Capítulo 8

Big Data Linkage no Brasil: Aspectos metodológicos e práticos

Robespierre Pita (IC-UFBA/Cidacs-Fiocruz), Roberto P. Carreiro (Cidacs-Fiocruz), Carlos J. C. Santos (IC-UFBA/Cidacs-Fiocruz), Laianne dos S. Protasio (IC-UFBA), Marcos E. Barreto (London School of Economics and Political Science/Cidacs-Fiocruz), Victor B. Orrico (IC-UFBA), José A. D. Gomes (Cidacs-Fiocruz), Fernanda S. Eustáquio (Cidacs-Fiocruz), Samila Sena (Cidacs-Fiocruz), Mauricio L. Barreto (Cidacs-Fiocruz), Pablo I. P. Ramos (Cidacs-Fiocruz), Denis Rangel (Cidacs-Fiocruz), Bethânia de A. Almeida (Cidacs-Fiocruz)

Abstract

Big Data Linkage (BDL) is a key step in the integration of large-scale datasets, addressing the challenges and solutions related to the management and linkage of diverse, high-dimensional records. This short course aims to present the fundamental concepts of BDL, with a special focus on the Brazilian context. Key methodological challenges—such as accuracy, scalability, and privacy preservation—are discussed, highlighting tools developed in Brazil, such as AtyImo and CIDACS-RL. The course also explores current research gaps, opportunities for further investigation, and emerging initiatives that are driving progress in the BDL field.

Resumo

O Big Data Linkage (BDL) é uma etapa essencial na integração de grandes volumes de dados, lidando com os desafios e soluções relacionados à gestão e ao cruzamento de registros diversos e de alta dimensionalidade. Este minicurso tem como objetivo apresentar os conceitos fundamentais do BDL, com ênfase no contexto brasileiro. São discutidos os principais desafios metodológicos — como acurácia, escalabilidade e preservação da privacidade —, com destaque para ferramentas desenvolvidas no Brasil, como o AtyImo e o CIDACS-RL. O minicurso também aborda lacunas atuais, oportunidades de pesquisa e as iniciativas emergentes que vêm impulsionando o avanço da área.

8.1. Introdução

A Integração de Dados (ID) abriga uma classe de aplicações científicas com inequívoco potencial para unificar informações provenientes de múltiplas fontes, permitindo análises mais robustas [Dong and Srivastava 2013]. O Record Linkage (RL) consiste numa classe de soluções computacionais capazes de vincular registros de uma mesma entidade que se encontram em bases de dados diferentes [Christen 2019]. Estas bases de dados, que frequentemente dão suporte a sistemas de informação, formulados para atender propósitos distintos, não compartilham uma chave única capaz de identificar estas entidades.

Considere duas bases de dados fictícias do domínio da saúde. A Base de Dados A (BD_A), ilustrada na Tabela 1, tem propósito e variáveis similares ao Sistema de Informação que suporta a notificação compulsória dos casos de tuberculose em qualquer estabelecimento de saúde (SINAN-TB). Esta fonte de dados administrativos tem o potencial de habilitar diversas análises para auxiliar o planejamento da saúde, monitorar surtos e emergências em saúde, políticas interventórias, além da avaliação do impacto destas intervenções. A Tabela 1 reúne cinco registros fictícios, atribuídos a pessoas distintas, descrevendo eventos de notificação. As variáveis nome, sexo, mãe, dn e cidade podem ser consideradas quase identificadoras (QI). As variáveis categóricas hemoptise e dispneia suportam a caracterização dos casos suspeitos, podendo assumir os valores 1= 'Sim', 2= 'Não', e 9= 'Ignorado'. Os valores no atributo cid correspondem aos códigos da Classificação Internacional de Doenças, utilizados para classificar e codificar doenças investigadas para cada caso. Por fim, a variável criterio indica qual foi o método utilizado para confirmar ou descartar o caso suspeito. Os valores possíveis para esta variável são 1=Laboratorial, 2=Clínico-epidemiológico, ou 3=Por imagem.

Tabela 8.1. Base de dados sintética A, simulando o Sistema de Informação de notificação e agravo da tuberculose (SINAN-TB)

nome	sexo	mae	dn	hemoptise	dispneia	cid10	criterio
KEIDSON RIBEIRO	1	MARIA SANTOS	2008-02-18	2	1	A15.0	3
CAMILA SARAIVA	2	KAREN DA SILVA	2007-01-30	1	1	None	1
VINICIUS DA COSTA	1	MAGNA LOPEZ	2007-05-17	2	1	A16.0	None
VIVIAN GOMES	2	ARLENE LABORDA	2006-12-18	2	2	A15.0	2
MYCHEL OLIVEIRA	1	ANA OLIVEIRA	2007-05-10	2	2	None	4

A Base de Dados B (BD_B), por sua vez, simula os registros do Sistema de Informação de Mortalidade (SIM). Além dos QI, a Tabela 2 apresenta outros dois atributos para BDB, o circobito e causabas. Os valores possíveis para circobito denotam as circunstâncias do óbito, podendo ser 1=Natural, 2=Acidente, 3=Suicídio, 4=Homicídio, e 5=Outros. Por sua vez, o CID10 que representa a causa básica da morte é armazenado em causabas. Esta fonte de dados tem o potencial de suportar a construção de indicadores de mortalidade geral, mortalidade infantil, tendências temporais ou impacto de epidemias.

Uma integração bem sucedida das bases BD_A e BD_B pode habilitar tarefas analíticas ainda mais robustas, tais como o perfil populacional associado à morbimortalidade da tuberculose, o risco de morte pela doença dos diferentes estratos demográficos, o impacto de políticas públicas e o comportamento espacial ou temporal dos desfechos. Considerando que não existe uma chave que identifique os indivíduos existentes nestas duas bases

Tabela 8.2. Base de dados sintética B, simulando o Sistema de Informação de Mortalidade (SIM)

nome	sexo	mae	dn	circobito	causabas
JULIA S LIMA	M	MARIA SANTOS	9-out.-2007	2	3
VIVIAN GOMES	F	ARLENE LABORDA	8-dez.-2006	2	2
RN DE MAGNA	F	MAGNA LOPEZ	15-jun.-2008	1	2
KEIDSON R	M	MARIA R SANTOS	18-fev.-2008	1	3
KAMILA S SARAIVA	F	KAREN SILVA	30-jan.-2007	5	1

de dados, o *pipeline* de RL deve cumprir seis etapas fundamentais. A primeira etapa consiste na seleção das variáveis QI capazes de identificar de forma unívoca os indivíduos e que estão presentes nas duas bases de dados envolvidas. Em seguida, diversas estratégias de pré-processamento são utilizadas para aprimorar a qualidade dos dados envolvidos.

A próxima etapa é a indexação/blocagem das bases com o intuito de reduzir a quantidade de comparações entre os registros dos bancos. A quarta consiste na comparação par-a-par através de medidas de similaridade das variáveis selecionadas para representar o indivíduo. Na próxima etapa, cada par comparado é classificado como i) "provável ligação", que possivelmente correspondem a mesma entidade, correspondentes, não correspondentes ou potenciais correspondentes, de acordo com o valor de similaridade, ii) "ligação em potencial", que alcançaram grau de semelhança ou probabilidade suficiente para que sejam submetidos a uma nova rodada de comparação (frequentemente manual), e iii) "provável não-ligação", que possivelmente não correspondem à mesma entidade. Por fim, a sexta etapa agrupa os esforços para avaliação da qualidade do RL.

Segundo [Harron et al. 2016], o desenvolvimento e pesquisa em RL são ameaçados por quatro desafios. O principal desafio metodológico associado com a minimização dos erros de vinculação ocasionados pelo preenchimento imperfeito ou incorreto dos atributos QI. O segundo desafio consiste na dificuldade em endereçar os requisitos trazidos pelo contexto do Big Data. Em terceiro lugar, a autora aponta para as adversidades de manter um plano de governança de dados e engajamento público alinhados com os mais novos desdobramentos e técnicas disponíveis na literatura. Por fim, o quarto desafio se refere a escassez de acesso a dados identificados que impacta diretamente na formação de novos pesquisadores e analistas capazes de executar estas vinculações. As seções a seguir explora os dois primeiros desafios mencionados, focando em iniciativas brasileiras que implementam soluções de RL capazes de produzir grandes volumes de dados integrados com alta qualidade.

Este capítulo tem o objetivo de discutir as particularidades metodológicas e tecnológicas das tarefas de integração de coleções massivas de registros que demandam *Big Data Linkage* (BDL), com foco especial nas experiências e soluções nacionais. Será feita uma discussão sobre a evolução das ferramentas brasileiras para integrar bases de dados administrativas, suportando a pesquisa, tomada de decisão ou a formulação de políticas públicas. Entre estas iniciativas nacionais, destacam-se as duas soluções implementadas no escopo do Centro de Integração de Dados para a Saúde (CIDACS/Fiocruz). Desde sua criação em 2016, o CIDACS desenvolveu uma infraestrutura robusta, além de frameworks de governança e compartilhamento de dados e protocolos operacionais para a aquisição, gestão e vinculação de registros eletrônicos de saúde e sociais de abrangência nacional,

coletados administrativamente no Brasil. Como fruto desta evolução, o CIDACS, em parceria com a UFBA, lançou o AtyImo [Pita et al. 2018], uma ferramenta baseada em Apache Spark capaz de prover preservação de privacidade, integrando as primeiras bases de dados da Coorte de 100 Milhões de Brasileiros [Barreto et al. 2022]. A segunda ferramenta, o CIDACS-RL [Barbosa et al. 2020], implementa uma sofisticada estratégia de indexação baseada em Elasticsearch para diminuir drasticamente o custo computacional da comparação par-a-par.

8.2. Big Data Linkage

A arquitetura da Integração de Dados (ID) [Dong and Srivastava 2013] tradicional inclui três principais componentes, chamados *Schema Alignment* (SA), *Record Linkage* (RL) e *Data Fusion* (DF). As estratégias compreendidas no SA têm o objetivo de mediar a modelagem nas bases de dados de mesmo domínio que estão envolvidas no ID. As soluções computacionais no componente de RL aplicam técnicas de classificação para identificar registros que correspondem a uma mesma entidade, indivíduo ou evento em mais de uma base de dados [Christen 2019]. Por fim, os métodos do DF produzem datamarts contendo o subconjunto dos valores de interesse de cada entidade integrada. Com foco em suportar processos analíticos cada vez mais sofisticados, pesquisadores das áreas de saúde pública, estatística e computação se dedicam a elaborar métodos de RL capazes de endereçar diversos desafios relacionados à gestão de dados, desempenho e preservação de privacidade [Dunn and Chief 1946, Harron et al. 2016].

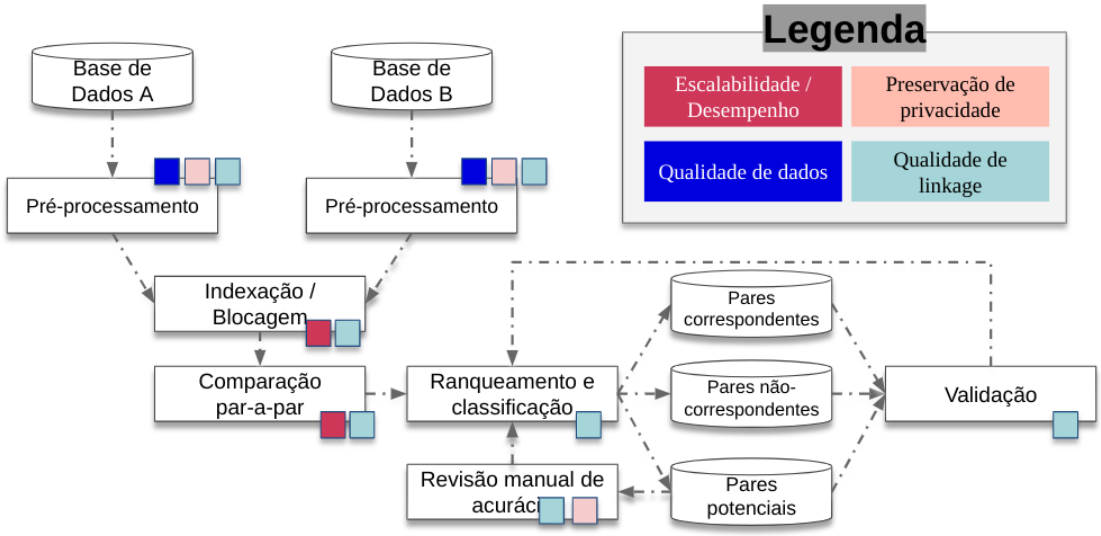

Figura 8.1. Fluxo genérico de Integração de Dados e seus desafios associados

Segundo [Harron et al. 2016], o desenvolvimento e pesquisa em RL são ameaçados por quatro desafios. O principal desafio metodológico associado com a minimização dos erros de vinculação ocasionados pelo preenchimento imperfeito ou incorreto dos atributos QI. O segundo desafio consiste na dificuldade em endereçar os requisitos trazidos pelo contexto do Big Data. Em terceiro lugar, a autora aponta para as adversidades de

manter um plano de governança de dados e engajamento público alinhados com os mais novos desdobramentos e técnicas disponíveis na literatura. Por fim, o quarto desafio se refere a escassez de acesso a dados identificados que impacta diretamente na formação de novos pesquisadores e analistas capazes de executar estas vinculações. Endereçar estes desafios ao longo do fluxo de ID, como ilustrado na Figura 8.2, requer a incorporação de algorítmos otimizados e adequados para a distribuição de dados envolvidas no processo.

