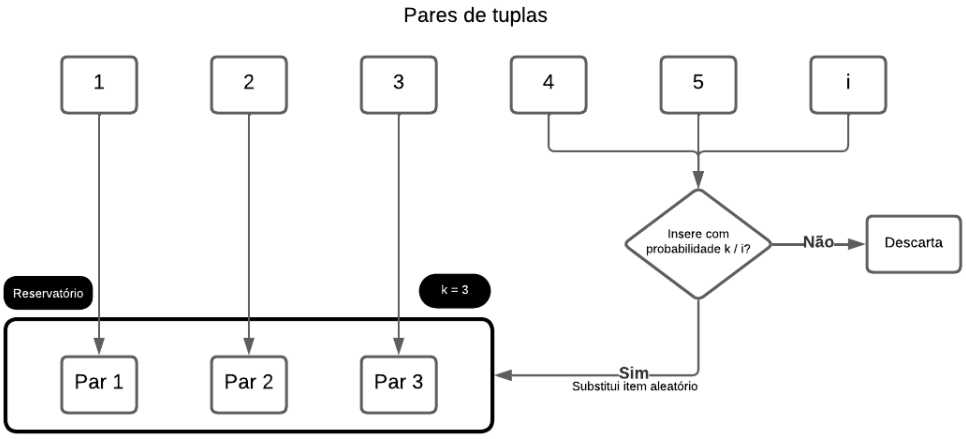


Figura 4.1: Esquema ilustrativo do algoritmo de *Reservoir Sampling* utilizado no sistema FACET.

A Figura 4.1 exemplifica o funcionamento do algoritmo no contexto do FACET. Observa-se que os três primeiros pares de tuplas ($k = 3$) são inseridos diretamente no reservatório, enquanto os elementos seguintes passam por uma decisão probabilística. Cada novo par de violação i é avaliada quanto à sua inclusão com probabilidade k/i; caso seja selecionado, ele substitui aleatoriamente um dos pares existentess; caso não seja selecionado, o par é descartado, mantendo assim a aleatoriedade e o tamanho fixo do reservatório durante todo o processamento do fluxo de dado.

4.2 COLETA E ANÁLISE DE ESTATÍSTICAS AGREGADAS

4.2.1 Coleta de Dados em Tempo Real

Para viabilizar a análise quantitativa dos erros sem armazenar todos os pares violadores, o FACET adota uma abordagem de coleta de estatísticas em tempo real. O pilar desta estratégia é uma estrutura de dados central: um *Hash Map* que mapeia o identificador de cada tupla (*tid*) para um contador de suas violações.

A estrutura é definida como `Map<Integer, Long>`, onde a chave representa o `tid` e o valor armazena a quantidade de vezes que a tupla participou de uma violação. Este processo incremental assegura que, ao final da detecção, o sistema disponha de um sumário completo da frequência de erros por tupla, com um custo computacional e de memória significativamente inferior ao armazenamento bruto das violações. A memória necessária é da ordem de $O(U)$, onde U é o número de tuplas *únicas* envolvidas em violações.

4.2.2 Cálculo de Métricas Descritivas

Após a conclusão do pipeline de detecção e o preenchimento completo do mapa de contadores, o sistema realiza uma única etapa de cálculo para gerar as métricas estatísticas que serão apresentadas ao analista. Este processo é dividido em duas análises principais.

4.2.2.1 Ranking das Tuplas Mais Problemáticas

Identificar os registros que mais contribuem para a inconsistência dos dados é um passo fundamental para priorizar os esforços de correção. Para isso, o sistema implementa um

algoritmo de ranking para determinar as "top K" tuplas mais problemáticas (com K fixado em 10, por padrão). O processo é executado utilizando a API de Streams do Java:

1. As entradas do *Hash Map* (pares de `tid` e contagem) são convertidas em um *stream*.
2. O *stream* é ordenado em ordem decrescente, utilizando a contagem de violações como critério principal.
3. A operação `limit(K)` é aplicada para truncar o resultado, retendo apenas as K tuplas com as maiores contagens.

A complexidade computacional desta operação é dominada pela ordenação, resultando em $O(U \log U)$, onde U é o número de tuplas únicas com violações.

4.2.2.2 Análise da Distribuição de Violações

Para fornecer uma visão macroscópica da distribuição dos erros, o sistema calcula um conjunto de estatísticas descritivas, incluindo quartis e a média de violações por tupla. Primeiramente, apenas os valores (as contagens de violações) do *Hash Map* são extraídos e armazenados em uma lista, que é então ordenada em ordem crescente. A partir desta lista ordenada de U contagens, as métricas são calculadas da seguinte forma:

- Mínimo e Máximo: Correspondem ao primeiro e ao último elemento da lista ordenada.
- Média Aritmética: Calculada pela soma de todos os valores da lista dividida por U.
- Quartis: O sistema utiliza um método de cálculo baseado em índices para estimar os quartis:
  - Primeiro Quartil (Q1 - 25%): Valor no índice $U/4$ da lista ordenada.
  - Mediana (Q2 - 50%): Valor no índice $U/2$.
  - Terceiro Quartil (Q3 - 75%): Valor no índice $3U/4$.

