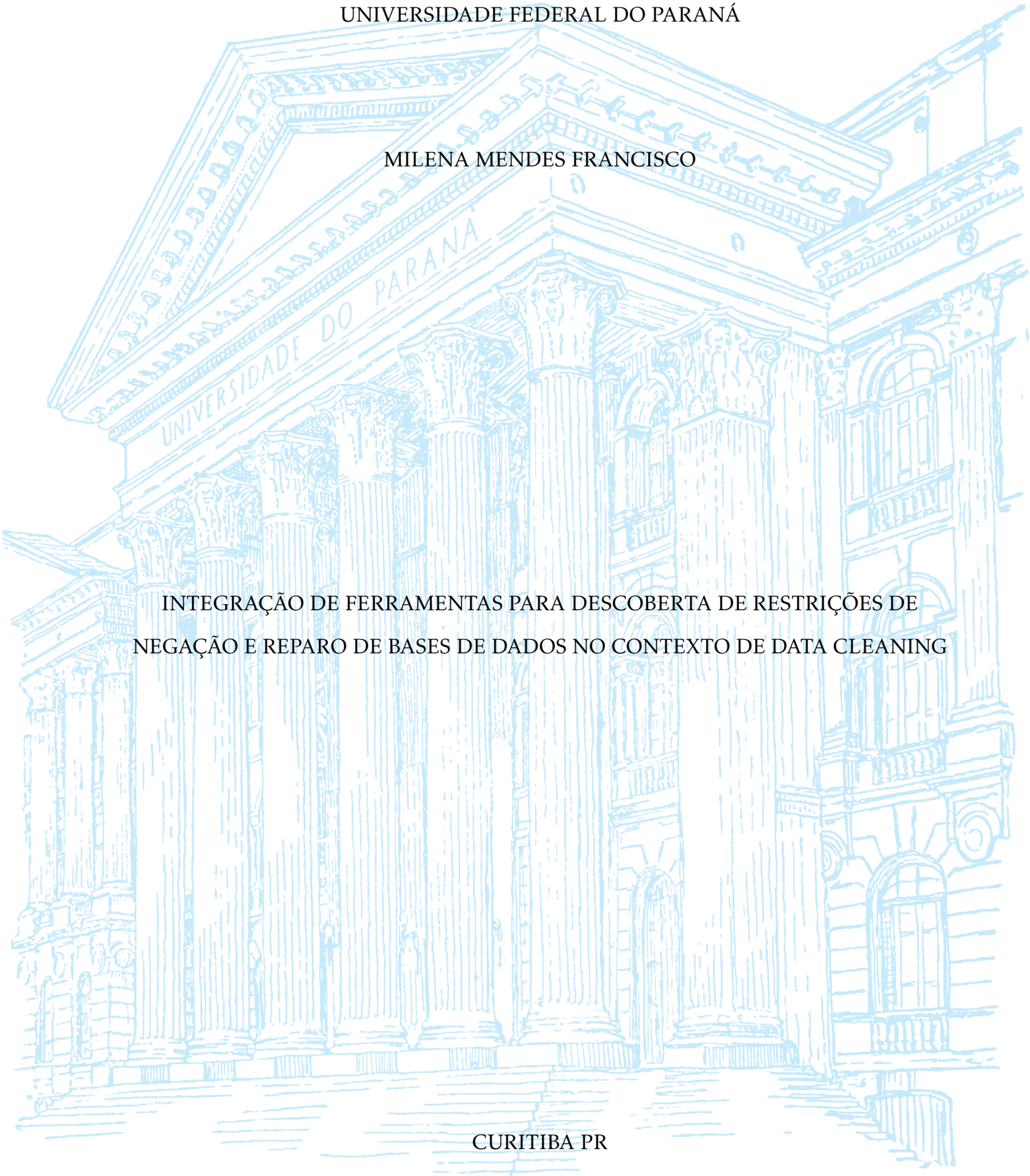
UNIVERSIDADE FEDERAL DO PARANÁ

MILENA MENDES FRANCISCO




# INTEGRAÇÃO DE FERRAMENTAS PARA DESCOBERTA DE RESTRIÇÕES DE NEGAÇÃO E REPARO DE BASES DE DADOS NO CONTEXTO DE DATA CLEANING

CURITIBA PR

2024

MILENA MENDES FRANCISCO

# INTEGRAÇÃO DE FERRAMENTAS PARA DESCOBERTA DE RESTRIÇÕES DE NEGAÇÃO E REPARO DE BASES DE DADOS NO CONTEXTO DE DATA CLEANING

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: *Computação*.

Orientador: Marcos Sfair Sunyé.

CURITIBA PR

2024

# Ficha catalográfica

Substituir o arquivo `0-iniciais/catalografica.pdf` pela ficha catalográfica fornecida pela Biblioteca da UFPR (PDF em formato A4).

**Instruções para obter a ficha catalográfica e fazer o depósito legal da tese/dissertação (contribuição de André Hochuli, abril 2019. Links atualizados Wellton Costa, Nov 2022):**

1. Estas instruções se aplicam a dissertações de mestrado e teses de doutorado. Trabalhos de conclusão de curso de graduação e textos de qualificação não precisam segui-las.
2. Verificar se está usando a versão mais recente do modelo do PPGInf e atualizar, se for necessário (`https://gitlab.c3sl.ufpr.br/maziero/tese`).
3. conferir o *checklist* de formato do Sistema de Bibliotecas da UFPR, em `https://bibliotecas.ufpr.br/servicos/normalizacao/`
4. Enviar e-mail para "`referencia.bct@ufpr.br`" com o arquivo PDF da dissertação/tese, solicitando a respectiva ficha catalográfica.
5. Ao receber a ficha, inseri-la em seu documento (substituir o arquivo `0-iniciais/catalografica.pdf` do diretório do modelo).
6. Emitir a Certidão Negativa (CND) de débito junto a biblioteca, em `https://bibliotecas.ufpr.br/servicos/certidao-negativa/`
7. Avisar a secretaria do PPGInf que você está pronto para o depósito. Eles irão mudar sua titulação no SIGA, o que irá liberar uma opção no SIGA pra você fazer o depósito legal.
8. Acesse o SIGA (`http://www.prppg.ufpr.br/siga`) e preencha com cuidado os dados solicitados para o depósito da tese.
9. Aguarde a confirmação da Biblioteca.
10. Após a aprovação do pedido, informe a secretaria do PPGInf que a dissertação/tese foi depositada pela biblioteca. Será então liberado no SIGA um link para a confirmação dos dados para a emissão do diploma.

# Ficha de aprovação

Substituir o arquivo 0-iniciais/aprovacao.pdf pela ficha de aprovação fornecida pela secretaria do programa, em formato PDF A4.

**RESUMO**

As Restrições de Negação (Denial Constraints, DCs) são expressões que definem condições proibidas em um conjunto de dados, desempenhando um papel essencial na identificação de inconsistências e erros. Ao verificar se os dados violam essas restrições, é possível isolar informações problemáticas e melhorar a integridade de uma base de dados. Essa abordagem facilita atividades como validação de registros, detecção de anomalias e correção de problemas estruturais, sendo fundamental para processos automatizados de limpeza de dados que buscam maior qualidade e confiabilidade. Este trabalho utiliza DCs para realizar a limpeza de dados em bancos relacionais e compara os resultados em bases de dados com diferentes tipos de desafios, como dados ausentes, duplicações, valores inconsistentes e erros estruturais. A metodologia proposta utiliza a ferramenta Metanome com o algoritmo DCFinder para identificar DCs a partir de uma base de dados. Essas restrições são traduzidas para o formato DDlog, por meio de um script em Python, garantindo sua integração com a ferramenta Holoclean. O banco de dados original e as restrições pré-selecionadas em DDlog são então processados pelo Holoclean, que retorna um banco otimizado e mais limpo. A abordagem foi aplicada em cinco bases de dados diferentes, demonstrando como o uso de DCs pode melhorar a qualidade dos dados e reduzir inconsistências. Este trabalho apresenta uma análise detalhada do processo de extração, formatação e aplicação das restrições, bem como os impactos observados. As conclusões incluem recomendações para o uso dessas ferramentas em cenários semelhantes e sugestões de melhorias no processo desenvolvido.

Palavras-chave: Limpeza de dados. Restrição de negação. Holoclean

**ABSTRACT**

Denial Constraints (DCs) are expressions that define prohibited conditions in a dataset, playing a crucial role in identifying inconsistencies and errors. By checking whether the data violates these constraints, it becomes possible to isolate problematic information and enhance the integrity of a database. This approach facilitates activities such as record validation, anomaly detection, and the correction of structural issues, making it fundamental for automated data cleaning processes aimed at achieving higher quality and reliability. This study utilizes DCs to perform data cleaning in relational databases and compares the results across datasets with different types of challenges, such as missing data, duplicates, inconsistent values, and structural errors. The proposed methodology employs the Metanome tool with the DCFinder algorithm to identify DCs from a database. These constraints are translated into the DDlog format using a Python script, ensuring their integration with the Holoclean tool. The original database and the preselected constraints in DDlog are then processed by Holoclean, which returns an optimized and cleaner database. The approach was applied to five different datasets, demonstrating how the use of DCs can improve data quality and reduce inconsistencies. This work presents a detailed analysis of the process of extracting, formatting, and applying the constraints, as well as the observed impacts. The conclusions include recommendations for using these tools in similar scenarios and suggestions for improving the developed process.

Keywords: Data cleaning. Denial Constraints. Holoclean.

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE ACRÔNIMOS

| | |
|---|---|
| DINF | Departamento de Informática |
| PPGINF | Programa de Pós-Graduação em Informática |
| UFPR | Universidade Federal do Paraná |
| DC | Dependência de Negação |
| PLI | Position List Indexes |
| DDlog | Differentiable Datalog |

**SUMÁRIO**

# 1 INTRODUÇÃO

Com o aumento do volume de dados disponíveis em diversos domínios, como saúde, educação e finanças, cresce a necessidade de garantir a qualidade desses dados para que análises e decisões sejam tomadas de forma confiável. Dados inconsistentes, incompletos ou duplicados podem comprometer os resultados de análises e algoritmos, tornando indispensável o processo de limpeza de dados (*data cleaning*). Esse processo visa identificar e corrigir problemas nas bases de dados, garantindo sua consistência e conformidade com as regras de negócios.

Uma abordagem eficiente para realizar a limpeza de dados envolve o uso de ferramentas baseadas em restrições, como restrições de negação (*denial constraints*). Essas restrições descrevem regras que devem ser respeitadas pelos dados, ajudando na identificação de inconsistências. No entanto, a definição e descoberta de restrições adequadas para bases de dados pode ser desafiadora, especialmente em cenários onde há presença de dados faltantes, que representam de 20% a 80% dos dados em aplicações práticas, segundo Berti-Équille et al. (2018).

Neste contexto, o presente trabalho explora o uso de restrições de negação para auxiliar no processo de limpeza de dados, utilizando a ferramenta Metanome para descoberta de tais restrições e o Holoclean para aplicar e avaliar essas restrições em bases de dados. O objetivo é investigar o impacto das restrições na melhoria da qualidade dos dados, comparando através de métricas de avaliação, os resultados da limpeza.

## 1.1 CONTRIBUIÇÃO

Este trabalho apresenta uma abordagem para a limpeza de dados utilizando *denial constraints*. As contribuições incluem: A criação de um método que traduz as restrições descobertas pelo Metanome para o formato DDlog utilizado pelo Holoclean, permitindo sua integração em pipelines de limpeza de dados e uma análise comparativa dos resultados obtidos após a limpeza das bases escolhidas, demonstrando sua eficiência na detecção de inconsistências e melhoria da confiabilidade dos dados. Além disso, este trabalho reforça a importância do uso integrado de ferramentas modernas para lidar com os desafios crescentes na qualidade de dados em bases cada vez maiores e mais complexas.