O *Big Data Linkage* (BDL) refere-se ao processo de conectar registros de múltiplas fontes de dados em larga escala, garantindo que informações relacionadas a uma mesma entidade sejam corretamente identificadas [Dong and Srivastava 2013]. Diferente da vinculação tradicional, que lida com um número limitado de registros armazenados utilizando o modelo estruturado, o linkage em Big Data deve ser capaz de processar milhões de fontes dinâmicas, muitas vezes altamente heterogêneas, com diferentes níveis de qualidade, esquemas, escalas de mensuração ou modalidades.

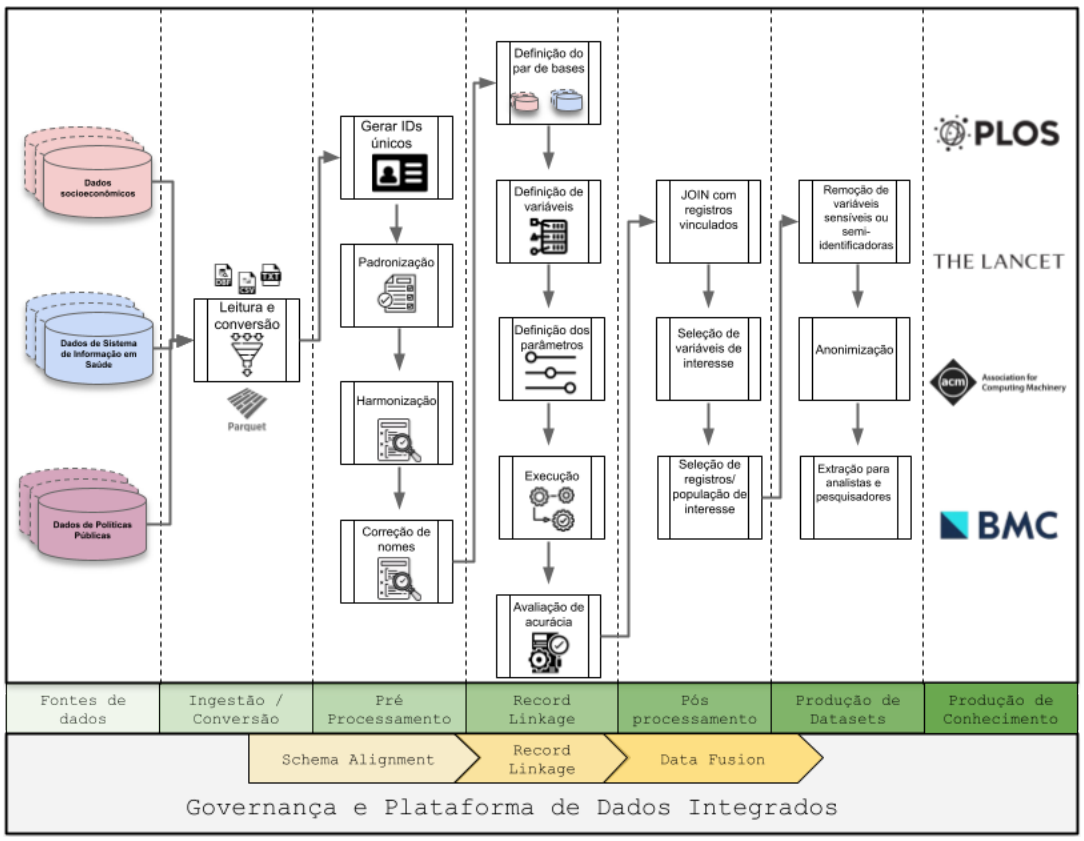

Figura 8.2. *Pipeline* Genérico de Integração de dados

No Brasil, diversos sistemas de informação suportam os processos de gestão do Sistema Único de Saúde (SUS). Estes sistemas diariamente coletam, armazenam e organizam dados capazes de subsidiar a medicina de precisão, o ajuste e a formulação de políticas públicas. Isoladamente, uma base de dados administrativa que registra eventos de hospitalizações pode oferecer *insights* capazes de subsidiar o processo de repasse de verbas federais, o provisionamento de insumos, o planejamento de estratégias para atenção primária, etc. Contudo, o enriquecimento destas informações com dados socio-

econômicos, notificação compulsória de doenças, ou morbimortalidade, permite a criação de um 'livro da vida', como proposto por [Dunn and Chief 1946], consolidando fatos relacionados a um mesmo indivíduo em várias fontes. Neste contexto, o uso de rotinas de RL tem chamado a atenção de pesquisadores, técnicos e gestores das áreas de gestão, governança, engenharia, análise, ciência e sistemas de dados [Harron et al. 2016].

A produção de conhecimentos em saúde através da ID dependem, portanto, da implementação de um *pipeline* maduro para gestão, manipulação, integração e uso destas bases massivas. A Figura 8.1 ilustra uma organização genérica deste *pipeline*, diluindo as três etapas propostas por [Dong and Srivastava 2013] em cinco fases de um modelo de Gestão e Governança numa Plataforma de Dados Integrados. Estas cinco fases são antecedidas de uma aquisição de dados e suportam a análise e descoberta de conhecimentos. É importante ressaltar que todo este processo de suporte a análises complexas, avaliação de políticas públicas e formulação de modelos para vigilância em saúde consiste num processo retroalimentado. Desta forma, as questões de pesquisa, hipóteses ou projetos elaborados impulsionam o pré-processamento, integração e extração destes dados da plataforma. Nas próximas subseções detalharemos essas fases, apresentando aspectos práticos de sua implementação através de pseudoalgoritmos comentados.

8.2.1. Técnicas de pré-processamento

Processos de vinculação de dados usam vários atributos que descrevem informações pessoais para classificar se um determinado par de registros se refere à mesma pessoa. As principais causas para baixos níveis de qualidade de dados estão relacionados com o fato de que bases de dados provenientes de diferentes sistemas de informação e mantenedores. Estes sitemas frequentemente definem diferentes padrões de formato, sistemas de dados e armazenamento físico. Do ponto de vista de rotina, os níveis de qualidade de dados são diretamente impactados pela supressão de valores, registros ou atributos para garantir segurança e privacidade [Christen et al. 2020].

As métricas de qualidade de dados são categorizadas entre aquelas capazes de medir a acurácia, validação, e completude [Christen 2012]. A acurácia é comumente avaliada de acordo com um documento auxiliar que mapeia a corretude dos dados coletados, tais como um dicionário de dados. A validade é aferida pela existência de valores plausíveis. Por exemplo, não é razoável haver indivíduos com idade maior que 200 anos. Para medir a completude, deve-se considerar a proporção de valores nulos ou ausentes.

Outras medidas de qualidade de dados incluem temporalidade, disponibilidade e credibilidade [Harron et al. 2016, Christen et al. 2020]. A temporalidade denota a distância entre os valores mensurados no tempo, suas versões mais atuais e a versão vinculada via ID. A disponibilidade indica a presença de atributos quase-identificadores com mesmo significado em ambas as bases. Por fim, para além do formato e da plausibilidade das métricas anteriores, a avaliação de credibilidade aferem se os registros armazenados numa coleção de dados são de origem confiável ou descrevem um fato.

Um dos principais desafios do BDI se refere a capacidade de conduzir um processo adequado de SA. produção de conhecimentos em saúde através da ID dependem, portanto, da implementação de um *pipeline* maduro para gestão, manipulação, integração e uso destas bases massivas. A Figura 8.1 ilustra uma organização genérica deste *pipe-*

Algorithm 1 Conversão Genérica de Formatos de Dados

Parâmetros:

- $dir\_entrada$: Diretório fonte (suporta múltiplos formatos - ver comentários)
- dir_saida: Diretório destino
- $formato\_saida \in \{\text{CSV}, \text{Parquet}, \text{ORC}\}$

procedure CONVERTERDADOS($dir\_entrada$, dir_saida, $formato\_saida$)
    $spark \leftarrow iniciar\_sessao()$ ▷ Suporta .dbf, .dbc, .csv, .txt, etc.
    $arquivo$ in $listar\_arquivos(dir\_entrada)$
    $df \leftarrow carregar\_qualquer\_formato(spark, arquivo)$
    $df \leftarrow tratar\_dados(df)$
    $escrever\_saida(df, dir\_saida, formato\_saida)$

    $spark.stop()$
end procedure

▷ Detalhes de implementação:
▷ - Suporta formatos: DBF, DBC, CSV, JSON, TXT e similares
▷ - Processamento otimizado para big data via Spark
▷ - Conversão preserva metadados e estrutura original

line, diluindo as três etapas propostas por [Dong and Srivastava 2013] em cinco fases de um modelo de Gestão e Governança de uma Plataforma de Dados Integrados. A seguir, faremos um detalhamentos das principais rotinas representadas na Figura 8.1, expondo os principais aspectos práticos que afetam a escalabilidade e acurácia de uma tarefa de integração de dados. Os conceitos e estratégias apresentadas a seguir estão de acordo com a experiência acumulada pelo Núcleo de Produção de Dados do Cidacs, que historicamente é liderado por pesquisadoras da estatística e formada tecnicamente por pessoas engenheiras de dados de diferentes níveis de senioridade. Esta equipe é, portanto, co-responsável, na Plataforma de Dados, pelas peças de software que implementam as rotinas nucleares da Figura 8.1.

As principais tarefas *upstream* que abordaremos aqui são as de maior impacto no contexto de BDI, principalmente para assegurar um bom resultado na etapa de BDL. Detalharemos as tarefas de i) Conversão, Geração de IDs únicos, ii) Padronização, e iii) Correção de nomes.

8.2.2. Conversão

Como ilustrado no Algorítmo 1, um primeiro esforço para garantir um bom resultado de integração consiste em estabelecer um formato inicial padronizado para todas as bases de origem (*raw*). Desta forma, arquivos com extensões suportadas por ferramentas analíticas tradicionais (csv, dat, dbf, json, sas, sql, txt, xls, xlsx ou xml) devem ser convertidos para uma única extensão/formato. Esta conversão objetiva, portanto, compatibilizar a leitura dos arquivos *raw*, permitindo a estruturação do *schema* em um formato fortemente tipado que suporte o processamento paralelo e distribuido, como o *parquet*.

Idealmente, este processo também deve incluir a substituição de caracteres que

possam interferir na quantidade de colunas e limitações de campos em um arquivo csv (contrabarra, vírgula e aspas duplas), remoção de acentuações e de caracteres binários não visíveis. Adicionalmente, converte-se a codificação dos caracteres do arquivo de UTF-8 (*8-bit Unicode Transformation Format* – padrão de codificação mais completo e complexo existente hoje, que inclui acentuações) para ASCII (*American Standard Code for Information Interchange*).

Algorithm 2 Geração de IDs Únicos para Bases de Dados

Parâmetros:

- $nome\_projeto$: Nome do projeto (ex: 'base_sim_2019_2020')
- $caminho\_entrada$: Caminho para os dados de entrada
- $caminho\_saida$: Caminho para os dados processados

procedure GERARIDSUNICOS
  Configurar ambiente: $spark.conf.set(\text{"}spark.sql.shuffle.partitions\text{"}, 288)$
  Listar arquivos de dados: $arquivos \leftarrow sorted(listar\_arquivos(caminho\_entrada))$
  Carregar dados:
  for all $arquivo$ in $arquivos$ **do**
    $dataframes \leftarrow spark.read.csv(arquivo, header = True, multiLine = True)$
  end for
  Função $adicionar\_id\_sequencial(dataframes, nome\_id, inicio)$:
  $dataframes\_com\_id \leftarrow []$
  $acumulador \leftarrow 0$
  $s \leftarrow inicio$
  for all df in $dataframes$ **do**
    $inicio \leftarrow acumulador + s$
    $acumulador \leftarrow acumulador + df.count()$
    $rdd\_com\_indice \leftarrow df.rdd.zipWithIndex()$
    Adicionar ID único:
    $dataframes\_com\_id.append(spark.createDataFrame(\ldots))$
  end for
  return $dataframes\_com\_id$
  Aplicar função: $dfs\_com\_id \leftarrow adicionar\_id\_sequencial(dataframes, \text{"}id\_unico\text{"})$
  Validação:
  for all df in dfs_com_id **do**
    $df.select(\text{"}id\_unico\text{"}).describe().show()$
  end for
  Organização dos resultados:
  for all i in $range(len(dfs\_com\_id))$ **do**
    $dfs\_com\_id[i].repartition(144).write.csv(\ldots)$
  end for
end procedure

Para garantia da integridade, a conversão deve ser sucedida de uma inspeção onde se compara a estrutura, o número de colunas, o nome das colunas e a quantidade de registros entre a base convertida e a *raw*.