Assim como no ranking, a complexidade desta etapa é de $O(U \log U)$ devido à necessidade de ordenação prévia dos dados.

4.3 INTEGRAÇÃO, EFICIÊNCIA E REPRODUTIBILIDADE

4.3.1 Integração ao Pipeline de Detecção

As estratégias de amostragem e coleta de estatísticas não são uma etapa de pós-processamento, mas sim uma operação integrada ao fluxo de detecção do FACET. O processo é orquestrado da seguinte forma:

1. O algoritmo principal do FACET processa os dados e gera partições contendo *clusters* de tuplas que violam uma determinada DC.
2. A classe `TPViolationCounter` itera sobre os pares de identificadores de tuplas (*tids*) dentro de cada partição.
3. Para cada par violador identificado, o método `addViolatingPair` é invocado. Esta função executa as duas ações — Atualização de Estatísticas no *Hash Map* e Execução da Amostragem no reservatório simultaneamente.

Esta integração garante que a coleta de dados ocorra de maneira incremental e com sobrecarga computacional mínima, sem a necessidade de uma segunda passagem sobre os resultados.

4.3.2 Garantias de Eficiência e Reprodutibilidade

A principal vantagem da abordagem reside em sua eficiência de memória. O uso de memória para a amostragem é constante, $O(1)$, devido ao reservatório de tamanho fixo ($k = 100$). No que tange à complexidade de tempo, cada par violador é processado em tempo constante, $O(1)$. Consequentemente, a complexidade de tempo total da etapa de coleta é linear em relação ao número de violações detectadas, V, resultando em uma complexidade de $O(V)$. A sobrecarga computacional do nosso módulo permanece estritamente ligada ao resultado do algoritmo de detecção e não introduz complexidade adicional.

Para garantir que as análises possam ser reproduzidas, o gerador de números aleatórios utilizado no algoritmo de *Reservoir Sampling* é inicializado com uma semente fixa (*seed*). Isso assegura que, para o mesmo conjunto de dados e a mesma DC, a amostra de violações gerada será idêntica em múltiplas execuções, uma característica essencial para a validação de resultados em um contexto de pesquisa.

Ao final da execução, as estatísticas agregadas e a amostra de violações são persistidas em um formato estruturado (JSON), prontas para serem consumidas por ferramentas de visualização e análise.

5 INTERFACE WEB DE ANÁLISE

Para resolver a lacuna entre a detecção de violações e a análise de erros, foi projetada e implementada uma solução de software com uma arquitetura cliente-servidor, composta por um *front-end* interativo e um *back-end* de processamento. A Figura 5.1 apresenta uma visão geral do fluxo de dados da aplicação, desde a submissão dos arquivos pelo usuário até a apresentação dos resultados.

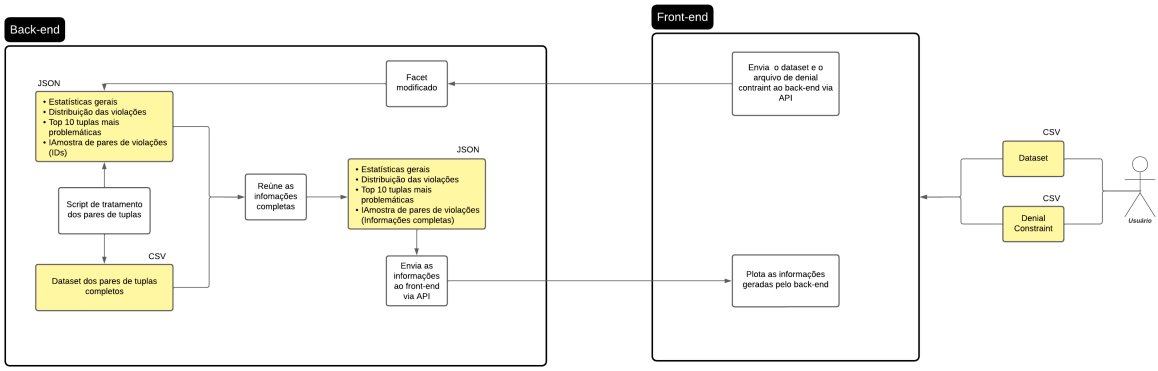


Figura 5.1: Fluxo de dados da aplicação, detalhando a interação entre o usuário, o front-end e os componentes do back-end.

O processo, conforme ilustrado, inicia-se no front-end, onde o usuário submete o *dataset* e as regras de *Denial Constraint*. Esses arquivos são enviados ao back-end, que orquestra a análise: primeiro, o FACET modificado gera um JSON com estatísticas e IDs de violações; em seguida, um script de tratamento utiliza esses IDs para buscar os dados completos das tuplas no *dataset* original. Por fim, o *back-end* reúne todas as informações em um JSON enriquecido e o envia de volta ao *front-end*, que exibe os resultados de forma visual para o usuário.

Este capítulo detalha a arquitetura e o funcionamento de cada um dos componentes envolvidos neste fluxo.