## 1.2 DESAFIOS

Durante o desenvolvimento deste trabalho, diversos desafios foram enfrentados. Entre eles, destaca-se a seleção de bases de dados, que foram selecionadas a partir do portal Hasso-Plattner (HPI) [1], e a necessidade de traduzir as restrições geradas pelo Metanome para o formato DDlog do Holoclean, o que exigiu a criação de scripts customizados. Além disso, a descoberta e seleção de restrições relevantes mostrou-se desafiadora devido à complexidade dos dados analisados. Outro ponto crítico foi a integração entre ferramentas que não possuem compatibilidade nativa, demandando a criação de pipelines específicos.

## 1.3 ORGANIZAÇÃO DO DOCUMENTO

Este trabalho está estruturado em seis capítulos principais. No Capítulo 1, é apresentada a introdução, contextualizando o problema e destacando os desafios enfrentados na limpeza de dados. O Capítulo 2 discorre sobre a fundamentação teórica necessária, abordando dependências de negação, algoritmos de descoberta, entre outros termos necessários para a compreensão do trabalho. No Capítulo 3, são revisados trabalhos relacionados, enfatizando as ferramentas utilizadas, o Metanome e o Holoclean, e outro importante algoritmo para descoberta de restrições. O Capítulo 4 detalha o desenvolvimento do trabalho, explicando o fluxo completo, desde a descoberta de restrições com o Metanome, a tradução para o formato DDlog, até a aplicação das restrições no Holoclean. O Capítulo 5 apresenta os resultados obtidos, incluindo análises comparativas de desempenho e impacto na qualidade dos dados. Por fim, o Capítulo 6 conclui o estudo, destacando os resultados e propondo possibilidades de trabalhos futuros, como o desenvolvimento de uma interface gráfica para automatização do fluxo.

# 2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, alguns conceitos fundamentais para a compreensão do trabalho são apresentados.

## 2.1 DEPENDÊNCIAS EM BANCO DE DADOS

As dependências em bancos de dados são propriedades semânticas que os dados devem satisfazer para representar um estado válido do mundo real. Elas definem relações e restrições entre os valores dos dados, assegurando a consistência e integridade da base de dados. Segundo Pena et al. [2], uma violação de dependência ocorre quando combinações de valores em um ou mais registros não satisfazem a relação imposta por essa dependência. Um banco de dados é considerado consistente quando não há violações das dependências definidas para ele. Essas dependências podem variar em complexidade e incluir dependências tradicionais, como dependências funcionais e de inclusão, além de versões estendidas, como dependências parciais, condicionais e aproximadas, que são úteis para capturar exceções ou erros nos dados.

### 2.1.1 DEPENDÊNCIAS DE NEGAÇÃO

As dependências de negação (*Denial Constraints*, DCs) são um formalismo generalizado que abrangem diversos tipos de dependências, elas especificam estados inconsistentes de valores em colunas usando predicados relacionais. Uma DC é representada como um conjunto de predicados que descrevem condições proibidas no banco de dados, assim, quaisquer tuplas ou conjuntos de tuplas que não satisfaçam as restrições definidas por uma DC constituem uma violação.[3]

De acordo com Pena et al. [3] uma DC $\phi$ sobre uma instância relacional $r$ é uma declaração na forma:

$$\phi : \forall t_x, t_y \in r, \neg(p_1 \wedge \cdots \wedge p_m)$$

onde $\phi$ é satisfeita por $r$ se, e somente se, para qualquer par de tuplas $t_x, t_y \in r$, pelo menos um dos predicados $p_1, \ldots, p_m$ for falso.

Uma DC $\phi_1$ é mínima se não existir uma $\phi_2$ tal que ambas $\phi_1$ e $\phi_2$ sejam satisfeitas por $r$, e os predicados de $\phi_2$ sejam um subconjunto dos de $\phi_1$.

Por exemplo, em um banco de dados que registra informações sobre personagens de *Genshin Impact*, uma DC (fictícia) pode ser usada para garantir que dois personagens com a mesma raridade (★) e elemento não podem ter valores diferentes de ataque base. A restrição seria representada da seguinte forma:

$$\phi : \neg(t.\text{Raridade} = t'.\text{Raridade} \wedge t.\text{Elemento} = t'.\text{Elemento} \wedge t.\text{Ataque} \neq t'.\text{Ataque})$$

Neste exemplo, a DC verifica se há personagens da mesma raridade e elemento, mas com valores de ataque diferentes. Aqui, Diluc e Hu Tao violam a DC, pois ambos possuem a mesma raridade (5★) e elemento (Pyro), mas têm níveis de ataque diferentes (1000 e 950).

A partir dessa violação, o sistema de limpeza de dados poderia sinalizar as tuplas ou aplicar correções baseadas nas regras definidas pela DC.

Tabela 2.1: Relação de personagens.

| Nome | Raridade | Elemento | Ataque |
|---|---|---|---|
| Diluc | 5★ | Pyro | 1000 |
| Bennett | 4★ | Pyro | 600 |
| Hu Tao | 5★ | Pyro | 950 |

#### *2.1.1.1 DCs Exatas e Aproximadas*

As Denial Constraints podem ser classificadas em exatas e aproximadas, de acordo com o nível de tolerância às violações permitidas. De acordo com Pena et all.[3], as DCs exatas são restrições rígidas que devem ser completamente satisfeitas por todas as tuplas de um banco de dados. Já as DCs aproximadas oferecem maior flexibilidade, permitindo que algumas violações sejam toleradas dentro de limites aceitáveis. Essa abordagem é útil em bancos de dados reais, onde erros e inconsistências são comuns devido a entradas incorretas ou dados provenientes de múltiplas fontes. As DCs aproximadas introduzem uma medida de tolerância, como um limite $\epsilon$, que define o grau de violação permitido, sendo amplamente utilizadas em sistemas probabilísticos como o HoloClean para modelar incertezas e identificar reparos prováveis.

A escolha entre DCs exatas e aproximadas depende do contexto do sistema de dados. Enquanto as exatas garantem precisão absoluta e são adequadas para cenários críticos, as aproximadas são mais indicadas para ambientes com dados ruidosos, onde flexibilidade e escalabilidade são prioridades. Assim, ambas desempenham papéis importantes no processo de limpeza e garantia da qualidade dos dados.

## 2.2 ALGORITMOS DE DESCOBERTA DE DEPENDÊNCIAS

Reforçamos aqui que a descoberta de dependências é um componente crucial na limpeza de dados, integração de bases e otimização de consultas. Em particular, as dependências de negação têm ganhado destaque por sua capacidade de generalizar vários tipos de dependências e identificar inconsistências em bancos de dados. No entanto, devido ao espaço de busca exponencial das DCs, a descoberta automática dessas restrições é uma tarefa desafiadora e requer algoritmos eficientes.

Diferentes tipos de dependências (como dependências funcionais, de inclusão e de negação) possuem características únicas que influenciam a escolha dos algoritmos de descoberta. Por exemplo, algoritmos como o TANE [4] para dependências funcionais utilizam estruturas para reduzir o custo computacional. Já para DCs, o espaço de busca envolve combinações de predicados, o que exige estruturas específicas para validação eficiente.

Neste trabalho, utilizamos o algoritmo DCFinder [3], que se destaca como um dos mais eficientes para a descoberta de DCs, tanto exatas quanto aproximadas. Esse algoritmo combina estruturas de dados chamadas *Position List Indexes (PLIs)* e técnicas baseadas na seletividade de predicados para validar candidatos de DCs. Ele é particularmente adequado para lidar com grandes volumes de dados e oferece suporte à descoberta de DCs aproximadas, permitindo lidar com inconsistências inevitáveis em bancos de dados reais.

## 2.3 DCFINDER

### 2.3.1 Funcionamento do DCFinder

O DCFinder [3] é estruturado em três etapas principais:

- Transformação do Dataset: A partir do esquema do banco de dados, o algoritmo define um espaço de predicados e constrói índices de posição (PLIs). Esses índices associam valores únicos a clusters de tuplas que compartilham esses valores, permitindo identificar rapidamente pares de tuplas que satisfazem determinados predicados.
- Geração do Conjunto de Evidências (*Evidence Set*): O conjunto de evidências representa pares de tuplas e os predicados que elas satisfazem. O DCFinder utiliza técnicas baseadas na seletividade de predicados para minimizar o número de operações necessárias, reduzindo significativamente os custos computacionais.
- Busca por Coberturas Mínimas (*Minimal Covers*): A partir do conjunto de evidências, o algoritmo realiza uma busca em profundidade (depth-first search) para identificar todas as DCs mínimas que cobrem o conjunto de evidências. Essa etapa inclui a validação de DCs exatas e aproximadas, dependendo do grau de tolerância a violações especificado pelo usuário.

### 2.3.2 Componentes do Algoritmo

- *Position List Indexes (PLIs)*: Estruturas que agrupam valores únicos e os identificadores das tuplas que compartilham esses valores. PLIs são usados para gerar pares de tuplas de forma eficiente, reduzindo a necessidade de comparar diretamente todas as combinações possíveis.
- Espaço de Predicados: Define todas as combinações possíveis de predicados para os atributos do banco de dados. Predicados incluem operadores como =, !=, <, <=, >, >=.
- Reconstrução de Evidências: Ajusta os dados armazenados para refletir corretamente os predicados satisfeitos por cada par de tuplas.

### 2.3.3 Vantagens do DCFinder

- Eficiência Computacional: Comparado a outros algoritmos, como FASTDC [4] e HYDRA [4], o DCFinder apresenta melhor desempenho em grandes bases de dados, sendo capaz de processar milhões de tuplas em menos tempo.
- Suporte a DCs Aproximadas: Permite descobrir restrições que toleram pequenas violações, o que é crucial em bases de dados com inconsistências inerentes.
- Integração com Ferramentas de Limpeza: As DCs descobertas podem ser aplicadas, depois de processamento e tradução, em ferramentas como o HoloClean para melhorar a qualidade dos dados.

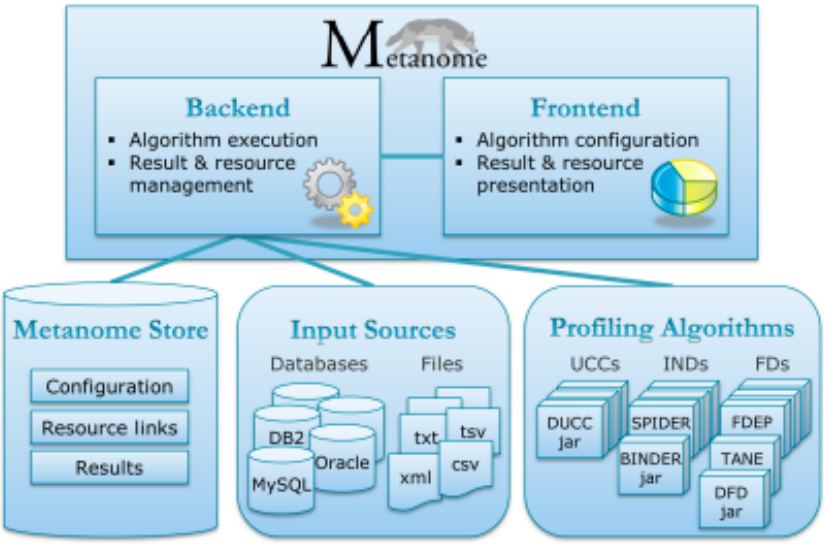



Figura 2.1: Arquitetura da plataforma de perfilamento Metanome [5].