8.2.3. Criação de IDs únicos

Em seguida, é importante atribuir um código identificador sequencial único controlado pela equipe de engenharia de dados envolvida na integração. Esta estratégia objetiva cumprir com requisitos de pseudonimização, onde o compartilhamento de identificadores internos não podem ser relacionados a registros idenficados fora do ambiente seguro. A inspeção do resultado do Algoritmo 2 consiste na verificação se o número de IDs únicos X coincide com a quantidade de registros N e se os valores vão entre um número inicial e $X + N$.

8.2.4. Padronização baseada em dicionário

A padronização explicitada no Algoritmo 3 tem papel relevante no SA. Além das bases de dados, deve-se considerar o dicionário de dados para compatibilizar os tipos de variáveis, sintaxe dos valores e nomes de atributos. Para fins de simplificação, consideraremos variáveis numéricas, categóricas, textuais (nomes) e de data. Considerando os tipos de dados suportados pelo Apache Spark, pode-se utilizar *LongType* para variáveis numéricas com mais de 13 dígitos e IntegerType para valores inteiros com menos de 13 dígitos que não são categóricas. A existência de caracteres não numéricos poderá indicar a nulidade daquele valor em determinados registros.

Para variáveis categóricas, os valores com correspondentes no dicionário deverão ser mantidos e os demais transformados. Registros nulos são mapeados para 0 (zero) e os inconsistentes (os que não estão contidos no dicionário) são mapeados para 99.

As variáveis de datas devem cumprir com algum formato suportado pelas ferramentas de processamento e análise e, para tanto, pode-se considerar a conversão destes valores para um padrão AAAA-MM-DD. Antes de padronizar cada variável de data, é utilizado uma Expressão Regular (regex) para substituir eventuais nomes de meses escritos ou abreviados em Inglês ou em Português, como mostrado na Tabela 8.2, para o valor numérico correspondente do mês.

Caso o campo de data contenha apenas 7 dígitos, a partir do formato da data informado no dicionário, faz-se 3 tentativas de adicionar um 0 à esquerda de cada um dos elementos da data (dia, mês, ano) até que uma data válida de 8 dígitos seja gerada correspondente com o formato no dicionário. Caso um valor não se enquadre como uma data ou contenha menos de 7 dígitos, ele é considerado inválido e seu campo será entendido como nulo (vazio). Para os casos de campo de data com 6 dígitos, deve-se fazer uma análise prévia e verificar se o ano não está abreviado em somente dois dígitos. Caso esteja, a depender do contexto da variável, o analista pode preencher este ano com '20' ou com '19', transformando o campo para 8 dígitos e, desta forma, a Padronização não irá anular estas datas.

8.2.5. Limpeza de nomes

Segundo [Christen et al. 2020], diferente de palavras comuns, nomes podem ser escritos e pronunciados de várias formas. Adicionalmente, os erros de grafia são os mais comuns neste tipo de variáveis, sendo que 80% acontecem apenas em um caracter [Damerau 1964]. Para que a base de dados seja submetida ao processo de Vinculação, é essencial que as variáveis utilizadas (nome, nome da mãe, data de nascimento, código do município de

Algorithm 3 Padronização de Dados com Base em Dicionário

Entrada:

- *dicionario*: Arquivo CSV com especificação dos campos (nome, tipo, transformação)
- *bases_dados*: Lista de DataFrames com os dados brutos
- *caminho_saida*: Diretório para armazenar dados padronizados

procedure PADRONIZARDADOS
  Definir funções de padronização:
  function PADRONIZARDATA(*valor*)
    if *valor* é nulo **then return** 0
    end if
    Tentar converter para formato AAAA-MM-DD
    if conversão falhar **then return** 0
    elsereturn data convertida
    end if
  end function
  function PADRONIZARNUMERO(*valor*)
    if *valor* é nulo **then return** 0
    else if *valor* é numérico **then return** inteiro(*valor*)
    elsereturn 0
    end if
  end function
  function PADRONIZARCATEGORIA(*valor*, *mapeamento*)
    if *valor* é nulo **then return** 0
    else if *valor* existe em *mapeamento* **then return** código correspondente
    elsereturn 99 ▷ Valor desconhecido
    end if
  end function
  Aplicar padronização:
  for all df in *bases_dados* **do**
    for all *coluna* in $df.colunas$ **do**
      Obter função de padronização correspondente
      $df[coluna] \leftarrow aplicar\_funcao(df[coluna])$
    end for
    Adicionar coluna *ano_base*
  end for
  Escrever resultados:
  for all df in *bases_dados* **do**
    $df.escrever\_parquet(caminho\_saida)$
  end for
end procedure

residência e sexo) estejam devidamente tratadas. Considerando que a etapa de Padronização já garante a remoção de inconsistências das variáveis de data de nascimento, sexo e município, faz-se necessário checar e remover inconsistências das variáveis de nome e nome da mãe.

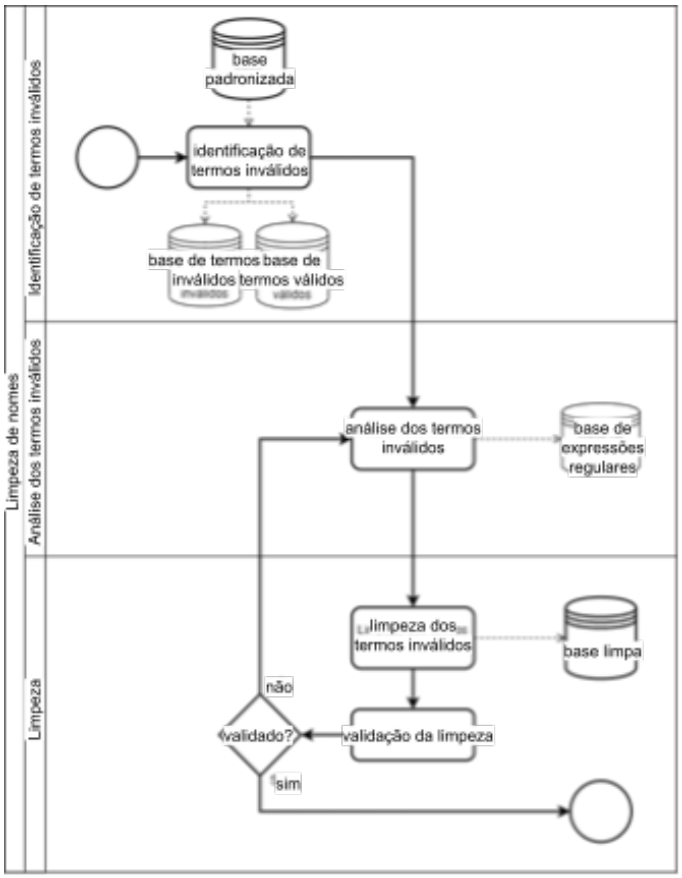

Figura 8.3. Fluxo de limpeza de nomes

Para as duas variáveis restantes o tratamento é mais complexo, pois se trata de variáveis do tipo String que possuem inconsistências diversas, incluindo o mau preenchimento por parte das pessoas que alimentam os sistemas de informação da base. A Figura 8.3 apresenta o fluxo da Limpeza de Nomes.

O processo de limpeza de nomes é iniciado com uma identificação dos tipos de inconsistência que ocorrem na base, das quais destacam-se três tipos de erros: i) gerados por ortografia ou uso de caracteres inválidos, ii) gerados por falta de informação, e iii) gerados por problemas na decriptação/dificação. No caso de erros ocasionados por problemas na digitação, um caractere pode ser trocado por outro, ou haver uma sequência de caracteres que não apresentam significado dentro do contexto da variável.

Muitas vezes, quando o indivíduo não possui documento de identificação no ato do cadastro no sistema, o campo nome ou nome da mãe são preenchidos com descrições da situação. Como, por exemplo, “IGNORADO”, “FILHO DE MARIA DA SILVA”; alguma caracterização do indivíduo como “DESCONHECIDA TRAJANDO SAIA ROSA” ou até algum apelido como “JOÃO DA BARBEARIA”.

Em alguns casos, a base, antes de ser cedida ao CIDACS, precisa passar por processos de criptografia e, quando é decriptada, revela alguns valores “corrompidos". Esses erros são bem específicos e com poucos padrões de repetição, o que torna difícil recuperar a informação original. Alguns exemplos de erros desse tipo são: "$EB", "&EBTJM", "TJM", "RHKUX", etc.

Uma vez mapeados os erros, é preciso definir o procedimento de tratamento para esse tipo de campo. Alguns procedimentos que devem ser avaliados e tratados são:

- Remoção de caracteres numéricos e especiais;
- Remoção de nome com somente um ou dois caracteres ou nomes com somente uma letra repetida *n* vezes, como, por exemplo: XXXX XXXXX;
- Identificação e remoção de inconsistências através de palavras-chave, por exemplo: IGNORADA, DESCONHECIDO, RECÉM-NASCIDO, RN DE, GEMELAR, LACTANTE, INDIGENTE;
- Identificação e análise de nomes que identifiquem a mãe do indivíduo como “RN DE <NOME_DA_MÃE>” ou “FILHO DE <NOME_DA_MÃE>”;
- Remoção de valores corrompidos por problemas na decriptação/codificação.

Em todos os casos, retira-se os termos que representam inconsistências e que eventualmente estejam no lugar do nome ou em meio aos nomes. Existem casos em que o nome do indivíduo está preenchido com algo que faz referência ao nome da mãe e o campo do nome da mãe está nulo. Neste caso, é possível recuperar o nome da mãe através da informação que está no nome do indivíduo,

8.2.6. Códigos fonéticos

Algoritmos fonéticos, incluindo Soundex e Metaphone, são frequentemente empregados no relacionamento de registros para identificar correspondências mesmo diante de divergências ortográficas ou erros de digitação. Essas técnicas exercem influência em quatro aspectos principais: proteção de dados, acurácia na vinculação, desempenho computacional e potenciais distorções. Embora contribuam para fortalecer a privacidade e aumentar a resiliência do processo de matching, podem acarretar certas perdas de exatidão e introduzir tendências sistemáticas [Karakasidis and Koloniari 2017, Herzog et al. 2007].

A transformação de nomes em códigos fonéticos irreversíveis reduz significativamente o risco de exposição de identificadores pessoais durante processos de record linkage, fortalecendo a proteção de dados sensíveis. Essa abordagem é particularmente valiosa em contextos que demandam privacidade por design, como na vinculação de registros de saúde ou dados governamentais. Contudo, para maior robustez contra tentativas de re-identificação, recomenda-se a combinação com técnicas complementares, tais como Filtros de Bloom ou injeção de ruídos [Schnell 2015].