5.1 MÓDULO DE ENRIQUECIMENTO DE AMOSTRAS

O resultado primário do FACET consiste em estatísticas agregadas e uma amostra de pares de identificadores de tuplas (*tids*) que violam as Restrições de Negação. Esses identificadores, por si sós, não oferecem contexto sobre o conteúdo dos erros. Para viabilizar a inspeção humana, foi desenvolvido um módulo em Python, o `export_violation_samples.py`, cuja função é enriquecer essas amostras.

O script recebe como entrada o arquivo JSON gerado pelo FACET e o arquivo CSV original que serviu de base para a análise. Sua principal tarefa é mapear os *tids* presentes nas amostras de violação de volta para as linhas completas correspondentes no arquivo de dados. O resultado é um novo arquivo de dados onde cada registro representa uma violação, contendo o identificador da amostra, os *tids* do par violador e os dados completos de ambas as tuplas, com as colunas duplicadas e prefixadas (e.g., `tuple_1_<nome_coluna>` e `tuple_2_<nome_coluna>`).

Para garantir a eficiência em datasets de grande volume, o script não carrega o arquivo CSV inteiro na memória. Em vez disso, ele primeiro extrai do JSON o conjunto de todos os *tids*

únicos que precisa recuperar. Em seguida, ele realiza uma única passagem de leitura (*streaming*) sobre o arquivo CSV, armazenando em memória apenas as linhas cujos índices correspondem aos *tids* necessários. Essa abordagem minimiza o consumo de memória e otimiza o tempo de processamento.

5.2 ARQUITETURA DO SISTEMA WEB DE VISUALIZAÇÃO

O segundo componente da solução é uma aplicação web de arquitetura cliente-servidor, projetada para oferecer uma interface interativa para a execução do FACET e a visualização dos resultados enriquecidos.

5.2.1 Backend: API de Análise e Orquestração

O backend foi desenvolvido para servir como o ponto central de processamento. Sua principal responsabilidade é expor um endpoint de API, `/api/analyze`, que orquestra todo o fluxo de análise de dados.

Ao receber uma requisição, o backend executa as seguintes etapas:

1. **Recebimento de Dados:** A API aceita requisições `POST` do tipo `multipart/form-data`, permitindo que o usuário envie os dois arquivos necessários: o dataset em formato CSV e as Restrições de Negação, também em CSV.

2. **Execução da Detecção:** Os arquivos são passados para o motor do FACET (modificado conforme descrito nos capítulos anteriores), que realiza a detecção de violações e gera o arquivo de resultados JSON contendo as estatísticas e as amostras de *tids*.

3. **Enriquecimento das Amostras:** O backend invoca a lógica do módulo de enriquecimento, utilizando o JSON recém-gerado e o CSV original. Este passo materializa os dados completos para cada par de *tids* na amostra.

4. **Retorno da Resposta:** Por fim, a API consolida todos os resultados — estatísticas, ranking de tuplas problemáticas e as amostras já enriquecidas com os dados completos — em um único objeto JSON e o retorna como resposta à requisição do cliente.

A Listagem 5.2 ilustra a estrutura do arquivo JSON intermediário gerado na Etapa 2. Neste exemplo simplificado, foram detectadas exatamente duas violações envolvendo três tuplas distintas (a tupla 101 viola a regra tanto com a 102 quanto com a 103), o que se reflete consistentemente nas estatísticas e nas amostras.

5.2.2 Frontend: Interface de Visualização Interativa

O frontend é uma aplicação desenvolvida com a biblioteca React e o ambiente de construção Vite. A navegação entre as páginas é gerenciada pelo React Router, e a interface do usuário é construída com componentes da biblioteca Material-UI.

A aplicação consiste em duas visualizações principais:

5.2.2.1 Página de Upload

Esta é a tela inicial da aplicação. Ela apresenta uma interface minimalista com dois campos de entrada de arquivo, um para o dataset e outro para as DCs, e um botão para submeter a análise. A validação nativa do navegador (`required`) impede o envio de formulários

```
{
  "timestamp": "2025-11-04 10:00:00",
  "total_violations": 2,
  "output_time_ms": 15,
  "statistics": {
    "total_tuples_involved": 3,
    "violations_distribution": {
      "min": 1, "q1": 1, "median": 1,
      "q3": 2, "max": 2, "average": 1.33
    },
    "top_problematic_tuples": [
      { "tuple_id": 101, "violations_count": 2 },
      { "tuple_id": 102, "violations_count": 1 },
      { "tuple_id": 103, "violations_count": 1 }
    ],
    "violation_samples": [
      { "sample_id": 1, "tuple_1": "101", "tuple_2": "102" },
      { "sample_id": 2, "tuple_1": "101", "tuple_2": "103" }
    ]
  }
}
```


Figura 5.2: Exemplo simplificado do JSON intermediário gerado pelo FACET.

incompletos. Durante o processamento, o botão de submissão é desabilitado e seu texto é alterado para fornecer feedback de carregamento ao usuário. Em caso de sucesso, a aplicação armazena os resultados retornados pela API em seu estado global e navega automaticamente para a página de resultados.