## 2.4 METANOME PARA DESCOBERTA DE RESTRIÇÕES DE NEGAÇÃO

O Metanome [5] é uma ferramenta extensível e modular projetada para a descoberta de metadados em bancos de dados, sendo amplamente utilizada em tarefas como limpeza de dados, integração de dados e otimização de consultas. No contexto deste trabalho, o Metanome é utilizado para a descoberta automática de restrições de negação. Essas restrições são essenciais para identificar inconsistências e melhorar a integridade de bases de dados, atuando como um alicerce no processo de limpeza automatizada.

### 2.4.1 Arquitetura e Funcionamento do Metanome

O Metanome organiza suas funcionalidades em três camadas principais que suportam todo o processo de descoberta de DCs, conforme figura 2.1:

- **Camada de Dados:** Responsável pela ingestão e abstração das fontes de dados. Os dados podem ser fornecidos em diferentes formatos, como arquivos CSV ou diretamente de bancos de dados relacionais, garantindo compatibilidade com uma ampla gama de aplicações.
- **Camada de Lógica:** Implementa os algoritmos de descoberta, gerencia os resultados e oferece suporte para processamento eficiente, como a criação de estruturas temporárias para otimizar a execução em grandes volumes de dados.
- **Camada de Apresentação:** Permite configurar os parâmetros de execução, selecionar algoritmos e visualizar os resultados. No caso da descoberta de DCs, os resultados são apresentados de forma legível e podem ser exportados para integração com outras ferramentas.

Essa arquitetura modular torna o Metanome uma ferramenta robusta e flexível, capaz de integrar algoritmos especializados para diferentes tipos de metadados, como DCs.

### 2.4.2 Descoberta de Restrições de Negação

O Metanome utiliza algoritmos especializados, como o **DCFinder**, para a descoberta de DCs em bancos de dados. Esses algoritmos exploram relações entre atributos e tuplas para identificar predicados que descrevem condições proibidas. O fluxo básico para a descoberta de DCs no Metanome pode ser descrito em quatro etapas:

1. **Definição das Fontes de Dados:** O usuário fornece os dados de entrada em formatos suportados, como arquivos CSV. O Metanome abstrai esses dados e cria uma estrutura interna padronizada para análise.

2. **Configuração do Algoritmo:** O DCFinder é configurado com parâmetros específicos, como o limite máximo de predicados por DC ou o tempo de execução permitido. Esses parâmetros influenciam o número e a qualidade das DCs descobertas.

3. **Execução do Algoritmo:** O DCFinder identifica pares de tuplas e gera o conjunto de predicados que descrevem inconsistências entre elas. Ele valida combinações de atributos, operadores lógicos (e.g., $=$, $!=$, $<$, $>$) e valores, criando um conjunto inicial de DCs candidatas.

4. **Filtragem e Exportação:** Após a descoberta, as DCs são filtradas para remover duplicatas, restrições triviais e redundâncias. O conjunto final é exportado em um formato que pode ser integrado com outras ferramentas, como o HoloClean.

### 2.4.3 Benefícios do Uso do Metanome para Descoberta de DCs

O uso do Metanome para a descoberta de DCs oferece diversos benefícios:

- **Automatização:** Reduz a necessidade de definir manualmente restrições, acelerando o processo de identificação de inconsistências em grandes conjuntos de dados.

- **Flexibilidade:** Permite a configuração de algoritmos para se adequar a diferentes cenários e tipos de dados.

- **Escalabilidade:** Seus algoritmos suportados são capazes de processar grandes volumes de dados, utilizando técnicas de otimização como armazenamento temporário e paralelismo.

## 2.5 MÉTRICAS DE AVALIAÇÃO DE DEPENDENCIAS DE NEGAÇÃO

Embora um algoritmo de descoberta de DCs, como o DCFinder, seja eficaz em identificar diversas restrições de negação a partir de um conjunto de dados, um desafio persistente é a quantidade de DCs geradas, pois muitas delas podem não ser relevantes ou significativas para a análise desejada. Para mitigar esse problema, é fundamental aplicar métricas que ajudem a classificar e priorizar as DCs, destacando aquelas que realmente contribuem para a qualidade e integridade dos dados. No contexto da descoberta de DCs, métricas como *Succinctness*, *Coverage* e *Interestingness* são comumente utilizadas para avaliar e ranquear as DCs, permitindo identificar as mais úteis e eficazes para o processo de limpeza de dados [3, 6]

### 2.5.1 *Succinctness*

A métrica *Succinctness* busca avaliar a concisão de uma restrição de negação, isto é, o quão 'compacta' uma DC é em relação à quantidade mínima de informação necessária para representá-la sem perder a lógica da restrição. O princípio subjacente a essa métrica sugere que, entre várias explicações possíveis para um fenômeno, a mais simples, que faz menos suposições, é geralmente a mais verdadeira. Esse princípio é aplicado na seleção de DCs, onde a *Succinctness* é usada para priorizar as restrições mais simples, ou seja, aquelas que podem ser expressas com um número reduzido de elementos e predicados.

Para calcular a *Succinctness* de uma DC, utiliza-se uma fórmula que compara o tamanho da DC com o tamanho da menor DC dentro de um conjunto de restrições extraídas do banco de dados. Formalmente, a *Succinctness* de uma DC é dada pela razão entre o tamanho da menor DC encontrada no conjunto e o tamanho da DC que estamos avaliando. O tamanho de uma DC é definido pelo número de símbolos únicos envolvidos na restrição, que inclui as tuplas, atributos, operadores e constantes. A métrica ajuda a eliminar restrições excessivamente complexas ou redundantes, favorecendo aquelas que são mais diretas e eficazes para a tarefa de limpeza de dados. Além disso, essa métrica visa evitar o fenômeno do overfitting, onde o modelo ou a regra descoberta se ajusta excessivamente aos dados específicos do conjunto de treinamento, sem generalizar bem para outros casos. Em um cenário prático, uma DC com alta *Succinctness* é preferida, pois ela reflete uma regra simples e robusta, representando o fenômeno de forma eficaz e sem complexidade desnecessária.

### 2.5.2 *Coverage*

A métrica *Coverage* está diretamente relacionada à importância e ao impacto de uma restrição de negação dentro de um banco de dados. Ela mede a extensão ou abrangência da DC, ou seja, quantas tuplas ou combinações de tuplas no conjunto de dados satisfazem os predicados da restrição. Quanto maior a *Coverage* de uma DC, mais relevante ela se torna para o banco de dados, pois significa que a restrição está sendo satisfeita por um maior número de evidências, ou seja, tuplas que atendem aos critérios da restrição.

A *Coverage* é calculada com base na quantidade de evidências que satisfazem a DC, onde uma evidência corresponde a um conjunto de tuplas que satisfaz pelo menos um predicado da restrição. O valor da *Coverage* aumenta à medida que a restrição abrange mais tuplas no banco de dados, tornando-se, assim, mais forte e significativa. Para formalizar esse conceito, a *Coverage* de uma DC é calculada somando-se a quantidade de vezes que os predicados da DC são satisfeitos, ponderada pelo peso de cada evidência. Esse peso é determinado pelo número de predicados satisfeitos por uma evidência, o que reflete a força da evidência em contribuir para a DC. Quanto maior a *Coverage*, maior é a probabilidade de a DC ser uma boa candidata para impor regras de integridade no banco de dados, já que ela é aplicada a uma quantidade significativa de dados.

Essa métrica é fundamental para entender a relevância das DCs em contextos práticos de limpeza de dados. Ao classificar e priorizar as DCs com maior *Coverage*, podemos assegurar que as restrições mais gerais, aquelas que têm maior aplicação dentro do banco de dados, sejam aplicadas primeiro, aumentando a eficácia do processo de limpeza de dados.

### 2.5.3 Interestingness

A métrica *Interestingness* é uma combinação ponderada das métricas *Succinctness* e *Coverage*, e tem como objetivo identificar as DCs que não apenas são concisas e simples, mas também têm uma ampla aplicabilidade nos dados. A ideia por trás da *Interestingness* é que as DCs mais interessantes são aquelas que têm um bom equilíbrio entre ser concisas e serem amplamente aplicáveis aos dados, ou seja, aquelas que são tanto simples quanto relevantes. Essa métrica é crucial, pois permite classificar as DCs de forma que as mais úteis para o processo de limpeza de dados sejam priorizadas.

A fórmula para calcular a Interestingness de uma DC combina as métricas *Succinctness* e *Coverage* de forma ponderada. O peso atribuído a cada uma das métricas é controlado por um parâmetro chamado $\alpha$, que define o nível de importância dado a cada uma das métricas na composição da pontuação final. Se $\alpha$ for definido como 0, o cálculo da Interestingness dependerá apenas da *Succinctness*, favorecendo DCs simples, mesmo que não cubram uma grande parte do banco de dados. Por outro lado, se $\alpha$ for 1, a *Interestingness* será determinada exclusivamente pela *Coverage*, priorizando DCs que têm um grande número de tuplas satisfazendo seus predicados. Em um cenário equilibrado, $\alpha$ é ajustado para 0.5, garantindo que tanto a simplicidade (concorrente com a *Succinctness*) quanto a aplicabilidade (medida pela *Coverage*) tenham peso igual na pontuação de *Interestingness*. Essa métrica ajuda a identificar quais DCs são verdadeiramente úteis para a análise, já que elas refletem tanto a simplicidade quanto a relevância dentro do contexto dos dados, proporcionando uma medida robusta de "interesse"para cada DC.

### 2.5.4 Exemplo de cálculo das métricas

Neste exemplo, utilizamos uma tabela fictícia de personagens do *Genshin Impact*, tabela 2.2, para demonstrar os cálculos das métricas ***Succinctness***, ***Coverage*** e ***Interestingness***.

Tabela 2.2: Tabela Fictícia de Personagens de *Genshin Impact*

| Nome | Raridade | Elemento | Ataque | Vida | Defesa |
|---|---|---|---|---|---|
| Diluc | 5★ | Pyro | 1000 | 1500 | 800 |
| Bennett | 4★ | Pyro | 600 | 1200 | 650 |
| Hu Tao | 5★ | Pyro | 950 | 1400 | 700 |
| Lisa | 4★ | Electro | 500 | 1100 | 600 |
| Fischl | 4★ | Electro | 550 | 1000 | 620 |
| Venti | 5★ | Anemo | 800 | 1300 | 700 |

#### *2.5.4.1 Definição das Restrições de Negação (DCs)*

Definimos DCs para este banco de dados fictício:

- **DC1**: Dois personagens com a mesma raridade não podem ter valores de **Ataque** diferentes:

$$\phi_1 : \forall t_x, t_y \in r, \neg(t_x.\text{Raridade} = t_y.\text{Raridade} \wedge t_x.\text{Ataque} \neq t_y.\text{Ataque})$$

- **DC2**: Personagens do elemento **Pyro** não podem ter **Defesa** maior que 700:

$$\phi_2 : \forall t_x \in r, \neg(t_x.\text{Elemento} = \text{Pyro} \wedge t_x.\text{Defesa} > 700)$$

- **DC3**: Personagens com raridade **5★** devem ter **Vida** maior que 1200:

$$\phi_3 : \forall t_x \in r, \neg(t_x.\text{Raridade} = 5\bigstar \wedge t_x.\text{Vida} \leq 1200)$$

#### 2.5.4.2 *Cálculo da Succinctness*

A métrica ***Succinctness*** é calculada pela fórmula:

$$Succinctness(\phi) = \frac{\text{Len}(\text{Min}(\text{Ø}))|\forall \text{Ø}}{\text{Len}(\text{Ø})}$$

onde:

- Len(Ø) é o tamanho da DC em termos de símbolos distintos (tuplas, atributos, operadores e constantes).
- Len(Min(Ø)) é o tamanho da menor DC no conjunto. Para o nosso exemplo, a menor DC é a DC1.