Algorithm 4 Correção de Nomes com Regex e Validação via Base Auxiliar

Entrada:

- *bases_dados*: Lista de DataFrames Spark com dados de saúde
- *padroes_invalidos*: Lista de expressões regulares (`regex`) para nomes claramente inválidos
- *padroes_dubios*: Lista de expressões regulares (`regex`) para nomes de validade duvidosa
- *nomes_validos*: DataFrame Spark com nomes validados manualmente em validações históricas
- *caminho_saida*: Diretório para armazenar dados corrigidos

procedure CORRIGIRNOMES
    Carregar bases usando Spark
    Unir todas as bases em um único DataFrame $df\_unificado$
    Etapa 1: Pré-limpeza dos Nomes
    $df\_unificado \leftarrow$ aplicar `regexp_replace` para:

- Remover acentuação e caracteres especiais
- Eliminar múltiplos espaços
- Converter para letras maiúsculas

    Etapa 2: Marcação de Nomes Inválidos
    for all *regex* in *padroes_invalidos* **do**
        Marcar registros cujo *nome* corresponde a *regex* usando `rlike`
    end for
    Etapa 3: Identificação de Nomes Duvidosos
    for all *regex* in *padroes_dubios* **do**
        Marcar registros cujo *nome* corresponde a *regex* usando `rlike`
    end for
    Etapa 4: Validação dos Nomes Duvidosos
    $df\_dubios \leftarrow$ registros marcados como duvidosos
    $df\_validados \leftarrow$ resultado da junção de df_dubios com *nomes_validos* usando Spark `join`
    Adicionar novos nomes validados manualmente ao DataFrame *nomes_validos*
    Etapa 5: Aplicação das Correções
    for all *registro* in $df\_unificado$ **do**
        if *registro.nome* é inválido (marcado na etapa 2) **then**
            Corrigir para `NULL` ou valor padrão
        else if *registro.nome* está em $df\_validados$ **then**
            Manter nome original
        else if *registro.nome* é duvidoso e não validado **then**
            Marcar como "nome pendente para revisão"
        else
            Manter nome pré-limpo
        end if
    end for
    Escrever resultados no formato Parquet em *caminho_saida*
end procedure

Tabela 8.3. Codificação fonética de nomes com diferentes algoritmos

Nome	Soundex [Herzog et al. 2007]	Metaphone ptBR [Jordão and Rosa 2012]	NYSIIS [Grannis et al. 2002]
MARIA	M600	MARIA	MARY
MARIANA	M650	MARIANA	MARYAN
MARIANNA	M655	MARIANA	MARYAN
MARIA ANA	M655	MARIANA	MARYAN
MARINA	M650	MARINA	MARYN
MIRIAN	M650	MIRIAN	MARYAN
MILLIAM	M450	MILIAM	MALLYAN
MARIAH	M600	MARIA	MARY

8.3. Iniciativas de Linkage no Brasil

O Brasil tem avançado significativamente no uso de dados administrativos para pesquisa em saúde, impulsionado pelo desenvolvimento de infraestruturas especializadas e metodologias de linkage [Sanni Ali et al. 2019]. Iniciativas como o Centro de Integração de Dados e Conhecimentos para Saúde (CIDACS), a Universidade Federal de Minas Gerais (UFMG) e a Universidade do Estado do Rio de Janeiro (UERJ) têm promovido a integração de bases como o Sistema de Informações sobre Mortalidade (SIM) e o Cadastro Único para Programas Sociais (CadÚnico). Esses esforços possibilitam a realização de estudos epidemiológicos e avaliações de políticas públicas baseadas em evidências.

O CIDACS [Barreto et al. 2019], criado em 2016 e vinculado à Fiocruz, abriga a Coorte dos 100 Milhões de Brasileiros [Barreto et al. 2022] e a Coorte de Nascimentos [Paixao et al. 2021], dois dos maiores bancos de dados do país voltados à pesquisa populacional. Além de consolidar informações socioeconômicas e de saúde, o centro desenvolveu os algoritmos AtyImo e CIDACS-RL, que aprimoram a vinculação de registros administrativos, garantindo alta precisão na identificação de indivíduos em diferentes bases. O centro opera com um parque computacional robusto, que inclui procedimentos rigorosos de governança, anonimização de dados e ferramentas de linkage, que permite a análise e processamento de grandes volumes de dados.

Outros centros de pesquisa também contribuíram significativamente para a inovação no linkage de dados no Brasil. A UFMG tem avançado no campo de linkage de dados por meio de algoritmos como o FERAPARDA e o PAREIA [Sanni Ali et al. 2019]. Essas ferramentas utilizam técnicas de linkage probabilístico e computação de alto desempenho para lidar com grandes volumes de dados, permitindo a integração de bases como SIH, SIA, SIM, SINASC e SINAN. Estes esforços resultaram num banco de dados nacional que integra registros individuais do SUS ao longo de 15 anos, permitindo análises aprofundadas em saúde pública. A UERJ, por sua vez, desenvolveu o RecLink, uma das ferramentas de linkage probabilístico mais utilizadas no Brasil. Baseado no modelo de Fellegi-Sunter, o RecLink [Camargo Jr and Coeli 2000] tem sido amplamente empregado em estudos epidemiológicos e na avaliação de subnotificações de doenças, como a tuberculose. Além disso, a UERJ tem trabalhado na criação de um data warehouse para integrar sistemas de informação relacionados ao câncer, como SIH-SUS, APAC-ONCO e

SIM, facilitando a análise de dados clínicos e administrativos. Essas iniciativas demonstram o avanço da ciência de dados aplicada à pesquisa em saúde e à gestão pública.

8.3.1. Métodos probabilísticos

O *Record Linkage* é uma ferramenta essencial para integrar dados administrativos ou clínicos e utiliza métodos, que podem ser divididos em dois tipos principais: determinísticos e probabilísticos. A escolha do método depende da qualidade dos dados e dos recursos disponíveis [Dusetzina 2014]. O método determinístico utiliza regras exatas para comparar registros. Os pares são analisados de acordo com identificadores comuns, e apenas os que coincidem completamente são considerados correspondências. Esse processo é razoavelmente simples, mas não lida bem com erros ou variações nos dados [Dusetzina 2014].

Os métodos probabilísticos, por sua vez, são indicados quando os dados possuem erros ou variações frequentes. Eles estimam a probabilidade de dois registros representarem a mesma entidade. Esses métodos permitem uma abordagem mais flexível e robusta para lidar com incertezas nos dados [Dusetzina 2014]. A integração de dados probabilística é utilizada quando não há a disponibilidade de atributos identifidores capazes de discriminar indivíduos de forma unívoca em duas bases de dados [Blake et al. 2022]. Eles funcionam calculando as probabilidades de correspondência baseadas nos dados observados.

De acordo com a formulação teórica proposta por [Fellegi and Sunter 1969], o método de RL probabilístico tem o objetivo de calcular a probabilidade de dois registros representarem a mesma entidade (*match*) ou não (*non-match*) através de uma comparação de seus atributos. Desta forma, considere uma tarefa de integração das bases expressas nas Tabelas 8.1 e 8.2, utilizando os atributos em comum (nome, mãe, dn e sexo). Inicialmente, deve-se estabelecer pesos $\boldsymbol{m}$ e $\boldsymbol{u}$ para cada atributo. Os pesos $\boldsymbol{m}$ correspondem à probabilidade de os valores concordarem, dado que se referem à mesma pessoa. De forma análoga, os pesos $\boldsymbol{u}$ expressam a probabilidade da concordância entre um par de valores ser mera coincidência, não se referindo à mesma entidade. Os passos subsequentes à definição dos parâmetros consistem em calcular os *scores* para cada par de registros e realizar a classificação dos pares utilizando limiares (*thresholds*). Desta forma, a Tabela considere este exemplo de pesos para as bases das Tabelas 8.1 e 8.2.

Tabela 8.4. Exemplo de Parâmetros *m* e *u* para linkage probabilístico sobre os dados nas Tabelas 8.1 e 8.2

Atributo	m	u
nome	0,95	0,10
mae	0,90	0,05
dn	0,85	0,01
sexo	0,80	0,15

A etapa dedicada ao cálculo de *score* em tarefas de RL probabilístico pode utilizar a equação de **log-odds ratio**

$$Peso = \log_2 \left(\frac{m}{u} \right) \textbf{ (se concordam)} \quad (1)$$

$$Peso = \log_2 \left(\frac{1-m}{1-u} \right) \textbf{ (se discordam)} \quad (2)$$

. Então, a Tabela 8.5 apresenta um exemplo de aplicação das Equações 1 e 2 entre os registros 2 e 4 das Tabelas 8.1 e 8.2, condiderando que as etapas de *Schem Alignment* e pré-processamento já foram realizadas. O resultado deste caso, é expresso por $3.25 + 4.17 - 2.72 + 2.42 = 7.12$. Por fim, o método determina que todos estes pares de registros e seus scores sejam submetidos a uma etapa de classificação, onde subsidiarão a decisão final do RL.

Tabela 8.5. Exemplo de cálculo de pesos para linkage probabilístico

Campo	Base 1	Base 2	Concorda?	Peso (cálculo)
nome	VIVIAN GOMES	VIVIAN GOMES	Sim	$\log_2 \left(\frac{0,95}{0,10} \right) = 3,25$
mae	ARLENE LABORDA	ARLENE LABORDA	Sim	$\log_2 \left(\frac{0,90}{0,05} \right) = 4,17$
dn	2006-12-18	2006-12-08	Não	$\log_2 \left(\frac{0,15}{0,99} \right) = -2,72$
sexo	2	2	Sim	$\log_2 \left(\frac{0,80}{0,15} \right) = 2,42$

A fase final do RL probabilístico envolve a definição dos limiares superior $(T+)$ e inferior $(T-)$. Desta forma, aqueles pares que produziram *scores* acima de $(T+)$ são imediatamente classificados com *matches*. Abaixo de $(T-)$ estão os pares de registros classificados como não correspondentes (*non-matctches*). O intervalo entre $(T+)$ e $(T-)$, portanto, configuram uma "zona cinzenta"que demandará um processo de revisão. A parametrizaçao adequada deste procedimento está associado a limiares que reduzam a zona cinzenta e consiga otimizar uma métrica de qualidade de linkage (especificidade, sensibilidade, precisão, etc).

Escolher variáveis adequadas para o linkage é fundamental. Elas devem ser discriminantes e possuir boa qualidade. Isso ajuda a derivar probabilidades mais precisas e confiáveis [Blake et al. 2022]. Os métodos probabilísticos se diferenciam dos determinísticos, pois consideram que alguns identificadores têm maior poder discriminativo. Por exemplo, uma coincidência em sobrenome e data de nascimento pode ser mais significativa do que uma coincidência no sexo [Dusetzina 2014].

As principais limitações do RL probabilístico circulam em três principais categorias: desempenho computacional, uso de recursos e a necessidade de informações de referência para definição de pesos e limiares. Considerando a complexidade do pareamento, expressa por $|A| \times |B|$, onde $|A|$ e $|B|$ se referem a quantidade de registros nas bases envolvidas, executar o RL em grandes volumes de dados é proibitivo. Para contornar isso, utiliza-se a indexação/blocagem. Estes métodos implementam estruturas de dados robustas capazes de suportar a organização e recuperação eficiente de subconjuntos de registros. Este filtro tem o potencial de diminuir drasticamente as comparações desnecessárias e diminuir o tempo total de execução do método de RL. Contudo, uma definição adequada do algorítmo de indexação e seus parâmetros vai garantir que pares legítimos sejam retirados do escopo de busca [Dusetzina 2014, Christen 2011].

Outra limitação latente associada a este tipo de RL é a ausência de informações que suportem uma parametrização adequada para cada bar de bases de dados. Por exemplo, sem um conjunto de documentos que suporte as definições dos pesos $\boldsymbol{m}$ e $\boldsymbol{u}$ na integração dos dados que estão nas Tabelas 8.1 e 8.2, o procedimento tende a falhar no *link* de muitos registros. Uma abordagem eficiente é usar as frequências empíricas para estimar as probabilidades de correspondência. Essa técnica utiliza a proporção real de coincidências dentro dos próprios dados. Ela simplifica o processo e torna o método mais adaptado à realidade dos conjuntos de dados [Blake et al. 2022]. A eficácia dos métodos probabilísticos depende da suposição de independência entre os identificadores. Caso essa suposição seja violada, as estimativas podem perder validade. No entanto, esses métodos são flexíveis o suficiente para lidar com configurações de dados complexas [Blake et al. 2022]. A seguir, abordaremos de forma resumida algumas iniciativas brasileiras que se dedicam a endereçar alguns destes desafios.

8.3.2. Reclink

RecLink [Camargo Jr and Coeli 2000] é um sistema de relacionamento de bases de dados que implementa o método probabilístico de record linkage, desenvolvido em C++ utilizando o ambiente Borland C++ Builder 3.0. Desde sua criação, o RecLink tem passado por atualizações relevantes [Camargo Júnior and Coeli 2002, Camargo Junior and Coeli 2006] e hoje possui uma versão opensource [Camargo Jr and Coeli 2015, Camargo and Coeli 2011] que potencializou seu uso por pesquisadores da saúde pública, estatística e computação. Dentre suas evoluções, destaca-se também a adoção de algorítmos para a estimação de pesos e limiares [Junger 2006]. Inúmeros trabalhos relataram o uso de bases de dados integradas com esta solução [Coeli et al. , Oliveira et al. 2016, Guillen et al. 2017, Vidal et al. 2006, Vieira et al. 2017], aprimorando a completude e qualidade das informações para pesquisa e vigilância epidemiológica.

O funcionamento do RecLink é similar ao descrito nas seções anteriores deste capítulo, seguindo as três etapas fundamentais do record linkage probabilístico: padronização, blocagem e pareamento. A padronização trata campos como nomes e datas de forma uniforme, removendo variações que poderiam prejudicar a comparação (por exemplo, convertendo letras minúsculas em maiúsculas e eliminando pontuação). A blocagem utiliza o algoritmo soundex aplicado aos nomes para agrupar registros com pronúncias semelhantes, limitando as comparações aos pares dentro de um mesmo bloco e otimizando o desempenho. O pareamento se baseia na construção de um escore ponderado para cada par de registros, calculado a partir das comparações dos campos selecionados (como nome, data de nascimento e sexo), utilizando pesos baseados na razão de verossimilhança entre as probabilidades de concordância em pares verdadeiros e falsos (segundo a metodologia de Fellegi Sunter). O programa permite que os pares sejam classificados como "verdadeiros", "falsos"ou "duvidosos", conforme seus escores em relação a limiares definidos.