5.2.2.2 Página de Resultados

Esta página é responsável por apresentar os dados da análise de forma clara e organizada. Ela recebe os resultados do backend e os renderiza em três seções principais:

- Painel de Estatísticas: Uma grade de cartões (*cards*) exibe as métricas chave:
  - *Estatísticas Gerais:* O total de violações e o número de tuplas únicas envolvidas.
  - *Distribuição de Violações:* Um resumo estatístico com os valores de mínimo, máximo, média e os quartis (Q1, Mediana, Q3) da quantidade de violações por tupla.
  - *Top 10 Tuplas Mais Problemáticas:* Uma tabela que classifica as dez tuplas que mais participaram de violações, exibindo o `tuple_id` e a respectiva contagem.
- Visualização das Amostras Enriquecidas: A seção mais importante para a análise qualitativa. O sistema itera sobre a lista de amostras enriquecidas. Para cada violação, é renderizado um cartão contendo uma tabela comparativa. As colunas desta tabela são geradas dinamicamente a partir dos atributos (chaves) dos dados da tupla, tornando a interface adaptável a qualquer esquema de dataset. As duas linhas da tabela exibem os

valores completos da "Tupla 1"e da "Tupla 2", permitindo que o analista compare lado a lado os registros que constituem a violação.

5.3 LIMITAÇÕES E ANÁLISE DE ESCALABILIDADE

Embora a arquitetura proposta resolva o problema da visualização de grandes volumes de erros através da amostragem estatística, a implementação atual apresenta limitações técnicas quando submetida a cenários de extrema escala (na ordem de bilhões de violações ou arquivos de múltiplos Gigabytes). As limitações podem ser categorizadas em latência de processamento e gerenciamento de memória.

5.3.1 Latência de Processamento (Backend)

A limitação mais perceptível para o usuário final em grandes datasets é o tempo de espera entre o envio do arquivo e a visualização dos resultados. O tempo total de resposta da API é composto por duas etapas intensivas:

1. **Custo Computacional da Detecção:** Mesmo com as otimizações do FACET, a detecção de bilhões de violações exige um tempo de CPU considerável. O algoritmo precisa iterar sobre milhões de pares de tuplas para contabilizar as estatísticas e preencher o reservatório de amostras.

2. **Gargalo de I/O no Enriquecimento:** O módulo de enriquecimento precisa ler o arquivo CSV original para buscar os dados das tuplas amostradas. Em arquivos muito grandes (vários GBs), a velocidade de leitura do disco (*Disk I/O*) torna-se um gargalo, pois o script precisa percorrer o arquivo até encontrar as linhas desejadas, mesmo operando em modo de *streaming*.

Consequentemente, em experimentos com bilhões de violações, observou-se que a interface permanece em estado de "carregamento"por um período prolongado, dependendo diretamente da capacidade de hardware do servidor.

5.3.2 Limitações de Renderização (Frontend)

A interface web adota uma estratégia de *Client-Side Rendering* (renderização no lado do cliente), onde todo o objeto JSON retornado pela API é carregado na memória do navegador.

Isso impõe um limite prático ao tamanho da amostra (K). Se o sistema fosse configurado para retornar uma amostra muito grande (por exemplo, 10.000 ou 50.000 pares de violação enriquecidos), o volume de dados transferidos e a quantidade de elementos DOM (Document Object Model) criados pelo React poderiam causar lentidão ou travamento da aba do navegador.

A solução atual mitiga isso fixando a amostra em um número gerenciável ($K = 100$), o que garante fluidez na interface independentemente do número total de erros no dataset. Para escalar a visualização para amostras maiores, seria necessária a implementação de paginação no lado do servidor (*Server-Side Pagination*), enviando os dados ao frontend sob demanda, e não em lote único.

6 RESULTADOS EXPERIMENTAIS E ANÁLISE

Este capítulo apresenta a validação experimental da solução proposta. O objetivo dos experimentos foi duplo: (1) avaliar a capacidade da ferramenta de diagnosticar diferentes classes de erros de dados (validando a eficácia da visualização) e (2) verificar o comportamento do sistema sob carga massiva de violações (validando a eficiência da arquitetura).

Os experimentos utilizaram *datasets* públicos amplamente adotados na literatura de perfilamento de dados: `Actorkey`, `Flights` e `Tax`, todos padronizados com um milhão de registros cada, conforme utilizado no trabalho original do FACET (Pena et al., 2022).

6.1 METODOLOGIA E AMBIENTE EXPERIMENTAL

Para garantir a reprodutibilidade dos resultados e isolar variáveis de desempenho, todos os experimentos foram conduzidos em um ambiente controlado.

6.1.1 Configuração do Sistema

A Tabela 6.1 detalha as especificações de hardware e software do ambiente utilizado para a execução do *backend* (motor FACET e API) e do *frontend*.

Tabela 6.1: Configuração do Sistema

Componente	**Especificação**
Processador	Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz
Sistema Operacional	Ubuntu 24.04.1 LTS
Versão do Kernel	6.6.87
Quantidade de RAM	8 GB

6.1.2 Configuração dos Cenários

Foram desenhados quatro cenários distintos, resumidos na Tabela 6.2. Cada cenário aplica uma DC específica para induzir um padrão de erro diferente.

6.2 CENÁRIO 1: DETECÇÃO DE DUPLICATAS E AGRUPAMENTOS (ACTORKEY)

Neste cenário, aplicou-se uma DC que atua como uma verificação de Combinação Única de Colunas nos atributos `title`, `role`, `name` e `charname` do dataset `Actorkey`. O objetivo é verificar como a ferramenta apresenta o problema clássico de duplicação de registros.