Os cálculos são os seguintes:

- **DC1**: Len(Ø) = 6 (tx, ty, Raridade, =, Ataque,≠), Min(Ø) = 6, então:

$$\text{Succ}(DC1) = \frac{6}{6} = 1$$

- **DC2**: Len(Ø) = 7 (tx, Elemento, =, Pyro, Defesa, >, 700), Min(Ø) = 6, então:

$$\text{Succ}(DC2) = \frac{6}{7} = 0.85$$

- **DC3**: Len(Ø) = 7 (tx, Raridade, =, 5★, Vida, ≤, 1200), Min(Ø) = 6, então:

$$\text{Succ}(DC3) = \frac{6}{7} = 0.85$$

#### 2.5.4.3 *Cálculo da Coverage*

A métrica ***Coverage*** é calculada pela fórmula:

$$Coverage(\phi) = \frac{\sum_{k=0}^{\text{Len}(\phi)-1} |kE| \cdot w(k)}{\sum_{k=0}^{\text{Len}(\phi)-1} |kE|}$$

onde:

- $kE$ é o número de tuplas que satisfazem pelo menos $k$ predicados da DC.
- $w(k)$ é o peso associado à evidência-$k$. Para esses cálculos, foi utilizado 0.5 para todas as restrições.

Os cálculos são os seguintes:

- **DC1**: $Coverage(DC1) = 0.33$ (Diluc e Hu Tao violam a DC).
- **DC2**: $Coverage(DC2) = 0.16$ (Diluc viola a DC).
- **DC3**: $Coverage(DC3) = 1$ (satisfeita por todas as tuplas no dataset).

#### *2.5.4.4 Cálculo da Interestingness*

A métrica ***Interestingness*** é calculada pela fórmula:

$$Interestingness(\phi) = \alpha \times Coverage(\phi) + (1 - \alpha) \times Succinctness(\phi)$$

onde $\alpha$ é o peso entre as métricas. Considerando $\alpha = 0.5$, os cálculos são:

- **DC1**: $Interestingness(DC1) = 0.5 \times 0.33 + 0.5 \times 1 = 0.665$
- **DC2**: $Interestingness(DC2) = 0.5 \times 0.16 + 0.5 \times 0.85 = 0.505$
- **DC3**: $Interestingness(DC3) = 0.5 \times 1 + 0.5 \times 0.85 = 0.925$

#### *2.5.4.5 Conclusão dos Cálculos*

Tabela 2.3: Resultados das Métricas para as DCs

| DC | *Succinctness* | *Coverage* | *Interestingness* |
|---|---|---|---|
| DC1 | 1 | 0.33 | 0.655 |
| DC2 | 0.85 | 0.16 | 0.505 |
| DC3 | 0.85 | 1 | 0.925 |

Com base nos cálculos, que podem ser melhor visualizados na tabela 2.3, a DC **DC3** apresenta maior ***Interestingness***, sendo a mais relevante para o banco de dados analisado.

## 2.6 DDLOG

### 2.6.1 Definição e Uso do DDlog

O DDlog (*Differentiable Datalog*) é uma linguagem declarativa derivada de Datalog, projetada para modelar e raciocinar sobre dados em sistemas de aprendizado probabilístico, como o HoloClean [7]. Ele é amplamente utilizado para criar gráficos fatoriais que representam dependências e incertezas nos dados, permitindo a reparação e limpeza de conjuntos de dados com alta precisão e escalabilidade[7].

#### *2.6.1.1 Por que o DDlog é importante?*

O DDlog permite expressar regras de inferência probabilística e dependências lógicas de forma eficiente. Ele é especialmente valioso em aplicações de limpeza de dados, onde diversas fontes de sinais (como restrições de negação, estatísticas quantitativas e dados externos) precisam ser combinadas para gerar reparos precisos[7]. Utilizando DDlog, é possível:

- Criar modelos probabilísticos para capturar incertezas nos dados;
- Definir relações entre variáveis que representam dados errados e seus valores corretos estimados;
- Escalar processos de inferência em grandes conjuntos de dados.

#### *2.6.1.2 Exemplo de Regra DDlog*

Abaixo, apresentamos um exemplo de regra DDlog para derivar tuplas com base em atributos:

$$Q(x, y) : -R(x, y), S(y), [x = "a"].$$

Nesta regra:

- Q(x, y) é a relação de saída.
- R(x, y) e S(y) são as relações de entrada.
- [x = "a"] é uma condição que limita os valores de x.

#### *2.6.1.3 Tradução de DCs para DDlog*

As restrições de negação podem ser representadas no DDlog através de regras que modelam as restrições entre tuplas. Consideremos a seguinte DC, que define que dois personagens com a mesma raridade não podem ter valores de ataque diferentes:

$$\phi_1 : \forall t_x, t_y \in r, \neg(t_x.\text{Raridade} = t_y.\text{Raridade} \wedge t_x.\text{Ataque} \neq t_y.\text{Ataque})$$

A tradução desta restrição para DDlog seria a seguinte:

```
!(Value?(tx, Raridade, r1) ∧ Value?(ty, Raridade, r2) ∧ Value?(tx,
Ataque, a1) ∧ Value?(ty, Ataque, a2) ∧ r1 = r2 ∧ a1 ≠ a2) :-
Tuple(tx), Tuple(ty).
```

Nesta regra:

- `Value?(tx, Raridade, r1)` e `Value?(ty, Raridade, r2)` representam os valores do atributo `Raridade` para as tuplas `tx` e `ty`.
- `Value?(tx, Ataque, a1)` e `Value?(ty, Ataque, a2)` representam os valores do atributo `Ataque`.
- `r1 = r2` e `a1 ≠ a2` garantem que a restrição seja aplicada quando as tuplas têm a mesma raridade, mas valores de ataque diferentes.
- `Tuple(tx), Tuple(ty)` indicam que as variáveis `tx` e `ty` representam tuplas válidas do conjunto de dados.

Esta tradução exemplifica como as DCs podem ser convertidas para o formato DDlog, permitindo sua integração em sistemas de limpeza de dados como o HoloClean. A linguagem DDlog torna possível representar essas regras de forma eficiente, facilitando a identificação de inconsistências e a aplicação de correções baseadas nos sinais gerados.

#### 2.6.1.4 *Aplicação prática*

No contexto do HoloClean, o DDlog é usado para mapear sinais de reparo em regras probabilísticas que são posteriormente inferidas para identificar e corrigir erros nos dados. Por exemplo:

1. **Erro de dados:** Se duas tuplas compartilharem um código postal, mas tiverem cidades diferentes, o DDlog sinaliza uma violação da restrição.

2. **Correção:** O HoloClean calcula a probabilidade de cada cidade ser a correta e sugere o reparo com maior confiança.

Combinando sua base declarativa e capacidade probabilística, o DDlog é uma ferramenta poderosa para modelar incertezas e raciocinar sobre dados inconsistentes, tornando-o essencial em sistemas modernos de limpeza e reparo de dados.

## 2.7 HOLOCLEAN

HoloClean [7] é uma plataforma inovadora para a reparação holística de dados com base em inferência probabilística. Sua abordagem combina sinais qualitativos, como restrições de integridade, e quantitativos, como propriedades estatísticas dos dados de entrada, para identificar e corrigir inconsistências. O funcionamento do HoloClean envolve três etapas principais: detecção de erros, compilação e reparação dos dados, conforme ilustrado na Figura 2.2.

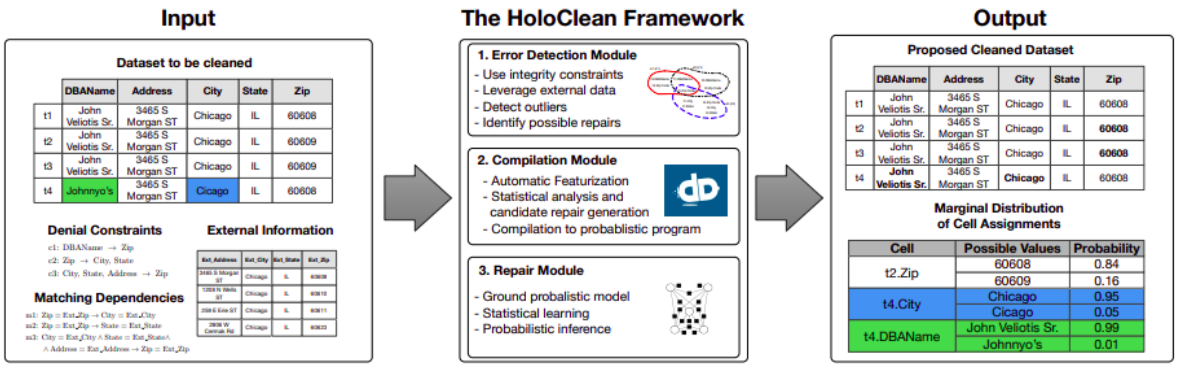



Figura 2.2: Fluxo de trabalho do HoloClean, destacando as etapas de detecção de erros, compilação e reparação de dados [7].

### 2.7.1 Detecção de Erros

O primeiro passo no fluxo de trabalho do HoloClean é a detecção de erros em células do conjunto de dados. Essa etapa separa os dados em células potencialmente ruidosas (Dn) e células limpas (Dc). O HoloClean suporta métodos como detecção baseada em restrições de negação, mecanismos de detecção de outliers e o uso de dados externos ou rotulados. A identificação de células ruidosas é crucial para guiar as etapas subsequentes de reparação.

### 2.7.2 Compilação

Após a detecção de erros, o HoloClean traduz os sinais disponíveis em um programa probabilístico usando a linguagem DDlog. Essa compilação gera variáveis aleatórias para cada célula do dataset, representando incertezas sobre seus valores. As regras no DDlog modelam relações entre variáveis, capturando estatísticas quantitativas, dependências de dados e informações externas. Isso resulta na construção de um grafo probabilístico de fatores, permitindo que o HoloClean combine múltiplos sinais durante o processo de reparação.

### 2.7.3 Reparação de Dados

Na etapa final, o HoloClean utiliza aprendizado estatístico e inferência probabilística para calcular as probabilidades marginais de cada célula estar correta. As reparações propostas são associadas a probabilidades que indicam a confiança do sistema em cada modificação. Por exemplo, um reparo com probabilidade marginal de 0.6 implica que o HoloClean está 60% confiante na correção proposta.

### 2.7.4 Uso de Restrições de Negação

O HoloClean suporta nativamente restrições de negação, que precisam estar traduzidas em regras DDlog para a construção do grafo probabilístico. Essas restrições ajudam a modelar dependências complexas e a identificar inconsistências. Além disso, o sistema permite relaxar restrições rígidas para melhorar a escalabilidade sem comprometer significativamente a qualidade dos reparos.

### 2.7.5 Eficiência e Escalabilidade

Para lidar com grandes conjuntos de dados, o HoloClean implementa otimizações como poda de domínios de variáveis aleatórias e particionamento de tuplas antes da geração de regras. Essas estratégias reduzem significativamente o custo computacional, permitindo que o sistema processe milhões de registros com eficiência.

### 2.7.6 Aplicação Prática

Estudos mostram que o HoloClean atinge precisão média de 90% e recall superior a 76% em diversos datasets reais, superando em mais de 2 vezes os métodos tradicionais de reparação de dados [7]. A combinação de múltiplos sinais e a flexibilidade para incorporar dados externos tornam o HoloClean uma ferramenta robusta para reparação de dados em cenários variados.