Em suas versões mais recentes, o RecLink aprimorou sua usabilidade e oferece uma interface interativa onde o usuário pode configurar os campos a serem usados, os algoritmos de comparação, e os limiares de decisão. Desde sua primeira versão, o programa permite diversas rodadas de blocagem com diferentes chaves, aumentando a chance de encontrar pares verdadeiros mesmo diante de erros ou variações nos dados. Com essas

características, o RecLink se estabelece como uma ferramenta acessível e eficiente para deduplicação e integração de registros em contextos reais.

8.3.3. PAREIA/ FERAPARDA

Em seu trabalho [Santos et al. 2007, dos Santos Filho 2008] descreve o "Filtro de Entidades para Remoção Automática Paralela de Registros Duplicados e Agressivamente Distribuída"(FERAPARDA). Esta ferramenta de deduplicação paralela, posteriormente referida como PAREIA [Junior et al. 2018], foi desenvolvida com foco em alto desempenho e escalabilidade. Ela foi projetada para identificar e remover registros duplicados em grandes bases de dados de forma eficiente, explorando arquiteturas distribuídas. O PAREIA foi originalmente construído sobre a plataforma Anthill [Ferreira et al. 2005], que permite a execução de aplicações baseadas em filtros encadeados (modelo de *pipeline*), onde cada filtro executa uma etapa do processo e pode ser replicado para mitigar a complexidade computacional da deduplicação. Trabalhos mais recentes relatam o uso do PAREIA para a criação de grandes ativos de dados integrados, produzidos a partir da deduplicação de mais de 1 bilhão de regisros [Junior et al. 2018].

Desde sua primeira publicação, o PAREIA propõe uma arquitetura em pipeline composta por filtros especializados. O filtro Reader divide a base de dados entre os processos e gera as chaves de blocagem. Em seguida, o Blocking agrupa os registros semelhantes, enquanto o Merge elimina pares redundantes. O Scheduler organiza os pares para reduzir o tráfego de dados entre processos, e o Comparator — a etapa mais custosa — aplica as funções de similaridade, podendo ser replicado para ganho de desempenho. Por fim, o Classifier avalia os escores de similaridade e decide se os pares são réplicas, com base em limiares pré-definidos. A comunicação entre os filtros é gerenciada pela biblioteca PVM (Parallel Virtual Machine). Projetado para suportar escalabilidade horizontal, o sistema permite aumentar dinamicamente o número de instâncias dos filtros mais exigentes. Essa arquitetura torna o FERAPARDA uma solução robusta e escalável para deduplicação de dados em ambientes distribuídos e de grande volume.

Seu funcionamento é dividido em estágios, implementados como filtros: o Reader divide a base entre os processos e gera as chaves de blocagem; o Blocking agrupa registros semelhantes; o Merge remove pares redundantes; o Scheduler organiza os pares para minimizar a comunicação; o Comparator aplica funções de similaridade e representa o estágio mais custoso (por isso pode ser replicado); e o Classifier determina se os pares são réplicas com base em limiares. A comunicação entre os filtros é gerenciada pela biblioteca PVM (Parallel Virtual Machine), e o sistema foi concebido para permitir a escalabilidade horizontal, aumentando o número de instâncias dos filtros críticos conforme a demanda. Essa arquitetura torna o FERAPARDA uma proposta inovadora para deduplicação de dados em ambientes distribuídos e com alto volume de informação.

8.3.4. AtyImo: Privacy-Preserving Record Linkage (PPRL)

AtyImo [Pita et al. 2018] é uma metodologia de record linkage com preservação de privacidade (PPRL - na sigla em inglês) escalável usada para criar a primeira versão de uma coorte de 100 milhões de indivíduos, integrando o CadÚnico e vários dados relacionados à saúde. O nome da ferramenta tem origem na concatenação de duas palavras,

o "*aty*"significa união em Tupy e "*imo*"se refere a conhecimento na língua Iorubá. Um 'átimo' também pode se referir a um evento que acontece num tempo bastante rápido.

Os Filtros de Bloom (BF, do inglês *Bloom Filters*) [Bloom 1970], originalmente desenvolvidos para verificar se a existência de um elemento pertence a um grupo, consistem em um vetor com M Bytes, todos iniciados com 0. Utilizando uma ou mais funções hash, define-se quais das M posições serão alteradas para 1. Na literatura, diversos estudos comunicam os avanços da capacidade dos Filtros de Bloom para assegurar a privacidade de atributos quasi-identificadores (QIDs) em tarefas de integração de dados [Franke et al. 2021, Ranbaduge and Schnell 2020, Vaiwsri et al. 2018]. A adoção do BF no AtyImo tem o objetivo de endereçar os requisitos originais de privacidade no acesso e integração de dados administrativos em ambientes computacionais com implementação limitada de segurança física e lógica. Outra propriedade relevante dos BF para o PPRL é o suporte ao cálculo de similaridade entre o par de registros codificados utilizando medidas como o Sorensen Dice [Cha 2008] ou o Jaccard [Fletcher et al. 2018].

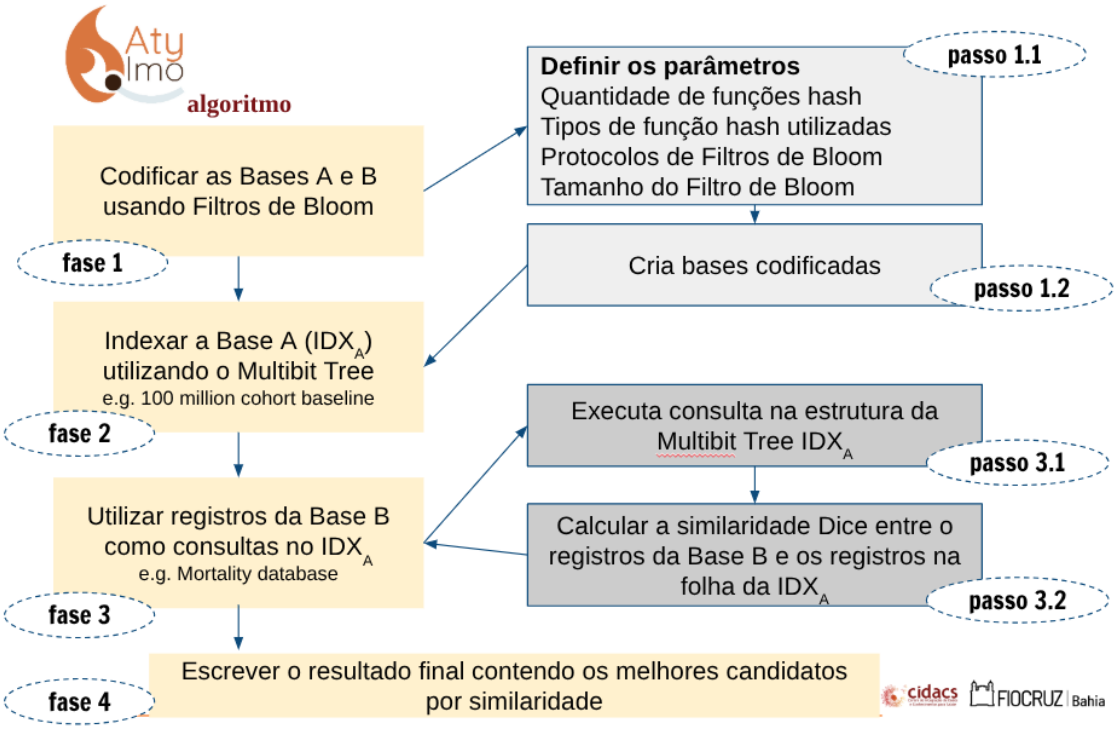

Figura 8.4. Fluxo de execução do AtyImo

8.3.4.1. Hashing e protocolos de Filtros de Bloom

Com o intuito de fortalecer a preservação de privacidade, é usual que o objeto usado na construção do BF passe por várias funções hash, alterando posições diferentes do vetor do BF para 1. Ao invés de utilizar várias funções hash diferentes, foi proposto [Dillinger and Manolios 2004] [Kirsch and Mitzenmacher 2006] o uso de duas funções hash (Double Hashing) dentro de uma fórmula, na qual uma das funções é multiplicada

por um fator que muda a cada iteração, simulando várias funções hash. Esse mesmo efeito pode ser alcançado através da Triple Hashing e Enhanced Double Hash.

```
import hashlib
# ============================
# Hashing Duplo (Double Hashing)
# ============================
def hash_duplo(token, repeticao, tamanho_filtro):
    """
    Calcula a posi  o no filtro com base em duas fun  es hash.
    Usa fator linear para dispers o das posi  es.
    """
    hash1 = hashlib.md5(token.encode()).hexdigest()
    hash2 = hashlib.sha256(token.encode()).hexdigest()
    posicao = (int(hash1, 16) + repeticao * int(hash2, 16)) %
   tamanho_filtro
    return posicao
# ============================
# Hashing Triplo (Triple Hashing)
# ============================
def hash_triplo(token, repeticao, tamanho_filtro):
    """
    Usa tres funcoes hash e termo quadratico para maior dispers o.
    """
    hash1 = hashlib.md5(token.encode()).hexdigest()
    hash2 = hashlib.sha256(token.encode()).hexdigest()
    hash3 = hashlib.sha512(token.encode()).hexdigest()
    termo_quadratico = int((repeticao * (repeticao - 1)) / 2)
    posicao = (
        int(hash1, 16)
        + repeticao * int(hash2, 16)
        + termo_quadratico * int(hash3, 16)
    ) % tamanho_filtro
    return posicao
# ====================================
# Hashing Duplo Aprimorado (Enhanced Double Hashing)
# ====================================
def hash_duplo_aprimorado(token, repeticao, tamanho_filtro):
    """
    Usa duas funcoes hash e termo cubico para aumentar a entropia.
    """
    hash1 = hashlib.md5(token.encode()).hexdigest()
    hash2 = hashlib.sha256(token.encode()).hexdigest()
    termo_cubico = int((repeticao**3 - repeticao) / 6)
    posicao = (
        int(hash1, 16)
        + repeticao * int(hash2, 16)
        + termo_cubico
    ) % tamanho_filtro
    return posicao
```

Para construir o BF em PPRL, geralmente os quase identificadores são divididos em q-gramas, que são strings de tamanho q. Por exemplo, usando q=3, é possível separar o nome João em dois q-gramas: "Joã"e "ão". Usando q-gramas, algumas técnicas foram desenvolvidas para preenchimento do BF. Em seu trabalho, Schnell et al. (2009)

sugeriu a codificação Attribute level Bloom filter (ABF), na qual cada atributo é utilizado para preencher um BF diferente. Como forma de tornar o BF mais resistentes a ataques, [Schnell et al. 2011] desenvolveu o Cryptographic Long-term Key (CLK), no qual todos os quase identificadores são inseridos no mesmo BF. Outra técnica é o record level Bloom filter (RBF), proposta por [Durham et al. 2013], na qual alguns bits do BF original são selecionados aleatoriamente para formar um novo BF.

Considere um registro $R = [$"JOAO PITA, "19870505, "1, "JANETE SANTOS$]$ com os seguintes valores para nome, data de nascimento, sexo e nome da mãe, respectivamente. Utilizando a estratégia de *Double Hashing*, deve cumprir com três etapas. Na primeira etapa, deve-se decompor o registro em n-gramas. Por exemplo, o valor "JOÃO PITA"deve ser inicialmente transformado em ['_J', 'JO', 'OA', 'AO', 'O ',' P', 'PI', 'IT', 'TA', 'A_']. Em seguida, aplica-se

$$\text{posição}_i = (\text{int}(h_1(\text{bigrama})) + i \cdot \text{int}(h_2(\text{bigrama}))) \mod m, \tag{3}$$

onde i é o índice da função hash simulada, variando de 1 até k (número total de funções hash), h_1 e h_2 são funções de hash distintas (por exemplo, MD5 e SHA256) e m é o tamanho do filtro de Bloom (número total de bits). Desta forma, se o valor produzido pela função $h_1(\texttt{"JO}) = 146379$ e $(h_2(\texttt{"JO})) = 94809$, com $m = 20$, temos para $i = 1$:

$$\begin{aligned}\text{posição}_1 &= (146379 + 1 \cdot 94809) \mod 20 \\ &= 241188 \mod 20 \\ &= 8,\end{aligned}$$

para $i = 2$:

$$\begin{aligned}\text{posição}_2 &= (146379 + 2 \cdot 94809) \mod 20 \\ &= 335997 \mod 20 \\ &= 17\end{aligned}$$

e para $i = 3$:

$$\begin{aligned}\text{posição}_3 &= (146379 + 3 \cdot 94809) \mod 20 \\ &= 430806 \mod 20 \\ &= 6.\end{aligned}$$

O vetor correspondente ao registro R iniciado com 20 zeros vai ter, portanto, as posições 8, 18, 10 alterados para 1. No final, codificar todo o registro R exigirá que estes passos sejam sucessivamente repetidos até que se esgotem os bi-gramas em R.