A Figura 6.1 apresenta o painel de resultados. As Estatísticas Gerais revelam 33.882 violações envolvendo 7.443 tuplas. A análise visual permite ir além da simples contagem:

- **Padrão de Erro:** Os quartis (Mínimo, Q1 e Mediana) são todos iguais a 2. Isso indica que a unidade básica do erro não é o par, mas sim **tripletos** de registros idênticos (uma tupla viola a regra com outras duas).
- **Severidade:** O Terceiro Quartil (Q3) de 8 e o Máximo de 108 revelam que, embora a maioria sejam tripletos, existem clusters massivos de duplicação.

Tabela 6.2: Configuração dos experimentos realizados

Experimento	Dataset	Nº de Linhas	Restrição de Negação (DC) Aplicada
1	`Actorkey`	1.000.000	$\neg(t_1.title = t_2.title \wedge t_1.role = t_2.role \wedge t_1.name = t_2.name \wedge t_1.charname = t_2.charname)$
2	`Flights`	1.000.000	$\neg(t_1.Origin = t_2.Destination \wedge t_1.Destination = t_2.Origin \wedge t_1.Distance \neq t_2.Distance)$
3	`Flights`	1.000.000	$\neg(t_1.Origin = t_2.Origin \wedge t_1.Destination = t_2.Destination \wedge t_1.Flights > t_2.Flights \wedge t_1.Passengers < t_2.Passengers)$
4	`Tax`	1.000.000	$\neg(t_1.State = t_2.State \wedge t_1.Salary > t_2.Salary \wedge t_1.Rate < t_2.Rate)$

Figura 6.1: Resultados do Cenário 1: Identificação de grupos de duplicatas no dataset `Actorkey`.

- **Ação Corretiva:** O ranking "Top 10" expõe imediatamente os IDs (ex: 719, 1690) que pertencem ao grupo de 109 duplicatas, orientando o analista a começar a limpeza por este cluster crítico.

6.3 CENÁRIO 2: ERROS SISTÊMICOS EM GRANDES VOLUMES (FLIGHTS - DISTÂNCIA)

Este experimento utilizou o dataset `Flights` para verificar a simetria de distâncias em rotas de ida e volta. O desafio aqui não é a complexidade da regra, mas o volume: o erro é sistêmico.

Figura 6.2: Resultados do Cenário 2: Análise de erro sistêmico de distâncias.

A ferramenta processou com sucesso mais de **7 milhões de violações**. As estatísticas agregadas permitiram um diagnóstico rápido da natureza do problema:

- **Concentração:** A média de 434 violações por tupla sugere que os erros estão concentrados em aeroportos específicos (hubs).
- **Teto de Violações:** O ranking mostra múltiplas tuplas atingindo exatamente o mesmo teto de 7.210 violações. Isso indica que o problema não é aleatório, mas provavelmente um erro nos dados de referência de um aeroporto muito movimentado, que afeta todas as suas rotas.

6.4 CENÁRIO 3: INCONSISTÊNCIA LÓGICA GENERALIZADA (FLIGHTS - PASSAGEIROS)

Ainda no dataset `Flights`, testou-se a regra lógica “Mais voos implicam mais passageiros”. Este cenário representa o teste de estresse da ferramenta, gerando quase 60 milhões de erros.

A Figura 6.3 demonstra a capacidade da ferramenta de lidar com a falha total de uma regra de negócio:

- **Escala:** 98,8% das tuplas estão envolvidas em erros. Isso prova ao analista que a regra não é válida para este dataset ou que os dados estão inteiramente corrompidos.
- **Distribuição Assimétrica:** A visualização dos quartis mostra uma forte assimetria à direita (distância entre Mediana e Q3 é grande). Embora o erro seja geral (mediana 83), existe uma "cauda longa"de casos extremos (máximo 2.873), identificados claramente no Top 10.

Figura 6.3: Resultados do Cenário 3: Colapso da regra de negócio em 98% dos dados.

6.5 CENÁRIO 4: IDENTIFICAÇÃO DE OUTLIERS E ANOMALIAS PONTUAIS (TAX)

O último cenário utilizou o dataset `Tax` para encontrar uma anomalia de progressividade fiscal. Este experimento é particularmente ilustrativo para demonstrar a diferença abismal entre a detecção bruta e o diagnóstico visual.

6.5.1 A Limitação da Abordagem Tradicional

Para contextualizar o problema resolvido por este trabalho, a Figura 6.4 apresenta a saída padrão gerada pelo FACET original ao processar este cenário.

```
21:12:50 [main] configurations.FacetConfigurationProvider  - Setting the Hybrid predefined configuration fo
21:12:50 [main] configurations.FacetConfiguration  - Setting PLIScheme to HybridPLI
21:12:50 [main] input.BaseInput  - Reading dataset file: tax_1000000.csv
21:12:50 [main] input.BaseInput  - Reading dc file: tax_dc4.csv
21:12:50 [main] input.BaseInput  - Columns required to handle DCs: [RATE, STATE, SALARY]
21:12:51 [main] input.BaseInput  - Number of rows: 999999
21:12:51 [main] facet.DependencyViolationDetector  - Handling the DC: not(t1.STATE==t2.STATE ...
21:12:51 [main] singleDCPlanners.SequentialPipelinePlanner  - Using a HybridSCBC predicate order planner.
...
21:12:51 [main] facet.DependencyViolationDetector  - Detecting Violations ...
21:12:52 [main] facet.DependencyViolationDetector  - Number of violations : 36
21:12:52 [main] mockers.FacetMocker  - Execution Time (in ms): 855
```

Figura 6.4: Saída do FACET original para o dataset Tax.