# 3 TRABALHOS RELACIONADOS

Diversos trabalhos na literatura abordam os desafios da descoberta de restrições e limpeza de dados em cenários de bases de dados ruidosas e inconsistentes. Neste capítulo, apresentamos brevemente um outro algoritmo para descoberta de restrições de negação.

## 3.1 FAST DETECTION OF DENIAL CONSTRAINT VIOLATIONS (FACET)

Outro algoritmo eficiente para detectar violações de restrições de negação em datasets é o FACET [8], que organiza os conflitos em um hipergrafo que representa as relações entre tuplas inconsistentes. Baseado no formalismo de DCs, esse algoritmo aproveita a facilidade de traduzir essas restrições em consultas SQL otimizadas.

Para aumentar a eficiência, o FACET categoriza os predicados das DCs em igualdade (=), desigualdade (≠) e desigualdade numérica (<, ≤, >, ≤), aplicando algoritmos especializados para cada tipo, como *hash-join* para igualdades e *IEJoin* para desigualdades numéricas. Essas estratégias minimizam intermediários grandes, reduzem chamadas de sistema e otimizam o uso de memória. Assim, o FACET oferece uma solução escalável e robusta para identificar erros em bases de dados, contribuindo para melhorar a qualidade e a integridade dos dados.

## 3.2 CONCLUSÃO

O Metanome desempenha um papel fundamental na descoberta de Denial Constraints, oferecendo uma plataforma robusta e configurável para automatizar a análise de dados. Com sua capacidade de integrar algoritmos especializados, como o DCFinder, a ferramenta possibilita a identificação de restrições relevantes, otimizando processos de limpeza de dados em cenários complexos e heterogêneos.

O HoloClean se destaca como uma ferramenta robusta e inovadora para limpeza de dados, combinando técnicas de inferência probabilística e aprendizado estatístico com sinais qualitativos e quantitativos. Sua capacidade de integrar múltiplos sinais, como restrições de negação, estatísticas de dados e fontes externas, permite identificar e corrigir inconsistências de forma eficiente e escalável, mesmo em grandes volumes de dados. Além disso, sua abordagem probabilística garante reparações confiáveis, apresentando resultados com níveis de confiança associados, o que facilita a validação posterior por especialistas. Com sua flexibilidade e alto desempenho, o HoloClean oferece uma abordagem abrangente para problemas complexos de limpeza de dados, sendo uma ferramenta indispensável para melhorar a integridade e a confiabilidade em bases de dados reais.

Assim, este trabalho contribui para a área ao explorar a sinergia entre o Metanome e o HoloClean, aplicando técnicas de descoberta e limpeza em conjunto para melhorar a qualidade dos dados e apresentando uma solução para a integração das duas ferramentas.

# 4 DESENVOLVIMENTO

Neste trabalho, buscamos integrar as ferramentas Metanome e HoloClean fazendo sua integração a partir da tradução das DCs geradas pela primeira ferramente que serão utilizadas como parâmetro de entrada para o uso da segunda ferramenta. Neste capítulo, será exemplificado todo o passo a passo do que foi realizado.

Todos os experimentos foram feitos utilizando uma máquina com o sistema operacional Linux Ubuntu 20.04.6 LTS 64 bits, com processador AMD® Ryzen 5 3500u, QuadCore e com 12GB RAM.

## 4.1 BASES DE DADOS

Para esse trabalho, foram utilizadas 5 bases de dados distintas. As bases Airport, Adult e Tax foram retiradas na plataforma do Hasso-Plattner-Institut [1], que além das bases abertas disponíveis também contem tutoriais explicativos da ferramenta Metanome. Já as bases Hospital e Met foram adquiridas no repositório aberto do HoloClean[7], que também possui um guia de uso da ferramenta. Elas estão distribuídas de acordo com a tabela 4.1:

Tabela 4.1: Bases de dados escolhidas.

| Base de dados | Número de registros | Número de colunas |
|---|---|---|
| Hospital | 1000 | 19 |
| Met | 1306 | 43 |
| Adults | 32561 | 15 |
| Airport | 81105 | 12 |
| Tax | 250000 | 15 |

As bases de dados exploradas neste trabalho abrangem diferentes domínios e estruturas. A base Hospital contém informações detalhadas sobre hospitais, incluindo identificadores como "ProviderNumber"e "HospitalName", dados de localização como endereços e códigos postais, além de características institucionais, como "HospitalType"e "HospitalOwner". Também abrange métricas de desempenho, como "Condition", "MeasureCode", "MeasureName", "Score"e "Stateavg". A base Met, que contém o maior número de colunas, traz registros sobre obras de arte do Museu Metropolitano de Arte, com atributos como "Object Number", "Title", "Artist Display Name", períodos históricos, materiais, dimensões, e informações geográficas relacionadas às peças. A base Airports reúne dados sobre aeroportos, incluindo identificadores como "local code"e "iata code", informações geográficas como "municipality", "iso region"e "coordinates", além de tipos de aeroportos ("type"). A base Adults é composta por dados de indivíduos, cobrindo aspectos demográficos e socioeconômicos. Por fim, a base Tax, com o maior número de registros, contém informações fiscais de pessoas físicas, incluindo nomes, gênero, localidade ("City", "State", "Zip"), estado civil, renda ("Salary"), taxas de impostos ("Rate") e números de dependentes, como "SingleExemp", "MarriedExemp"e "ChildExemp". Essas bases oferecem cenários diversos para validação de técnicas de limpeza e análise de dados.

## 4.2 DESCOBERTA DE DCS

Para a descoberta das Dependências de Negação, foi utilizada a ferramenta gráfica do Metanome[5]. Para seu uso, é necessário que rodemos o backend e o frontend da aplicação através dos arquivos executáveis disponíveis no repositório (.jar para o frontend, e .bat ou .sh para o backend, dependendo do sistema operacional).

Logo após, a ferramenta estará disponível na porta default, podendo ser acessada pela url `http://localhost:8080/`. Sua página inicial é apresentada como na figura 4.1.

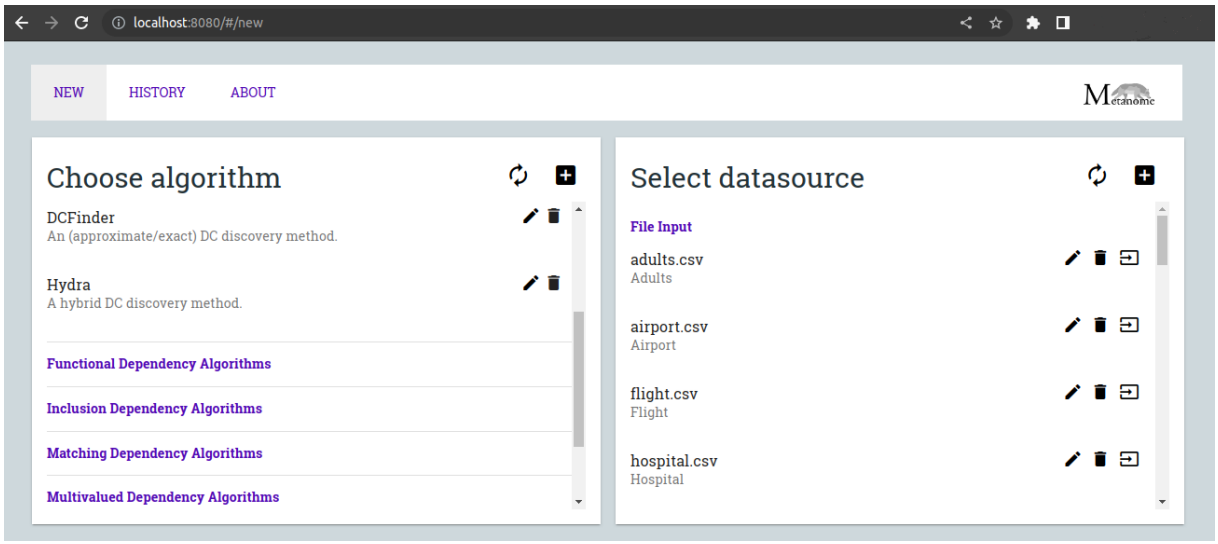



Figura 4.1: Página Inicial do Metanome.

A partir disso, podemos incluir nossos próprios arquivos de dados, sendo que temos a restrição de que o banco precisa ser relacional, e temos opções de formatos `.csv` ou `.tsv` para arquivos. Podemos também adicionar uma conexão com uma base de dados já existente caso necessário(Figura 4.2). Nesta fase, é importante preencher corretamente as informações solicitadas, como caracter separador, caractere de escape, caractere para quote, entre outros, pois elas serão utilizadas para formatar os dados no momento de realizar a busca. Para esse trabalho, todas as bases de dados utilizadas possuem formato de arquivo `.csv`.

Para os algorítmos de descoberta de restrições, caso ele não esteja entre os disponíveis ao realizar o download da ferramenta, podemos adicioná-lo à lista diretamente pelos arquivos do backend. Para isso, precisamos apenas do arquivo .jar refente ao algoritmo. O algoritmo utilizado neste trabalho, o DCFinder, adquirido através do tópico de repetibilidade do Hasso-Plattner-Institut[1].

Sobre a execução do algoritmo, basta selecionar a base de dados desejada e confirmar as configurações adicionais disponíveis na ferramenta, que definem métricas de como o algoritmo será executado. As apresentadas no Metanome são, conforme a figura 4.3 :

- *Aproximation Degree*: Define o grau de aproximação a ser utilizado na análise. Em alguns algoritmos, aproximações são usadas para lidar com grandes volumes de dados sem realizar verificações completas. Na figura 4.3, 0.01 indica que o algoritmo permite uma margem de erro de até 1% para seus cálculos.

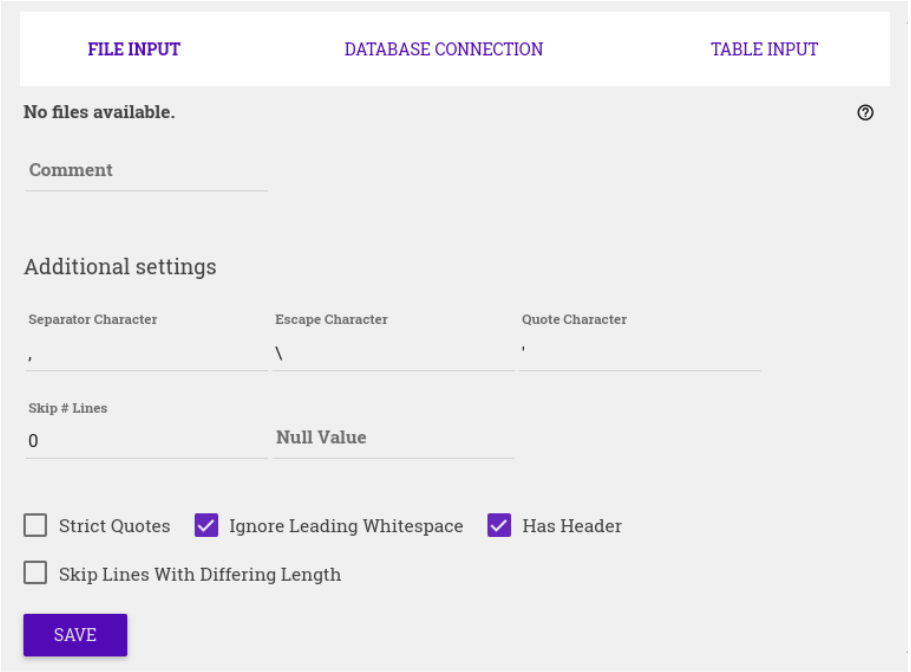



Figura 4.2: Inserção de Base de Dados.