O *Triple Hashing* consiste na estratégia de construir BF de R utilizando três funções de hash distintas (h_1, h_2, h_3), e a posição i-ésima no vetor é obtida pela fórmula

$$\text{posição}_i = \left(\text{int}(h_1(\text{bigrama})) + i \cdot \text{int}(h_2(\text{bigrama})) + \frac{i \cdot (i-1)}{2} \cdot \text{int}(h_3(\text{bigrama}))\right) \mod m. \tag{4}$$

Portanto, considerando que $\text{int}(h_1(\texttt{"JO})) = 146379$, $\text{int}(h_2(\texttt{"JO})) = 94809$ e $\text{int}h_3(\texttt{"JO}))$ = 30721, então, para $i = 1$:

$$\begin{aligned}\text{posição}_1 &= \left(146379 + 1 \cdot 94809 + \frac{1 \cdot (1-1)}{2} \cdot 30721\right) \mod 20 \\ &= (146379 + 94809 + 0) \mod 20 \\ &= 241188 \mod 20 \\ &= 8\end{aligned}$$

para $i = 2$:

$$\begin{aligned}\text{posição}_2 &= \left(146379 + 2 \cdot 94809 + \frac{2 \cdot (2-1)}{2} \cdot 30721\right) \mod 20 \\ &= (146379 + 189618 + 30721) \mod 20 \\ &= 366718 \mod 20 \\ &= 18\end{aligned}$$

e para $i = 3$:

$$\begin{aligned}\text{posição}_3 &= \left(146379 + 3 \cdot 94809 + \frac{3 \cdot (3-1)}{2} \cdot 30721\right) \mod 20 \\ &= (146379 + 284427 + 2 \cdot 30721) \mod 20 \\ &= (146379 + 284427 + 61442) \mod 20 \\ &= 492248 \mod 20 \\ &= 8.\end{aligned}$$

Neste caso, enquanto o *Double Hashing* produzirá Bytes 1 nas posições 8, 18 e 10 do BF de R, o *Triple Hashing* vai incidir apenas nas posições 8 e 18.

A formulação do *Enhanced Double Hashing*, incluindo um termo cúbico dependente do índice i, envolve o seguinte calculo de posição:

$$\text{posição}_i = \left(\text{int}(h_1(\text{bigrama})) + i \cdot \text{int}(h_2(\text{bigrama})) + \frac{i^3 - i}{6}\right) \mod m. \qquad (5)$$

O termo cúbico expresso em $\frac{i^3-i}{6}$ tem o objetivo de i) reduzir colisões como as produzidas na demonstração do *Double Hashing*, ii) introduzir uma variação adicional que simula o uso de várias funções hash consecutivas, e iii) amplia o alcance das Equações 3 e 4 para atribuir valores 1 em posições mais distribuídas ao longo do BF. Então, considerando que $\text{int}(h_1(\texttt{"JO})) = 146379$ e $\text{int}(h_2(\texttt{"JO})) = 94809$, com $m = 20$, temos para $i = 1$:

$$\begin{aligned}\text{posição}_1 &= (146379 + 1 \cdot 94809 + \frac{1^3 - 1}{6}) \mod 20 \\ &= (146379 + 94809 + 0) \mod 20 \\ &= 241188 \mod 20 \\ &= 8,\end{aligned}$$

para $i = 2$:

$$\begin{aligned}\text{posição}_2 &= (146379 + 2 \cdot 94809 + \frac{8-2}{6}) \mod 20 \\ &= (146379 + 189618 + 1) \mod 20 = 335998 \mod 20 \\ &= 18,\end{aligned}$$

e para $i = 3$:

$$\begin{aligned}\text{posição}_3 &= (146379 + 3 \cdot 94809 + \frac{27-3}{6}) \mod 20 \\ &= (146379 + 284427 + 4) \mod 20 \\ &= 430810 \mod 20 \\ &= 10.\end{aligned}$$

A decisão sobre qual a melhor estratégia de composição de funções Hash para a decisão das posições que receberão o valor 1 em BFs deve ser tomada com base em avaliação experimental. Deve-se levar em consideração as bases de dados envolvidas, a quantidade média de q-gramas e a qualidade associada a estes registros.

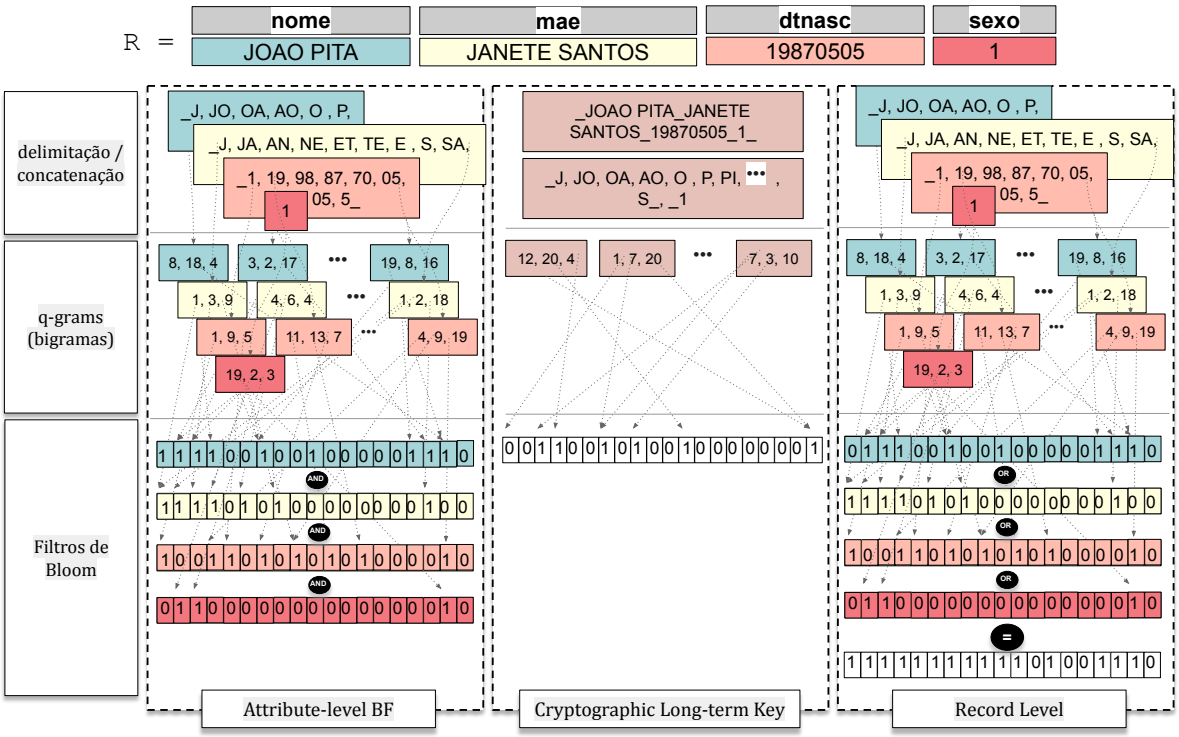

Figura 8.5. Exemplos de Protocolos de Bloom para o Registro R com Double Hashing com 20 posições

Outra decisão relevante no projeto de um PPRL consiste na escolha do Protocolo BF que será utilizado. As alternativas mais citadas na literatura são o *Attribute level Bloom filter* (ABF) [Schnell et al. 2009], *Cryptographic Long-term Key* (CLK) [Schnell et al. 2011] e o *record level Bloom filter* (RLB) [Durham et al. 2013]. Considere um cenário hipotético em que pretende codificar o registro *R* usando o *Triple Hashing* e em um vetor de 20

posições, como ilustrado na Figura 8.5. O protocolo ABF implementa três passos. No primeiro passo, cada atributo é decomposto em q-grams - em bigramas neste caso. Cada conjunto de bigramas de um respectivo atributo será submetido à Equação 4 para definir qual posição de um vetor independente receberá o valor 1. Portanto, o resultado do ABF é um vetor com em todos os bigramas produzidos será de $a \times m = 80$, sendo a e m a quantidade de atributos e o tamanho do BF, respectivamente.

```
# ABF UDF: Geracao do filtro de Bloom em nivel de atributo
def abf(registro, m, k, t_hash):
    """
    Cria um Attribute Level Bloom Filter (ABF) para cada atributo do registro.
    Para cada valor, gera bigramas e aplica a funcao de hash definida.

    Parametros:
    - registro: lista de valores de atributos
    - m: tamanho do filtro de Bloom (numero de bits)
    - k: numero de iteracoes (funcoes hash simuladas)
    - t_hash: tipo de funcao de hash ("DoubleHash", "TripleHash" ou outro)

    Retorna:
    - Filtro de Bloom concatenado para todos os atributos.
    """
    filtro_bloom_final = []
    for item in registro:
        filtro = [0] * m
        if item is not None:
            item = '_' + str(item) + '_'
            for j in range(len(item) - 1):
                cont = 1
                while cont <= k:
                    substring = item[j:j+2]
                    if t_hash == "DoubleHash":
                        pos = dbhash(substring, cont, m)
                    elif t_hash == "TripleHash":
                        pos = tphash(substring, cont, m)
                    else:
                        pos = endbhash(substring, cont, m)
                    filtro[pos] = 1
                    cont += 1
        # Concatena o filtro do atributo ao vetor final
        filtro_bloom_final.extend(filtro)
    return filtro_bloom_final
```

O CLK, por sua vez, concatena todos os atributos como uma única *string* e produz os bigramas que serão submetidos à Equação 4, resultando em um BF de 20 posições.

```
# CLK UDF: Geracao do filtro de Bloom baseado no registro completo (Cryptographic Long-term Key)
def clk(registro, m, k, t_hash):
    """
    Cria um filtro de Bloom unico para um registro completo, aplicando a tecnica
    Cryptographic Long-term Key (CLK). Todos os atributos sao
```



```
    considerados juntos.

    Para cada valor do registro, sao gerados bigramas com delimitadores
     e, para cada bigrama,
    aplicam-se funcoes de hash simuladas para definir posicoes no
    filtro de Bloom.

    Parametros:
    - registro: lista de valores de atributos do registro
    - m: tamanho do filtro de Bloom (numero total de bits)
    - k: numero de funcoes hash simuladas
    - t_hash: tipo da funcao de hash a ser utilizada ("DoubleHash", "
    TripleHash" ou outro)

    Retorna:
    - Um vetor binario representando o filtro de Bloom do registro
    inteiro.
    """
    # Inicializa o vetor do filtro de Bloom com zeros
    filtro_bloom = [0] * m

    # Itera sobre cada atributo do registro
    for item in registro:
        if item is not None:
            # Converte o valor para string e adiciona delimitadores
            item = '_' + str(item) + '_'

            # Gera bigramas e aplica as funcoes de hash simuladas
            for j in range(len(item) - 1):
                bigrama = item[j:j+2]
                for cont in range(1, k + 1):
                    # Escolhe a funcao de hash conforme o tipo indicado
                    if t_hash == "DoubleHash":
                        pos = dbhash(bigrama, cont, m)
                    elif t_hash == "TripleHash":
                        pos = tphash(bigrama, cont, m)
                    else:
                        pos = endbhash(bigrama, cont, m)

                    # Define a posicao correspondente como 1
                    filtro_bloom[pos] = 1

    return filtro_bloom
```

Similarmente ao ABF, o RLB gera um BF independente para cada produto. Contudo, o RLB computa funções "OR"entre os bytes de mesma posição para definir quais posições do BF resultante receberão o valor 1. A decisão sobre a combinação mais adequada entre as estratégias de hashing e o protocolo de BF depende de várias características do projeto de integração de dados, a exemplo da capacidade de armazenamento e processamento, impactados pelo tamanho dos vetores e dos requisitos de privacidade.

```
# RLB UDF: Geracao do filtro de Bloom em nivel de registro (Record-
    level Bloom Filter)
def rlb(registro, a, m, k, t_hash, positions, vetor):
```



```
    """
    Cria um Record-Level Bloom Filter (RLB) a partir de um ABF e um
    conjunto
    de posicoes sorteadas, aplicando embaralhamento controlado.

    Parametros:
    - registro: lista de valores de atributos
    - a: tamanho final do filtro de Bloom (numero de bits no RLB)
    - m: tamanho do filtro de Bloom por atributo (usado no ABF)
    - k: numero de iteracoes (funcoes hash simuladas)
    - t_hash: tipo de funcao de hash ("DoubleHash", "TripleHash" ou
    outro)
    - positions: vetor com posicoes sorteadas para cada bloco do ABF
    - vetor: vetor de indices usados para embaralhar o resultado final

    Retorna:
    - Um vetor binario com 'a' bits representando o RLB.
    """
    filtro_intermediario = []
    filtro_final = [0] * a
    inicio = 0
    deslocamento = 0

    # Gera o ABF completo concatenado para todos os atributos
    abf_completo = abf(registro, m, k, t_hash)

    # Para cada bloco de tamanho m, seleciona posicoes sorteadas
    while inicio < a:
        for i in range(inicio, m + inicio):
            posicao = positions[i] + deslocamento
            filtro_intermediario.append(abf_completo[posicao])
        inicio += m
        deslocamento += m

    # Embaralha o vetor intermediario conforme o vetor de indices
    for i in range(len(filtro_intermediario)):
        filtro_final[i] = filtro_intermediario[vetor[i]]

    return filtro_final
```