Como observado na última linha do log, o sistema é extremamente eficiente, executando a verificação em 855ms. No entanto, o resultado resume-se a uma única métrica: "`Number of violations : 36`".

Para o analista, este resultado é opaco. Não há indicação de quais tuplas estão envolvidas, nem se os 36 erros são casos isolados ou se pertencem a um mesmo grupo. O ônus da investigação recai inteiramente sobre o usuário, que precisaria realizar consultas SQL complexas para tentar reconstruir os pares violadores.

6.5.2 O Diagnóstico Visual Proposto

Em contraste, a ferramenta proposta neste trabalho transforma essa contagem abstrata em diagnóstico acionável. A Figura 6.5 exibe o painel de resultados gerado pela nossa solução para a mesma execução.

Figura 6.5: Resultados do Cenário 4: Identificação precisa de um outlier único (Hub-and-Spoke).

O painel diagnosticou com precisão um padrão *Hub-and-Spoke*:

- **Diagnóstico Visual:** Diferente do log que apenas informa "36", o painel de distribuição e o Ranking Top-10 mostram que uma única tupla (ID 111912) participa de todas as 36 violações (o *hub*), enquanto as outras 36 tuplas participam de apenas uma (os *spokes*).
- **Confirmação pela Amostra:** A inspeção das amostras (Figura 6.6) revela que a Tupla 111912 possui taxa 0.0, confirmando-a como a causa raiz.

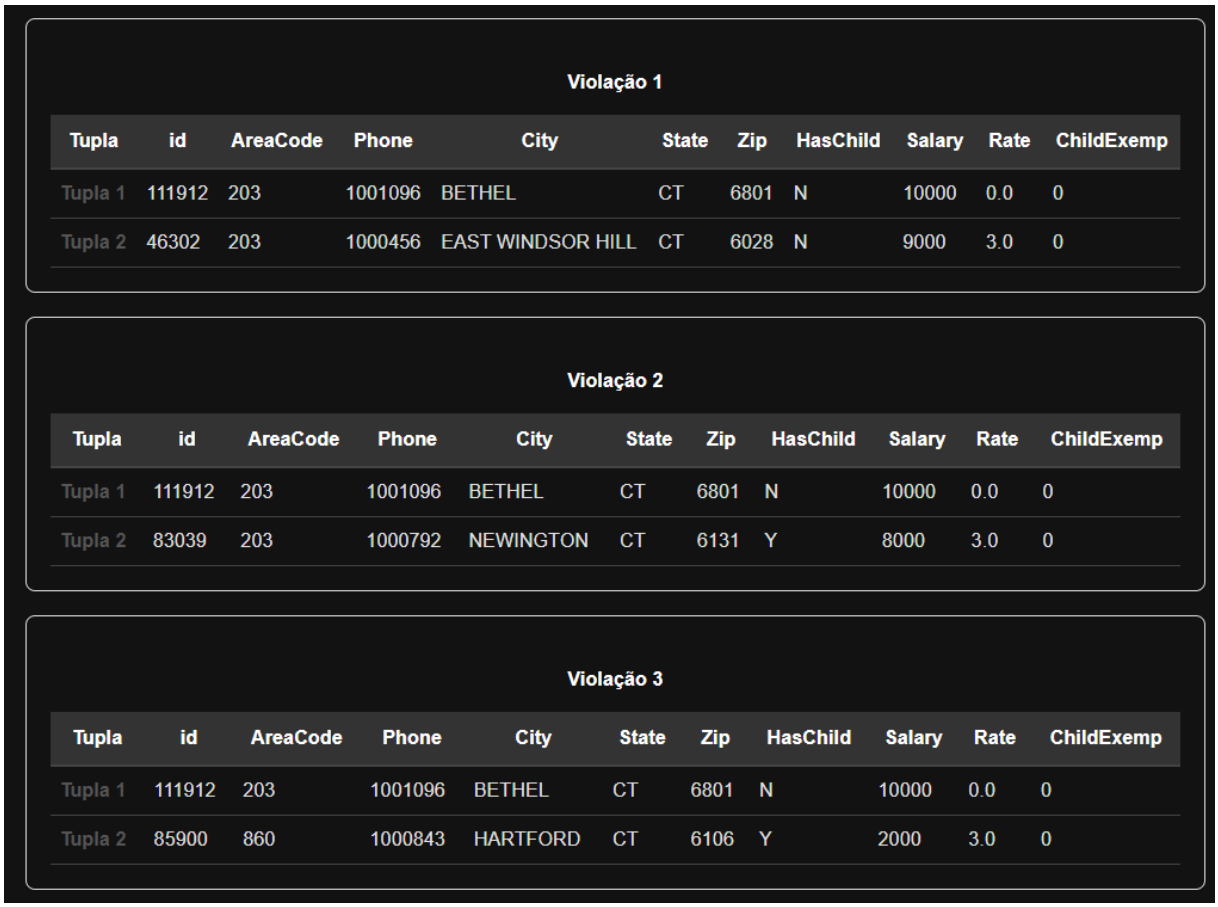
A comparação entre a Figura 6.4 e a Figura 6.5 valida a hipótese deste trabalho: enquanto o FACET original apenas *alerta* sobre a existência de problemas, a ferramenta desenvolvida permite *resolver* o problema, guiando o analista a ignorar 36 erros aparentes para focar na única correção necessária (a tupla 111912).