- *Cross Column String Min Overlap*: Determina o mínimo de sobreposição entre strings para identificar dependências ou similaridades entre colunas. Na figura 4.3, 0.3 epresenta 30% de sobreposição mínima entre os valores de strings em duas colunas.
- *Chunk Lenght*: Define o tamanho dos blocos de dados a serem processados de cada vez. O valor 50000000 apresentado na figura 4.3 indica que o algoritmo processará 50 milhões de unidades (linhas ou bytes, dependendo do contexto) em cada bloco, o que pode ser útil para ajustar o consumo de memória em grandes bases de dados.
- *Buffer Lenght*: Especifica o tamanho do buffer usado durante o processamento. Na figura 4.3, o valor 5000 indica que o buffer terá capacidade para 5.000 unidades.
- *No Cross Column*: Um parâmetro booleano que desativa análises cruzadas entre colunas.

Ao finalizar, o resultado da execução fica disponível na aba de histórico, que apresenta uma lista com todas as restrições de depêndencias encontradas seguidas pelo número de predicados na expressão. Porém, o que nos importa é o arquivo gerado na pasta com caminho '/metanome/results' com seu início mostrado na figura 4.4, pois é ele que será utilizado no passo da tradução.

## 4.3 PROCESSAMENTO DOS DADOS E TRADUÇÃO

O arquivo gerado pela ferramenta Metanome que pode ser visualizado na figura 4.4 apresenta um formato JSON Lines, onde cada linha define uma restrição em termos de colunas e operadores.

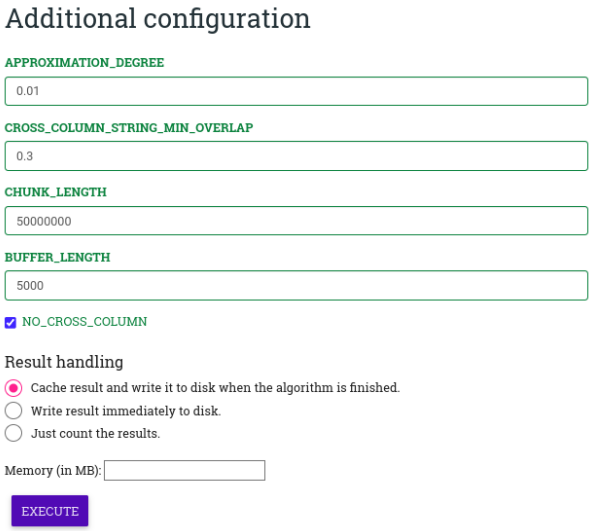



Figura 4.3: Execução do algoritmo.


Figura 4.4: Arquivo de resultado da busca de restrições pelo Metanome.

Portanto, o primeiro código em python que precisamos rodar é um que identifique as restrições e as monte no formato das expressões padrões que conhecemos. O arquivo que faz isso é o `geraDCsFormatadas.py`, que pode ser visto abaixo:

Listing 4.1: geraDCsFormatadas.py

```
1  import json
2
3  def processa_denial_constraints(input_file, output_file):
4      def formata_dc(dc):
5          predicados = dc["predicados"]
6          predicados_formatados = []
7          for pred in predicados:
8              col1 = pred["column1"]["columnIdentifier"]
9              col2 = pred["column2"]["columnIdentifier"]
10             op = pred["op"]
11             index1 = f"t{pred['index1']}.{col1}"
12             index2 = f"t{pred['index2']}.{col2}"
13
```

```
            # Map operators to the required format
            operator_map = {
               "EQUAL": "=",
               "UNEQUAL": "≠",
               "GREATER_EQUAL": "≥",
               "LESS": "<",
               "GREATER": ">",
               "LESS_EQUAL": "≤"
            }
            predicados_formatados.append(f"{index1} {operator_map[op]} {index2}")

        expressao = " ∧ ".join(predicados_formatados)
        return f"¬[{expressao}]"

    with open(input_file, 'r', encoding='utf-8') as infile:
        lines = infile.readlines()

    results = []
    for line in lines:
        dc = json.loads(line.strip())
        if dc["type"] == "DenialConstraint":
            results.append(formata_dc(dc))

    with open(output_file, 'w', encoding='utf-8') as outfile:
        outfile.write("\n".join(results))


input_file = "dcs-hospital.txt"
output_file = "dcs-hospital_formatted.txt"
processa_denial_constraints(input_file, output_file)
```

Com as DCs já em formato padrão, preferimos utilizar a métrica de succinctness para selecionar as restriçoes que serão utilizadas na fase de limpeza de dados. Para isso, utilizamos o arquivo `calculaSuccinctness.py`, que calcula a métrica para todas as DCs e o arquivo `ordena.py`, que as ordena em ordem crecente, ou seja, da mais concisa para a menos concisa.

Listing 4.2: calculaSuccinctness.py

```
import csv

# Definição de tuplas, atributos e operadores
TUPLAS = {"t0", "t1"} # Liste todas as tuplas possíveis

ATRIBUTOS = {
    "column1","column2","column3","column4","column5",
    "column6","column7","column8","column9","column10"
    ,"column11","column12","column13","column14","column15"
} # Liste todos os atributos possíveis (adults.csv)

OPERADORES = {"=", ">", "<", "≠", "≤", "≥"} # Operadores permitidos

def calculate_dc_length(dc):
    """
    Calcula o comprimento de uma DC com base em elementos pré-definidos.
    """
    elements = set()

```

```
    # Verificar se cada tupla está presente na DC
    for t in TUPLAS:
        if t in dc:
            elements.add(t)

    # Verificar se cada atributo está presente na DC
    for attr in ATRIBUTOS:
        if attr in dc:
            elements.add(attr)

    # Verificar se cada operador está presente na DC
    for op in OPERADORES:
        if op in dc:
            elements.add(op)

    return len(elements)

def calculate_succinctness(dcs):
    """
    Calcula a Succinctness para cada DC no conjunto.
    """
    lengths = [calculate_dc_length(dc) for dc in dcs]
    min_length = min(lengths) # O menor tamanho entre as DCs
    succinctness = [min_length / length for length in lengths]
    return succinctness, lengths

def process_dc_file(input_file, output_file):
    """
    Lê um arquivo de DCs, calcula a Succinctness e o Length de cada uma,
    e salva em um novo arquivo CSV.
    """
    with open(input_file, 'r') as f:
        dcs = [line.strip() for line in f.readlines() if line.strip()]

    # Calcular Succinctness e Length para cada DC
    succinctness_values, lengths = calculate_succinctness(dcs)

    # Escrever os resultados em um arquivo CSV
    with open(output_file, 'w', newline='') as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(["DC", "Length", "Succinctness"]) # Cabeçalhos
        for dc, length, succinctness in zip(dcs, lengths, succinctness_values):
            writer.writerow([dc, length, succinctness])

# Arquivos de entrada e saída
input_file = "dcs-adults_formatted.txt" # Arquivo de DCs no formato texto
output_file = "adults_succinctness.csv" # Resultado com Succinctness

# Executar o processamento
process_dc_file(input_file, output_file)
```

Listing 4.3: ordena.py

```
import csv

def sort_csv_by_length(input_file, output_file):
    # Lê o arquivo CSV
    with open(input_file, 'r') as csvfile:
```

```
        reader = csv.reader(csvfile)
        header = next(reader) # Lê o cabeçalho
        rows = list(reader) # Lê o restante das linhas

    # Ordena as linhas com base na coluna Length (índice 1)
    rows_sorted = sorted(rows, key=lambda row: int(row[1]))

    # Escreve o resultado no arquivo de saída
    with open(output_file, 'w', newline='') as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(header) # Escreve o cabeçalho
        writer.writerows(rows_sorted) # Escreve as linhas ordenadas

# Arquivo de entrada e saída
input_file = "tax_succinctness.csv"
output_file = "tax_succinctness_ordenado.csv"

# Executa a ordenação
sort_csv_by_length(input_file, output_file)
```

Por fim, já temos as restrições de negação que iremos utilizar na etapa de limpeza com o HoloClean, porém, para que a ferramenta seja executada, precisamos que essas restrições estajam no formato DDlog, como já exemplificado em capítulo anteriores. Para isso, utilizamos o arquivo `traduzDDlog.py`, que recebe as restrições definidas e gera um novo arquivo (Figura 4.5) que será utilizado como entrada da ferramenta de limpeza de dados.

Listing 4.4: traduzDDlog.py

```
import csv

def translate_to_holoclean(input_file, output_file, num_dcs=30):
    holoclean_rules = []

    # Função para traduzir uma DC para o formato Holoclean
    def dc_to_holoclean(dc):
        # Remover os delimitadores ¬[] e dividir os predicados
        predicates = dc.strip("¬[]").split(" ∧ ")
        holoclean_parts = ["t1&t2"] # Prefixo fixo para cada regra

        for predicate in predicates:
            # Identificar operador e atributos
            if "≠" in predicate:
                attr1, attr2 = map(str.strip, predicate.split("≠"))
                holoclean_parts.append(f"IQ({attr1},{attr2})")
            elif "=" in predicate:
                attr1, attr2 = map(str.strip, predicate.split("="))
                holoclean_parts.append(f"EQ({attr1},{attr2})")
            elif ">" in predicate:
                attr1, attr2 = map(str.strip, predicate.split(">"))
                holoclean_parts.append(f"GT({attr1},{attr2})")
            elif "<" in predicate:
                attr1, attr2 = map(str.strip, predicate.split("<"))
                holoclean_parts.append(f"LT({attr1},{attr2})")
            elif "≥" in predicate:
                attr1, attr2 = map(str.strip, predicate.split("≥"))
                holoclean_parts.append(f"GE({attr1},{attr2})")
            elif "≤" in predicate:
```

```
            attr1, attr2 = map(str.strip, predicate.split("≤"))
            holoclean_parts.append(f"LE({attr1},{attr2})")

        # Combinar as partes em uma regra Holoclean
        return "&".join(holoclean_parts)

    # Ler o arquivo CSV
    with open(input_file, 'r') as csvfile:
        print("Traduzindo as primeiras", num_dcs, "DCs do arquivo",
        input_file)
        reader = csv.DictReader(csvfile)
        for i, row in enumerate(reader):
            if i >= num_dcs: # Limitar ao número de DCs especificado
                break
            dc = row['DC']
            holoclean_rule = dc_to_holoclean(dc)
            holoclean_rules.append(holoclean_rule)

    # Escrever as regras traduzidas no arquivo de saída
    with open(output_file, 'w') as outfile:
        print("Escrevendo as regras traduzidas no arquivo", output_file)
        outfile.write("\n".join(holoclean_rules))

# Arquivos de entrada e saída
input_file = "tax_succinctness_ordenado.csv"
output_file = "ddlog_tax.txt"

# Executar a tradução
translate_to_holoclean(input_file, output_file)
```

```
t1&t2&EQ(t1.Sample,t2.Sample)&EQ(t1.HospitalName,t2.HospitalName)
t1&t2&EQ(t1.PhoneNumber,t2.PhoneNumber)&EQ(t1.MeasureCode,t2.MeasureCode)
t1&t2&EQ(t1.Sample,t2.Sample)&EQ(t1.ZipCode,t2.ZipCode)
t1&t2&EQ(t1.Address1,t2.Address1)&EQ(t1.MeasureName,t2.MeasureName)
t1&t2&EQ(t1.HospitalName,t2.HospitalName)&EQ(t1.Score,t2.Score)
t1&t2&EQ(t1.MeasureCode,t2.MeasureCode)&EQ(t1.Address1,t2.Address1)
t1&t2&EQ(t1.CountyName,t2.CountyName)&EQ(t1.MeasureName,t2.MeasureName)
t1&t2&EQ(t1.PhoneNumber,t2.PhoneNumber)&EQ(t1.MeasureName,t2.MeasureName)
t1&t2&EQ(t1.ZipCode,t2.ZipCode)&EQ(t1.Score,t2.Score)
t1&t2&EQ(t1.Sample,t2.Sample)&EQ(t1.MeasureName,t2.MeasureName)
t1&t2&EQ(t1.Score,t2.Score)&EQ(t1.MeasureName,t2.MeasureName)
t1&t2&EQ(t1.Sample,t2.Sample)&EQ(t1.City,t2.City)
t1&t2&EQ(t1.CountyName,t2.CountyName)&EQ(t1.Score,t2.Score)
t1&t2&EQ(t1.HospitalOwner,t2.HospitalOwner)&EQ(t1.MeasureName,t2.MeasureName)
t1&t2&EQ(t1.ProviderNumber,t2.ProviderNumber)&EQ(t1.Score,t2.Score)
t1&t2&EQ(t1.Condition,t2.Condition)&EQ(t1.CountyName,t2.CountyName)
t1&t2&EQ(t1.HospitalName,t2.HospitalName)&EQ(t1.MeasureName,t2.MeasureName)
t1&t2&EQ(t1.MeasureCode,t2.MeasureCode)&EQ(t1.City,t2.City)
t1&t2&EQ(t1.Sample,t2.Sample)&EQ(t1.Address1,t2.Address1)
t1&t2&EQ(t1.PhoneNumber,t2.PhoneNumber)&EQ(t1.Score,t2.Score)
t1&t2&EQ(t1.MeasureCode,t2.MeasureCode)&EQ(t1.Score,t2.Score)
t1&t2&EQ(t1.ZipCode,t2.ZipCode)&EQ(t1.MeasureName,t2.MeasureName)
t1&t2&EQ(t1.MeasureCode,t2.MeasureCode)&EQ(t1.ZipCode,t2.ZipCode)
t1&t2&EQ(t1.City,t2.City)&EQ(t1.MeasureName,t2.MeasureName)
t1&t2&EQ(t1.MeasureCode,t2.MeasureCode)&EQ(t1.HospitalName,t2.HospitalName)
t1&t2&EQ(t1.Sample,t2.Sample)&EQ(t1.HospitalOwner,t2.HospitalOwner)
t1&t2&EQ(t1.ProviderNumber,t2.ProviderNumber)&EQ(t1.MeasureName,t2.MeasureName)
```