Por fim, pode-se utilizar UDFs em rotinas Apache Spark para garantir a adaptação destas tarefas em uma infraestrutura distribuída e escalável.

```
# ABF - Aplicacao da funcao ABF como UDF no PySpark
# Referencia: https://sparkbyexamples.com/pyspark/pyspark-udf-user-
    defined-function/
abfUDF = udf(lambda z: abf(z, m, k, t_hash), ArrayType(IntegerType()))
df_a_ABF = df_a.withColumn("bloom", abfUDF(df_a.vet))
df_b_ABF = df_b.withColumn("bloom", abfUDF(df_b.vet))

# CLK - Aplicacao da funcao CLK como UDF no PySpark
clkUDF = udf(lambda z: clk(z, a, k, t_hash), ArrayType(IntegerType()))
df_a_CLK = df_a.withColumn("bloom", clkUDF(df_a.vet))
df_b_CLK = df_b.withColumn("bloom", clkUDF(df_b.vet))

```



```
# RLB - Aplicacao da funcao RLB como UDF no PySpark
rlbUDF = udf(lambda z: rlb(z, a, m, k, t_hash, positions, vetor),
    ArrayType(IntegerType()))
df_a_RLB = df_a.withColumn("bloom", rlbUDF(df_a.vet))
df_b_RLB = df_b.withColumn("bloom", rlbUDF(df_b.vet))
```

8.4. CIDACS-RL

O CIDACS-RL é uma ferramenta de BDL que implementa uma abordagem baseada em pontuação de similaridade para pareamento de bases de dados em larga escala. Desenvolvido para operar em ambientes distribuídos com Apache Spark e Elasticsearch, o sistema foi projetado para lidar com desafios típicos de ID em saúde e registros administrativos, como variabilidade nos identificadores, erros de digitação e ausência de chaves únicas universais. O núcleo metodológico do CIDACS-RL baseia-se em um processo de duas fases: uma etapa inicial de busca exata (exact match) seguida por uma fase de correspondência aproximada (fuzzy match), otimizada por técnicas de indexação e bloqueio para reduzir o espaço de busca.

Na fase de busca exata, o sistema constrói consultas estruturadas no Elasticsearch utilizando campos pré-definidos como obrigatórios (must_match), como nome completo e data de nascimento. Essas consultas são geradas dinamicamente a partir de um arquivo de configuração que especifica os campos relevantes, pesos e limiares de similaridade. Quando um registro atinge um score de similaridade acima do limiar configurado (cutoff_exact_match), ele é considerado uma correspondência válida e removido do pool de registros a serem processados na fase seguinte. Essa abordagem garante eficiência computacional ao resolver casos inequívocos precocemente.

Para registros não pareados na primeira fase, como ilustrado na Figura 8.6, o CIDACS-RL aplica algoritmos de correspondência aproximada com tolerância a erros. A ferramenta utiliza medidas de similaridade como Jaro-Winkler para nomes, Hamming para datas e overlap para campos categóricos, com pesos ajustáveis para cada atributo. Um aspecto inovador é a integração nativa com recursos de fuzzy matching do Elasticsearch, como consultas com parâmetros de fuzziness e boosting, que permitem capturar variações ortográficas sem sacrificar desempenho. A combinação linear ponderada dessas medidas produz um escore final de similaridade, utilizado para selecionar a melhor correspondência entre os candidatos pré-filtrados.

O CIDACS-RL é uma ferramenta avançada que implementa estratégias de indexação baseadas em Elasticsearch, reduzindo drasticamente o custo computacional do pareamento de registros. Desenvolvido pelo CIDACS/Fiocruz, o CIDACS-RL tem sido utilizado em projetos nacionais, demonstrando alta eficiência e escalabilidade [Barbosa et al. 2020].

A versão atual, publicamente disponível para uso inclui: i) o provisionamento local de infraestrutura usando conteinerização através do Docker e Podman, ii) a conexão com o serviço de computação em nuvem de preferência do usuário, ou iii) a conexão com a infraestrutura on-premises do pesquisador.

valores_de_variaveis	consulta_elasticsearch	melhor candidato exato	similaridade	vetor_candid	linked_from
...	...	...	...	...	...
['JOAO PITA', '19870505', '1', 'JANETE SANTOS']	{ "size": "50", "query": { "bool": { "should": [{'match': {'nome_a': {'query': 'JOAO PITA', 'fuzziness':'AUTO', 'operator':'or', 'boost':'3.0'}}}, {match": {"birthdate":"19870505"}}] } } }, "term": {"sexo_a":"1"}}] } } }	13	0.94	{7: 0.98, 3: 0.8...	non_exact_match
['MARIA SENA, 19891112', '2', NAIRA DILMA']	{ "size": "50", "query": { "bool": { "should": [{'match': {'nome_a': {'query': 'JOAO PITA', 'fuzziness':'AUTO', 'operator':'or', 'boost':'3.0'}}}, {match": {"birthdate":"19870505"}}] } } }, "term": {"sexo_a":"1"}}] } } }	null	null	null	exact_match
...	...	...	...	...	...

Figura 8.6. Exemplo de Dataframe Spark operado pelo CIDACS-RL

8.4.1. Algorítmo do CIDACS-RL

O CIDACS-RL, descrito em detalhes na Figura 8.7, é baseado em quatro fases. A primeira fase consiste numa indexação da base de busca (geralmente a base com maior quantidade de registros) usando a biblioteca Elasticsearch-Hadoop [1]. Nesta fase, um Dataframe Spark pré-processado é transformado num Índice Elasticsearch que poderá ser alvo de consultas posteriormente. O Elasticsearch [Shukla 2015] é uma plataforma de indexação e busca distribuída que oferece uma API, linguagem de consulta própria e ferramentas nativas que suportam análises em bancos de dados de alta dimensionalidade. Apesar de implementar o modelo de dados orientado a documentos, é possível usar, em bases estruturadas advindas de bancos relacionais, representar abstração de tuplas em documentos através de dicionários Python ou arquivos JSON. Desta forma, é possível navegar ou consultar coleções indexadas pelo Elasticsearch usando estuturas de dados chave-valor.

```
# Funcao python para indexacao
def index_dataframe(dataframe, es_index_name):
    # creating new index
    dataframe.write.format("org.elasticsearch.spark.sql") \
                .option("es.resource", es_index_name).mode('overwrite'
   ).save()
```

Para garantir a escalabilidade, permitindo o paralelismo ou distribuição plena das suas instruções, a implementação do CIDACS-RL foca em funções nativas ou definidas por usuário (UDF, do inglês *User Defined Function*) que operem diretamente em Spark Dataframes. Isso implica em evitar funções e procedimentos classificados como "Ações", permitindo um acúmulo de "Transformações"para garantir uma otimização maior do plano de execução da rotina.

A segunda fase do algorítmo do CIDACS-RL tem o objetivo de endereçar a complexidade computacional imposta pelo alto volume de dados. Sua implementação consiste em construir dois tipos de consultas a partir dos valores nas colunas que serão utilizadas na comparação par-a-par. Por exemplo, para uma base com atributos "nome", "dtnasc",

[1] https://mvnrepository.com/artifact/org.elasticsearch/elasticsearch-hadoop

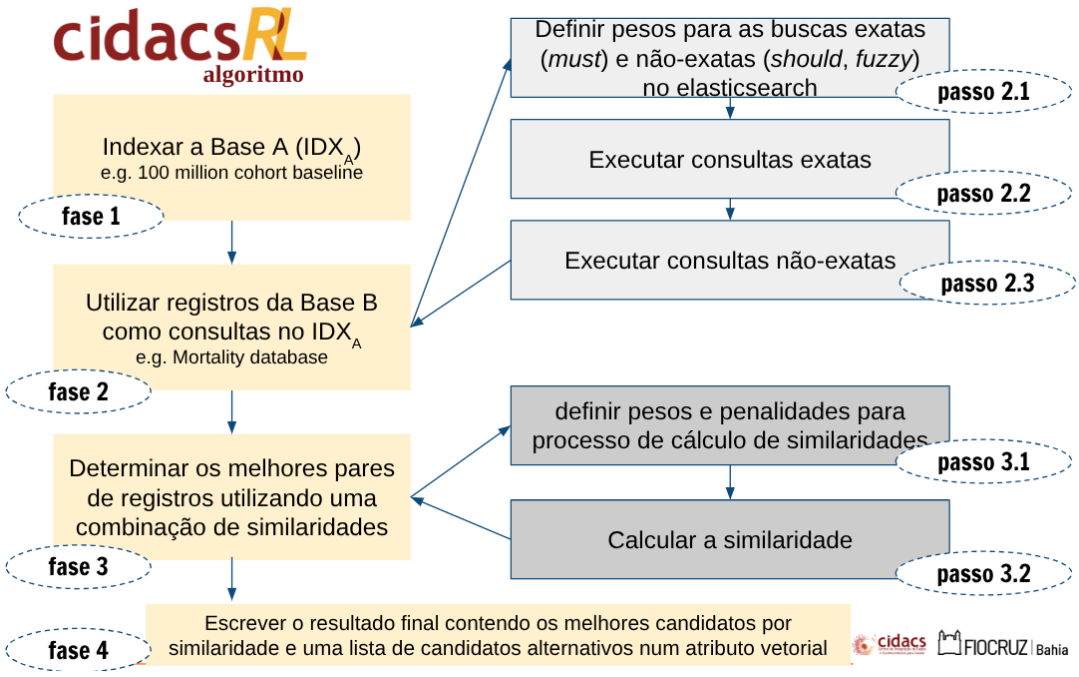

Figura 8.7. Fluxo de execução do CIDACS-RL

"sexo"e "nome_mae", como descrito na Figura 8.6. Os valores de uma tupla qualquer, neste Dataframe, pode ser, portanto, transformado em [‘JOAO PITA’, ‘19870505’, ‘1’, ‘JANETE SANTOS’] usando a Linha 7 do código a seguir.

```
1  # Trecho com implementacao da criacao de busca exata
2  def cidacs_rl_exact_phase(tolink_dataset):
3  # [...]
4      # selecting columns
5      tolink_dataset = tolink_dataset.select(tolink_columns)
6      # building array of variable values
7      tolink_dataset = tolink_dataset.withColumn('vars', F.array(
       tolink_columns))
8      # building exact queries
9      tolink_dataset = tolink_dataset.withColumn('exact_queries',
       udf_build_exact_queries(F.col('vars')))
10     # finding the best candidate for each tolink record
11     tolink_dataset = tolink_dataset.withColumn('result_exact_search', F
       .explode(F.array(udf_find_elasticsearch_exact_best_candidate(F.col(
       'vars'), F.col('exact_queries')))))
12 # [...]
13     # exploding array columns from the last function into 4 atomic cols
14     tolink_dataset = tolink_dataset.withColumn('best_candidate_exact',
       tolink_dataset.result_exact_search['best_candidate_exact'])
15     tolink_dataset = tolink_dataset.withColumn('
       sim_best_candidate_exact', tolink_dataset.result_exact_search['
       sim_best_candidate_exact'])
16     tolink_dataset = tolink_dataset.withColumn('
       similarity_exact_candidates', tolink_dataset.result_exact_search['
       similarity_exact_candidates'])
17
```



```
18      tolink_dataset = tolink_dataset.withColumn('
        sim_best_candidate_exact', F.col('sim_best_candidate_exact').cast('
        float'))
19
20      # dropping array columns
21      cols_to_drop = ['result_exact_search']
22      tolink_dataset = tolink_dataset.drop(*cols_to_drop)
23  # [...]
```

O resultado da UDF udf_build_exact_queries, expresso na Linha 9 do código acima seria algo parecido com o seguinte JSON, utilizado para efetivar a busca exata, exposta no Passo 2.2 da Figura 8.7.

```
1  # Exemplo de busca exata
2  { "size": "50",
3   "query": {
4    "bool": {
5     "must": [
6      {"match": {"nome":"JOAO PITA"}},
7      {"match": {"dtnasc":"19870505"}}] } } }
```

Perceba que, neste exemplo hipotético de busca exata apenas foram considerados o nome e a data de nascimento. Isso se dá porquê na busca exata, sugere-se utilizar um conjunto menor de variáveis que, conjuntamente, tem o potencial de identificar univocamente uma pessoa nas duas bases de dados envolvidas. Esta definição deve ser feita por um arquivo de configuração, onde deve-se definir também os pesos para valores correspondentes no Elasticsearch. Em seguida, produzidos os dicionários de busca exata para cada linha do Spark Dataframe, executa-se a UDF udf_find_elasticsearch_exact_best_candidate, exposta na Linha 11 do código acima. Esta UDF é responsável por enviar, de forma distribuída, as consultas a um cluster Elasticsearch. A latência associada a estas buscas dependem das configurações de *tunning* destes serviços e servidores. Uma vez finalizados, os *hits* de cada consulta são retornados na coluna result_exact_search(). Em seguida, estes candidatos são submetidos para um cálculo *ad hoc* de similaridade entre os candidatos e o registro que originou a consulta. São considerados pares "exatos"aqueles trazidos pelo Elasticserch que tenham obtido, pelo menos, 0$\dot{9}$5 de similaridade. Todos os registros que não obtiverem um candidato "exato"devem, portanto, ser submetidos a um outro passo da Etapa 2, que implementa uma busca mais flexível.