6.6 VALIDAÇÃO DAS PERGUNTAS DE PESQUISA

Os experimentos realizados fornecem evidências empíricas para responder às Perguntas de Pesquisa formuladas no Capítulo 3.

6.6.1 Resposta à Pergunta de Pesquisa 1 (Modificações e Performance)

Os experimentos 2 e 3, que lidaram com 7 milhões e 60 milhões de violações respectivamente, demonstraram que a arquitetura de coleta de estatísticas *in-memory* (Hash Maps) integrada



Violação 1

Tupla	id	AreaCode	Phone	City	State	Zip	HasChild	Salary	Rate	ChildExemp
Tupla 1	111912	203	1001096	BETHEL	CT	6801	N	10000	0.0	0
Tupla 2	46302	203	1000456	EAST WINDSOR HILL	CT	6028	N	9000	3.0	0

Violação 2

Tupla	id	AreaCode	Phone	City	State	Zip	HasChild	Salary	Rate	ChildExemp
Tupla 1	111912	203	1001096	BETHEL	CT	6801	N	10000	0.0	0
Tupla 2	83039	203	1000792	NEWINGTON	CT	6131	Y	8000	3.0	0

Violação 3

Tupla	id	AreaCode	Phone	City	State	Zip	HasChild	Salary	Rate	ChildExemp
Tupla 1	111912	203	1001096	BETHEL	CT	6801	N	10000	0.0	0
Tupla 2	85900	860	1000843	HARTFORD	CT	6106	Y	2000	3.0	0

Figura 6.6: Amostra das violações do Cenário 4, evidenciando a Tupla 111912 como pivô do erro.

ao FACET é robusta. O sistema foi capaz de agregar metadados de milhões de erros sem exaurir a memória do servidor, confirmando que a complexidade $O(1)$ por violação (para amostragem e contagem) tem impacto negligenciável no tempo total de execução quando comparado ao custo quadrático da detecção em si.

6.6.2 Resposta à Pergunta de Pesquisa 2 (Processamento Escalável)

A estratégia de utilizar o algoritmo *Reservoir Sampling* provou-se eficaz. Nos quatro cenários, independentemente de haver 30 erros (Tax) ou 60 milhões (Flights), o *backend* gerou um objeto JSON de tamanho fixo e previsível contendo uma amostra representativa ($k = 100$). Isso valida o modelo de processamento proposto, onde a escalabilidade é garantida pelo descarte probabilístico de dados redundantes no servidor, enviando ao cliente apenas o necessário para a construção dos indicadores visuais.

6.6.3 Resposta à Pergunta de Pesquisa 3 (Eficácia da Visualização)

A eficácia analítica foi demonstrada pela diversidade de diagnósticos obtidos nos quatro cenários. Utilizando a mesma interface:

1. No **Cenário 1**, a distribuição dos quartis revelou agrupamentos (tripletos).
2. No **Cenário 3**, as estatísticas gerais diagnosticaram a falha sistêmica de uma regra de negócio.

3. No **Cenário 4**, o ranking Top-10 isolou um *outlier* único em meio a um milhão de registros.

Esses resultados confirmam que a combinação de estatísticas agregadas (macro) com amostras de dados enriquecidas (micro) oferece o contexto necessário para que o analista compreenda a natureza do erro sem precisar acessar o banco de dados manualmente.

7 CONCLUSÃO

Este trabalho abordou a lacuna crítica existente entre a detecção de erros em alta performance e a interpretação humana desses resultados. Ferramentas como o FACET, embora extremamente eficientes, operam como "caixas pretas" para o analista de dados. A saída nativa do FACET restringe-se a arquivos de log contendo identificadores numéricos de tuplas e contagens brutas de violações. Diante de uma Restrição de Negação abstrata (por exemplo, uma regra lógica complexa envolvendo múltiplos atributos), o usuário se via obrigado a realizar consultas manuais ao banco de dados, copiando IDs de linhas (ex: tupla 1042 e tupla 5091) para tentar reconstruir o contexto do erro.

A solução desenvolvida alterou fundamentalmente esse fluxo de trabalho. Ao substituir a abstração numérica pela visualização contextual, a ferramenta permite que o analista veja as tuplas conflitantes lado a lado, destacando exatamente quais valores quebraram a regra. Essa mudança de paradigma — de "contar erros" para "diagnosticar causas" — valida a hipótese de que a visualização é o elo faltante para tornar algoritmos complexos de limpeza de dados acessíveis e acionáveis.