Figura 4.5: Resultado do algoritmo de tradução.

## 4.4 LIMPEZA DE BASE DE DADOS USANDO O HOLOCLEAN

Para a fase de limpeza das 5 bases de dados, utilizamos os arquivos de texto em formato DDlog com as restrições e a própria base de dados como entrada. O código utilizado para executar uma sessão da ferramenta foi o `holoclean.py`. A seguir, detalhamos cada etapa do fluxo, que consiste em:

Listing 4.5: holoclean.py

```
import sys
sys.path.append('../')
import holoclean
from detect import NullDetector, ViolationDetector
from repair.featurize import *


# 1. Setup a HoloClean session.
hc = holoclean.HoloClean(
    db_name='holo',
    domain_thresh_1=0,
    domain_thresh_2=0,
    weak_label_thresh=0.99,
    max_domain=10000,
    cor_strength=0.6,
    nb_cor_strength=0.8,
    epochs=10,
    weight_decay=0.01,
    learning_rate=0.001,
    threads=1,
    batch_size=1,
    verbose=True,
    timeout=3*60000,
    feature_norm=False,
    weight_norm=False,
    print_fw=True
).session

# 2. Load training data and denial constraints.
hc.load_data('adults', '../testdata/AdultFull.csv')
hc.load_dcs('../testdata/adult_constraints.txt')
hc.ds.set_constraints(hc.get_dcs())

# 3. Detect erroneous cells using these two detectors.
detectors = [NullDetector(), ViolationDetector()]
hc.detect_errors(detectors)

# 4. Repair errors utilizing the defined features.
hc.setup_domain()
featurizers = [
    InitAttrFeaturizer(),
    OccurAttrFeaturizer(),
    FreqFeaturizer(),
    ConstraintFeaturizer(),
]

hc.repair_errors(featurizers)


# 5. Evaluate the correctness of the results.
hc.evaluate(fpath='../testdata/AdultFull.csv',
          tid_col='tid',
          attr_col='attribute',
          val_col='correct_val')
```

1. **Configuração**: Define os parâmetros para análise e reparo.

2. **Carregamento**: Importa os dados e restrições de negação.
3. **Detecção**: Identifica células errôneas usando detectores.
4. **Reparo**: Corrige os valores com base em featurizadores e aprendizado de máquina.
5. **Avaliação**: Compara os resultados com dados de referência para medir a eficácia.

### 4.4.1 Configuração do HoloClean

```
hc = holoclean.HoloClean(
    db_name='holo',
    domain_thresh_1=0,
    domain_thresh_2=0,
    weak_label_thresh=0.99,
    max_domain=10000,
    cor_strength=0.6,
    nb_cor_strength=0.8,
    epochs=10,
    weight_decay=0.01,
    learning_rate=0.001,
    threads=1,
    batch_size=1,
    verbose=True,
    timeout=3*60000,
    feature_norm=False,
    weight_norm=False,
    print_fw=True
).session
```

Esta etapa configura os parâmetros para a sessão do HoloClean, que definem o comportamento do sistema durante a detecção e reparo de erros. Os principais parâmetros são:

- **db_name='holo'**: Nome do banco de dados onde os dados e modelos serão armazenados. (Precisa ser criado previamente)
- **domain_thresh_1** e **domain_thresh_2**: Definem os limiares para geração de domínios (valores possíveis para os atributos).
- **weak_label_thresh=0.99**: Limiar para selecionar rótulos fracos (valores candidatos).
- **epochs=10**: Número de épocas para o treinamento do modelo de aprendizado de máquina.
- **learning_rate=0.001**: Taxa de aprendizado usada no treinamento.
- **timeout=3*60000**: Limite de tempo para execução do HoloClean.

### 4.4.2 Carregamento de Dados e Restrições

```
hc.load_data('hospital', '../testdata/hospital.csv')
hc.load_dcs('../testdata/ddlog_hospital.txt')
hc.ds.set_constraints(hc.get_dcs())
```

Nesta etapa, os dados a serem analisados (`hospital.csv`) e as restrições de negação (`ddlog_hospital.txt`) são carregados. As restrições ajudam a identificar inconsistências entre colunas e valores. Os passos incluem:

- **hc.load_data('hospital', ...)**: Associa o nome lógico `hospital` ao arquivo de dados.
- **hc.load_dcs(...)**: Carrega as restrições de negação no formato DDLOG.
- **hc.ds.set_constraints(...)**: Configura as restrições no sistema para uso durante a análise.

### 4.4.3 Detecção de Erros

```
detectors = [NullDetector(), ViolationDetector()]
hc.detect_errors(detectors)
```

Nesta etapa, detectores são utilizados para identificar células errôneas na tabela. Os principais detectores são:

- **NullDetector()**: Detecta células com valores nulos ou ausentes.
- **ViolationDetector()**: Identifica células que violam as restrições de negação carregadas.

### 4.4.4 Reparo de Erros

```
hc.setup_domain()
featurizers = [
    InitAttrFeaturizer(),
    OccurAttrFeaturizer(),
    FreqFeaturizer(),
    ConstraintFeaturizer(),
]
hc.repair_errors(featurizers)
```

Esta etapa define o domínio dos valores possíveis para as células errôneas (`hc.setup_domain()`) e usa featurizadores para criar representações utilizadas no treinamento do modelo de reparo. Os principais featurizadores são:

- **InitAttrFeaturizer()**: Inicializa atributos e valores candidatos.
- **OccurAttrFeaturizer()**: Adiciona características baseadas na ocorrência de atributos.
- **FreqFeaturizer()**: Baseia-se na frequência dos valores no dataset.

- **ConstraintFfeaturizer()**: Usa as restrições de negação como características para prever os reparos.

O comando `hc.repair_errors(...)` repara os valores errôneos utilizando as características geradas pelos featurizadores.

### 4.4.5 Avaliação

```
hc.evaluate(fpath='../testdata/hospital_clean.csv',
            tid_col='tid',
            attr_col='attribute',
            val_col='correct_val')
```

Nesta etapa, os reparos realizados são avaliados comparando os resultados com um arquivo de referência (`hospital_clean.csv`). O arquivo de avaliação deve conter as seguintes colunas:

- **tid**: Identificador único da tupla (linha).
- **attribute**: Nome do atributo avaliado.
- **correct_val**: Valor correto esperado para o atributo.

Podem ser realizadas outras formas de validação para comparação entre as bases de dados original e a suja que recebeu a limpeza, mas para esse trabalho, foi mantido o formato utilizado no tutorial [7] para facilitar a análise dos resultados, pois o método *evaluate* já utiliza esse formato.

## 4.5 DETALHES DE IMPLEMENTAÇÃO

Para as bases de dados Hospital e Met, os arquivos `.csv` que contém o banco com inconsistencias e o banco limpo, que são utilizados para comparação, foram adquiridos através do repositório do tutorial do HoloClean. O mesmo foi feito com suas restrições, pois foram utilizadas restrições definidas no repositório e as geradas através do Metanome.

Já para as bases de dados Adults, Airport e Tax, as restrições utilizadas foram somente as geradas pelo Metanome, enquanto a base considerada suja foi intencionalmente manipulada, onde foram inseridos valores inconsistentes (duplicatas, inconsistencias em valores chave) e campos nulos.

# 5 RESULTADOS

Ao terminar sua execução, a ferramenta gera um arquvo de log que apresenta diversos resultados e métricas para as fases de descoberta de células com erros e para a fase de reparo de dados.

Começando pela descoberta das células inconsistentes, temos métricas de *precision*, *recall* e *F1-score* para cada base de dados. Para as bases Hospital e Met, que já possuem restrições definidas no repositório do tutorial do HoloClean, comparamos as métricas adquiridas através dessas restrições já definidas (valores em azul) com os valores adquiridos pela integração com o Metanome (valores em laranja). Essas métricas podem ser vistos nos gráficos das figutas 5.1 e 5.2

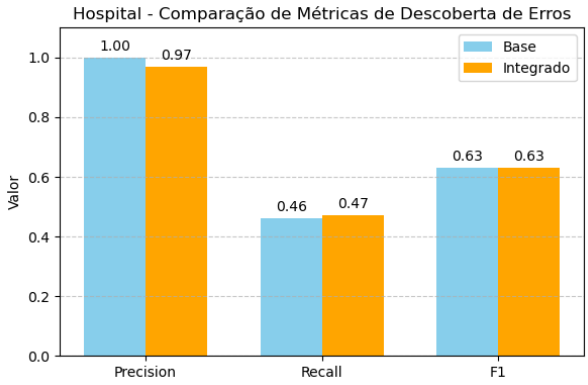



Figura 5.1: Métricas para detecção de células inconsistentes - Hospital.