A integração realizada entre o motor modificado, a API de orquestração e a interface web demonstrou ser uma arquitetura viável para expor a complexidade interna da validação de dados de forma intuitiva. No entanto, a implementação desta arquitetura em cenários de extrema escala revelou desafios técnicos importantes.

7.1 LIMITAÇÕES E TRABALHOS FUTUROS

Embora a arquitetura proposta resolva o problema funcional da visualização, a implementação atual apresenta limitações de desempenho quando submetida a *datasets* na ordem de múltiplos Gigabytes ou bilhões de violações. Essas restrições foram categorizadas em dois vetores principais: latência de processamento no *backend* e gerenciamento de memória no *frontend*.

No que tange à latência, o tempo de resposta da API em grandes volumes é impactado por duas etapas intensivas. Primeiramente, o custo computacional da detecção permanece alto; mesmo com otimizações, iterar sobre milhões de pares de tuplas consome tempo considerável de CPU. Em segundo lugar, identificou-se um gargalo de I/O no módulo de enriquecimento. A leitura de arquivos CSV massivos para recuperar os dados brutos das tuplas amostradas esbarra na velocidade de leitura do disco, resultando em períodos prolongados de espera ("carregamento") para o usuário final.

No lado do cliente (*frontend*), a estratégia de renderização adotada (*Client-Side Rendering*) impõe um limite prático ao tamanho da amostra visualizada. Como todo o objeto JSON retornado é carregado na memória do navegador, tentar renderizar milhares de violações simultaneamente gera uma quantidade excessiva de elementos DOM (*Document Object Model*), o que pode causar lentidão ou travamento da interface. A solução atual mitiga este risco fixando a amostra em um número gerenciável ($K = 100$), garantindo a fluidez da navegação independentemente do tamanho total do erro.

Para trabalhos futuros, a evolução natural deste sistema reside na superação dessas barreiras de escalabilidade. A implementação de paginação no lado do servidor (*Server-Side Pagination*) é imperativa para permitir a visualização de amostras maiores sem sobrecarregar o cliente, enviando dados sob demanda. Além disso, otimizações no acesso a disco (como

indexação prévia dos arquivos CSV) poderiam reduzir drasticamente o gargalo de I/O, tornando a ferramenta robusta para ambientes de Big Data.

REFERÊNCIAS

Chu, X. e Ilyas, I. F. (2016). Data cleaning: Overview and emerging challenges. *SIGMOD Record*, 45(2):31–36.

Chu, X., Ilyas, I. F. e Papotti, P. (2013). Discovering Denial Constraints. *Proceedings of the VLDB Endowment*, 6(13):1498–1509.

Elmasri, R. e Navathe, S. B. (2016). *Fundamentals of Database Systems*. Pearson, Boston, seventh edition.

Fan, W., Geerts, F., He, X. e Riedewald, M. (2008). Foundation of data quality management. Em *Proceedings of the Twenty-seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '08)*, páginas 173–182, New York, NY, USA. ACM.

Farias, A. M. L. d. (2020). Estatística descritiva. Apostila, Departamento de Estatística, Instituto de Matemática e Estatística, Universidade Federal Fluminense, Niterói, RJ.

Heise, A., Quiané-Ruiz, J.-A., Abedjan, Z., Jentzsch, A. e Naumann, F. (2013). Scalable Discovery of Unique Column Combinations. *Proceedings of the VLDB Endowment*, 7(4):301–312.

Kandel, S., Paepcke, A., Hellerstein, J. M. e Heer, J. (2012). Profiler: Integrated Statistical Analysis and Visualization for Data Quality Assessment. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2917–2926.

Kepe, T. R., de Almeida, E. C. e Cerqueus, T. (2015). Ksample: Dynamic sampling over unbounded data streams. *Journal of Information and Data Management*, 6(1):32.

Lee, B.-H., Kim, S.-W., Cho, D.-S. e Lee, S.-H. (2007). Adaptive reservoir sampling over data streams. Em *Proceedings of the 19th International Conference on Scientific and Statistical Database Management (SSDBM 2007)*, páginas 37–37.

Martin, A., de Almeida, E. C., Romero, O. e Queralt, A. (2025). How and why false denial constraints are discovered. *Proceedings of the VLDB Endowment (PVLDB)*, 18(10):3477–3489.

Montgomery, D. C. e Runger, G. C. (2018). *Applied Statistics and Probability for Engineers*. Wiley, Hoboken, NJ, 7th edition.

Papenbrock, T., Bergmann, T., Finke, M., Zwiener, J. e Naumann, F. (2015). Data profiling with metanome. *Proceedings of the VLDB Endowment*, 8(12):1860–1871.

Pena, E. H. M., de Almeida, E. C. e Naumann, F. (2022). Fast Detection of Denial Constraint Violations. *Proceedings of the VLDB Endowment*, 15(4):859–871.

Rahm, E. e Do, H. H. (2000). Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23(4):3–13.

Tukey, J. W. (1977). *Exploratory Data Analysis*. Addison-Wesley, Reading, MA.