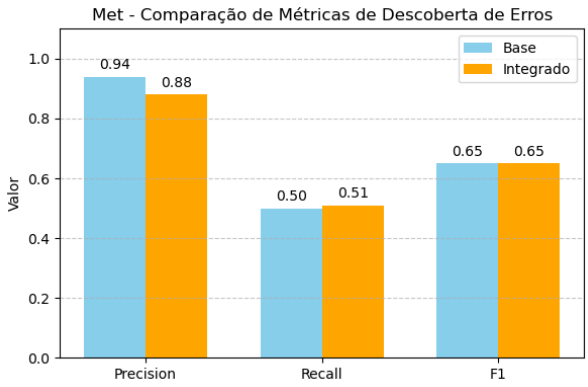



Figura 5.2: Métricas para detecção de células inconsistentes - Met.

Tendo em vista esses resultados, percebemos que ambos os conjuntos de restrição obtiveram valores identicos de *F1-score*, porém para a base Met, sua métrica de precision foi bastante inferior, ou seja, células que o HoloClean identificou como inconsistentes na verdade já possuíam valor correto.

Já para as bases de dados Adult, Airport e Tax, todo o processo foi aplicado igualmente, porém utilizando somente as restrições descobertas através do Metanome, e seus resultados para a descoberta de células inconsistentes podem ser vistos nas figuras 5.3, 5.4 e 5.5.

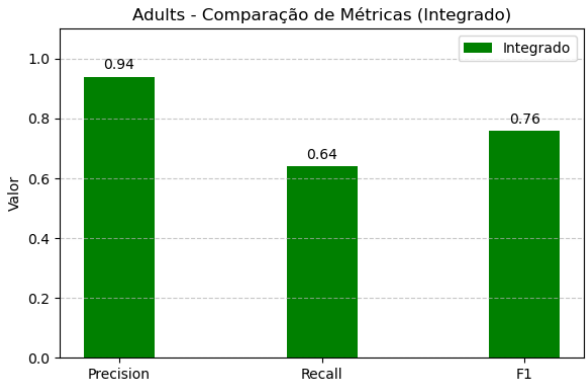



Figura 5.3: Métricas para detecção de células inconsistentes - Adults.

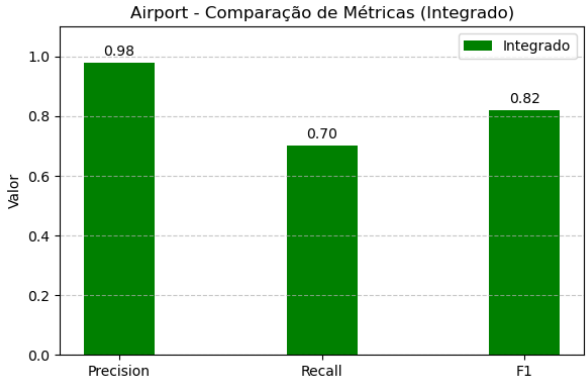



Figura 5.4: Métricas para detecção de células inconsistentes - Airport.

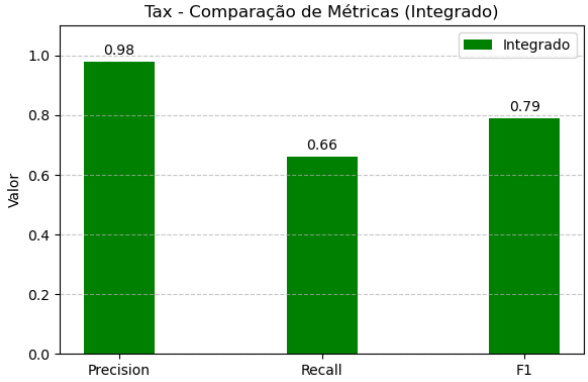



Figura 5.5: Métricas para detecção de células inconsistentes - Tax.

Com esses resultados podemos notar que a descoberta de células com inconsistências foi mais precisa, tivemos um valor de *precision* bem alto em todas as bases de dados, que por sinal possuem menos colunas. E mesmo possuindo um número muito maior de linhas, dados que podem ser vistos na Tabela 4.1, esse fato não teve muita influência na descoberta dessas células.

Agora sobre os resultados que foram obtidos na fase de reparo das bases de dados: gráfico da figura 5.6 apresenta métricas de avaliação de *F1-score* para as bases Hospital e Met. Em sequência, temos o gráfico da figura 5.7 que indica a métrica

para as bases Adult, Airport e Tax, utilizando os conjuntos de restrições geradas pelo Metanome.

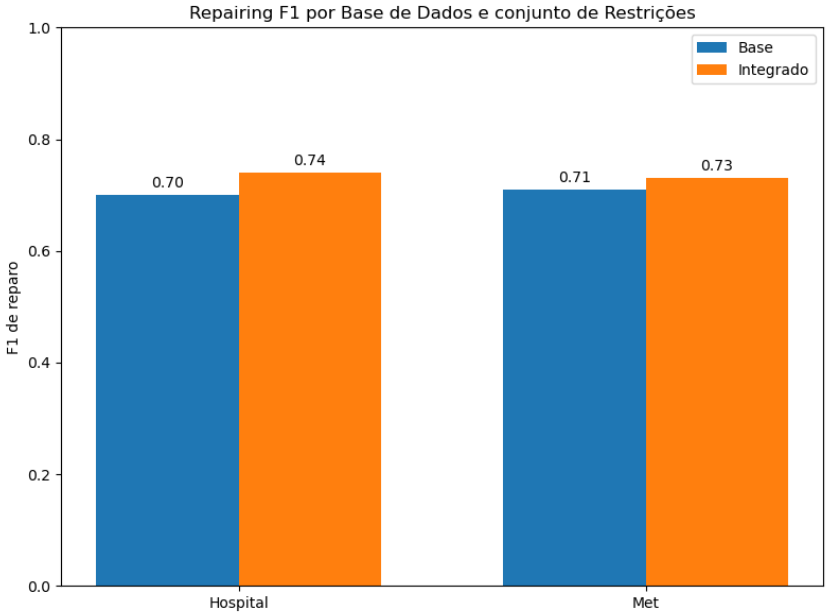



Figura 5.6: Métricas de F1-score para a fase de reparo - Hospital, Met.

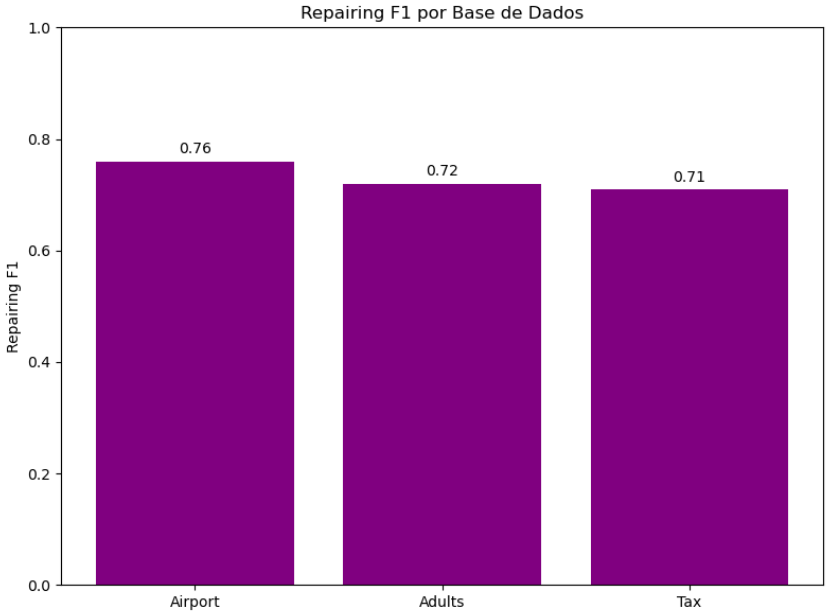



Figura 5.7: Métricas de F1-score para a fase de reparo - Adults, Airport, Tax.

Verificamos através dos gráficos que o reparo para as bases Hospital e Met por meio do conjunto de restrições obtido pelo Metanome, pré-processado, e definido por métricas obteve um resultado levemente melhor do que com o conjunto de restrições do repositório Holoclean.

Já sobre as demais bases, consguimos adquirir resultados satisfatórios, todos ultrapassando 0.70 de *F1-Score*, o que é o padrão para a ferramenta Holoclean, que possui uma média de 0.80 para essa mesma métrica [7].

# 6 CONCLUSÃO

Neste trabalho, foi apresentada uma forma de integração entre as ferramentas Metanome e Holoclean para realizar a limpeza de dados utilizando restrições de negação. O fluxo de trabalho proposto mostrou-se eficaz, desde a descoberta de restrições no Metanome até a tradução para o formato DDlog e posterior aplicação no Holoclean. Os resultados demonstraram que, em todos os casos, a qualidade dos dados melhorou significativamente após o processo de limpeza, com métricas de F1 superiores a 0,70 e, em alguns cenários, resultados levemente superiores aos obtidos com restrições predefinidas no repositório do Holoclean.

O fluxo desenvolvido se destacou por sua capacidade de lidar com bases de dados de diferentes tamanhos e complexidades, o que reforça sua aplicabilidade em cenários reais e heterogêneos. Além disso, a integração facilitada entre as ferramentas provou ser um avanço importante para automatizar etapas que frequentemente exigem intervenções manuais e detalhadas.

Como trabalho futuro, poderia ser feito o desenvolvimento de uma interface gráfica que automatize e simplifique ainda mais todo o processo de integração entre Metanome e Holoclean. Essa interface poderia incluir funcionalidades como o carregamento direto de bases de dados, configuração de parâmetros, escolha de métricas e visualização dos resultados, tornando a ferramenta acessível a um público mais amplo e menos técnico. Assim, esta proposta visaria expandir o alcance e a usabilidade das técnicas de limpeza de dados baseadas em restrições de negação.

# REFERÊNCIAS


[1] Hasso-Plattner (HPI). Metanome tool and profiling algorithms. https://hpi.de/naumann/projects/data-profiling-and-analytics/metanome-data-profiling/algorithms.html. 2018.

[2] Eduardo H. M. Pena, Edson R. Lucas Filho, Eduardo C. de Almeida, and Felix Naumann. Efficient detection of data dependency violations. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, CIKM '20, page 1235–1244, New York, NY, USA, 2020. Association for Computing Machinery.

[3] Eduardo H. M. Pena, Eduardo C. de Almeida, and Felix Naumann. Discovery of approximate (and exact) denial constraints. *Proc. VLDB Endow.*, 13(3):266–278, November 2019.

[4] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. Functional dependency discovery: an experimental evaluation of seven algorithms. *Proc. VLDB Endow.*, 8(10):1082–1093, June 2015.

[5] Thorsten Papenbrock, Tanja Bergmann, Moritz Finke, Jakob Zwiener, and Felix Naumann. Data profiling with metanome. *Proc. VLDB Endow.*, 8(12):1860–1863, August 2015.

[6] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. Discovering denial constraints. *Proc. VLDB Endow.*, 6(13):1498–1509, August 2013.

[7] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. Holoclean: holistic data repairs with probabilistic inference. *Proc. VLDB Endow.*, 10(11):1190–1201, August 2017.

[8] Eduardo H. M. Pena, Eduardo C. de Almeida, and Felix Naumann. Fast detection of denial constraint violations. *Proc. VLDB Endow.*, 15(4):859–871, December 2021.

[9] Laure Berti-Équille, Hazar Harmouch, Felix Naumann, Noël Novelli, and Saravanan Thirumuruganathan. Discovery of genuine functional dependencies from relational data with missing values. *Proc. VLDB Endow.*, 11(8):880–892, April 2018.

[10] Eduardo H. M. Pena, Fabio Porto, and Felix Naumann. Fast algorithms for denial constraint discovery. *Proc. VLDB Endow.*, 16(4):684–696, December 2022.