Similarmente ao passo 2.2, o passo 2.3 produz uma consulta utilizando um conjunto mais amplo de atributos utilizados na comparação par-a-par. Neste passo, deve-se considerar os parâmetros de *boosting* do Elasticseaerch, especialmente para atributos textuais nominais com diretiva de busca "*fuzzy*".

```
1  # Exemplo de busca exata
2  { "size": "50",  "query": {  bool ": {  should ": [
3
4  {'match': {'nome_a':
5      {'query': 'JOAO PITA', 'fuzziness':'AUTO', 'operator':'or', 'boost'
       :'3.0'}}},
6
7    {match": {"birthdate":"19870505"}} ] } } },
8      term ": {"sexo_a":"1"}} ] } } }
```


A consulta Elasticseach acima produzirá no máximo 50 candidatos para o registro da Base B utilizado no passo 2.2, flexibilizando erros na grafia do nome ou nos valores das demais variáveis envolvidas. Desta forma, as operações subsequentes são determinadas por uma UDF específica, considerando apenas os registros sem resultados no passo 2.2 (busca exata).

```
# Implementa  o do Passo 2.3 (busca nao - exata)
# [...]
    # building linked_from column. Non-null values on
    sim_best_candidate_exact must be filled
    # as 'exact_match', otherwise as 'non_exact_match'.
    filter_isnull = F.col('sim_best_candidate_exact').isNull()
    tolink_dataset = tolink_dataset.withColumn('linked_from', F.when(
    filter_isnull, 'non_exact_match').otherwise('exact_match'))

    # preparing filters for debug and non-debug executions
    filter_exact = F.col('linked_from') == 'exact_match'
    filter_non_exact = F.col('linked_from') == 'non_exact_match'

# [...]

    # creating, for remainder dataframe, the cols created in this
    function to ensure union
    tolink_dataset = tolink_dataset.withColumn('
    best_candidate_non_exact', F.lit(None))
    tolink_dataset = tolink_dataset.withColumn('
    sim_best_candidate_non_exact', F.lit(None))
    tolink_dataset = tolink_dataset.withColumn('
    similarity_non_exact_candidates', F.lit(None))
    tolink_dataset = tolink_dataset.withColumn('non_exact_queries', F.
    lit(None))
# [...]
```

Por fim, na função principal do CIDACS-RL, é estabelecida a ordem de execução do algorítimo, tal como ilustrado na Figura 8.7.

```
# Implementacao do Passo 2.3 (busca nao - exata)
# [...]
    tolink_dataset = cidacs_rl_exact_phase(tolink_dataset)

    tolink_dataset = cidacs_rl_non_exact_phase(tolink_dataset)

    tolink_dataset = tolink_dataset.withColumn('final_cidacs_rl_score',
     F.when(F.col('linked_from') == 'exact_match', F.col('
    sim_best_candidate_exact')).otherwise(F.col('
    sim_best_candidate_non_exact')))

    tolink_dataset = tolink_dataset.withColumn('final_cidacs_rl_id', F.
    when(F.col('linked_from') == 'exact_match', F.col('
    best_candidate_exact')).otherwise(F.col('best_candidate_non_exact')
    ))
# [...]
```

8.5. Iniciativas de Linkage emergentes

Nesta seção, classificaremos as iniciativas emergentes em quatro categorias: i) preservação de privacidade, ii) integração federada, iii) habilitada por modelos de linguagem ou inteligência artificial iv) alto desempenho e escalabilidade, e v) justiça algorítmica.

8.5.1. Métodos de PPRL emergentes

Avaliações experimentais recentes [Vidanage et al. 2023, Christen et al. 2025, Christen 2014] relatam que diversas estratégias de codificação, inclusive os Bloom Filters [Vidanage et al. 2019], nos diferentes modelos adversariais, podem ser explorados por ameaças amplamente discutidas na literatura. Uma iniciativa que visa superar a maior parte destes desafios contemporâneos foi apresentada por [Christen et al. 2024] que propõe um framework criptográfico que permite a comparação de similaridade entre pares de registros em ambientes não-confiáveis com o compartilhamento de apenas um parâmetro.

8.5.2. RL federado

A integração de dados federada é outra tendência emergente, impulsionada por projetos colaborativos que buscam combinar registros de saúde, educação e assistência social de diferentes países ou regiões. Plataformas como o OpenSAFELY, no Reino Unido, e o Population Data BC, no Canadá, demonstram como é possível realizar linkage em larga escala de forma segura e eficiente, mesmo em ambientes regulatórios distintos [Williamson 2020, Population Data BC 2024].

8.5.3. LLM e IA para RL

Nesse contexto de evolução, destaca-se também o uso de modelos de linguagem avançados (LLMs) aplicados à vinculação de registros, como exemplificado pelo LinkTransformer (LT), proposto por [Arora and Dell 2023].

O LT é uma ferramenta que oferece uma forma intuitiva de integrar Large Language Models (LLMs) nas análises de dados, promovendo eficiência e precisão na vinculação de registros. O LT preenche a lacuna entre os métodos tradicionais de correspondência de strings e os recursos avançados dos LLMs. A proposta do LT inclui uma API intuitiva, que permite aos usuários realizar tarefas de limpeza de dados de maneira simplificada, suportando múltiplos idiomas e oferecendo integração com modelos pré-treinados do Hugging Face ou OpenAI [Arora and Dell 2023]. A ferramenta visa democratizar o acesso a métodos sofisticados de vinculação de registros, tornando-os acessíveis mesmo para pesquisadores sem experiência em aprendizado profundo.

Ao superar as limitações das técnicas convencionais de correspondência de strings, o LT demonstra uma melhoria significativa na precisão, especialmente em tarefas como a vinculação de dados históricos [Arora and Dell 2023]. Com isso, o LT se destaca ao permitir que os pesquisadores realizem análises de dados complexas de maneira mais eficiente, sem necessidade de um profundo conhecimento em codificação ou estruturas avançadas de aprendizado de máquina.

8.5.4. RL de alto desempenho

Nos últimos anos, as pesquisas em Record Linkage têm se concentrado na área de ciência da computação, com ênfase na melhoria da eficiência computacional, seja por meio da paralelização de processos e de algoritmos avançados de busca ou na aplicação de modelos sofisticados de machine learning e estratégias para estimar erros de vinculação mesmo na ausência de dados rotulados. Apesar do avanço técnico, diversas análises apontam que nenhum método superou de forma consistente o modelo clássico de Fellegi-Sunter, especialmente em aplicações práticas de larga escala que envolvem dezenas de milhões de registros [Christen 2012]. Contudo, a demanda por modelos de alto desempenho tem motivados os pesquisadores a modelar soluções de RL baseados em sistemas computacionais especialistas ou placas gráficas [Rasch et al. 2019].

8.5.5. Quantificação e mitigação de vieses em RL

Os erros de RL podem provocar vieses importantes na análise *downstream* quando não são aleatórios e potencialmente favorecem um grupo demográfico em detrimento de outro [Doidge and Harron 2019]. Estes erros podem ocasionar grandes alterações nos achados, especialmente nas ferramentas mais modernas de linkage, que implementam modelos inteligentes ou mesmo baseados em heurísticas no seu *pipeline*. Faz-se, então, necessária uma reflexão que produza soluções capazes de vincular registros em bases de dados distintas que objetive a otimização da justiça algorítmica em conjunto com a acurácia ou precisão [Vatsalan et al. 2020].

8.6. Tendências e desafios de pesquisa

Dentre as as principais tendências apontadas por [Harron et al. 2016] e [Dong and Srivastava 2013], a quantificação e estudo dos impactos causados pelos vieses de vinculação são temas prioritários que devem ser endereçados por pesquisadores da área. Em seu trabalho, [Christen 2019] aponta, como tendências atuais no BDL, o uso de aprendizado de máquina para melhorar a acurácia do pareamento e o desenvolvimento de técnicas de PPRL para garantir a privacidade. No entanto, desafios como a integração de dados em tempo real e a heterogeneidade das fontes de dados ainda persistem, exigindo avanços metodológicos e tecnológicos. Além destes tópicos, nesta seção, discutiremos a avaliação quantitativa de vieses e as evoluções no estado-da-arte nas disciplinas de gestão de dados e inteligência artificial que podem ser incorporadas ao BDL.

Tradicionalmente, os modelos de Record Linkage (RL) eram treinados de forma específica para cada conjunto de dados, exigindo um novo processo de treinamento para cada domínio [Tang 2022, Li 2020, Brunner and Stockinger 2020]. No entanto, estudos como o de [Tang 2022] propõem o uso de modelos genéricos que, treinados sobre múltiplos domínios, conseguem generalizar para novos datasets com desempenho competitivo. Essa abordagem baseia-se na utilização de modelos de linguagem pré-treinados (PLMs) como BERT e RoBERTa, que encapsulam conhecimento semântico transferível e reduzem a necessidade de engenharia de atributos.

Com a chegada dos LLMs, como GPT-3, tornou-se viável realizar tarefas de RL com pouca ou nenhuma anotação supervisionada. [Peeters and Bizer 2023] demonstram que, por meio de prompt engineering e in-context learning, é possível obter resultados

comparáveis a modelos supervisionados, mesmo em cenários desafiadores. Esse modelo é promissor, especialmente para aplicações em que a anotação manual de dados é inviável.

O trabalho de [Arora and Dell 2023] apresenta o LinkTransformer, um pacote unificado que permite realizar RL com diversos tipos de codificadores (bi-encoders, cross-encoders e prompting com LLMs). Essa padronização do pipeline de RL representa um passo importante em direção à democratização da tecnologia, facilitando a reprodutibilidade e a comparabilidade entre métodos.

Pesquisas aplicadas como as de [Barbosa et al. 2020] e [Coeli 2021] reforçam a importância de desenvolver sistemas robustos e escaláveis. Em especial, eles destacam os desafios enfrentados ao aplicar RL em bases de dados públicas de saúde no Brasil, com dados ruidosos, ausentes ou inconsistentes. Essas iniciativas evidenciam a necessidade de soluções eficientes, voltadas à realidade dos sistemas de informação governamentais e científicos.

Apesar dos avanços notáveis nas técnicas de RL, inúmeros desafios ainda persistem e demandam atenção da comunidade científica. Um dos obstáculos mais recorrentes refere-se à escalabilidade computacional. Modelos baseados em cross-encoders, que processam pares de registros de forma conjunta para capturar interações contextuais, são computacionalmente custosos [Li 2020, Arora and Dell 2023]. Especialmente em bases de dados com milhões de combinações possíveis. Alternativas mais eficientes, como os bi-encoders, permitem maior paralelização e pré-cálculo de embeddings, mas frequentemente resultam em perda de precisão [Li 2020, Arora and Dell 2023]. Equilibrar desempenho preditivo e viabilidade computacional é um desafio da pesquisa contemporânea.

Além disso, embora *benchmarks* amplamente utilizados sejam úteis para validação inicial, eles frequentemente não refletem a complexidade dos dados reais [Coeli 2021, Tang 2022]. A maioria desses conjuntos é bem curada, com atributos bem definidos e baixo nível de ruído. Entretanto, aplicações reais, como aquelas documentadas por [Coeli 2021] no contexto do sistema de saúde brasileiro, mostram que a ausência de identificadores únicos, a presença de erros tipográficos e a heterogeneidade de esquemas são fatores comuns. A carência de benchmarks realistas compromete a avaliação generalizável dos modelos e evidencia a necessidade de conjuntos de dados mais desafiadores e representativos [Abramitzky 2021, Barbosa et al. 2020].

Outro desafio significativo é a falta de explicabilidade dos modelos modernos. À medida que técnicas baseadas em deep learning e modelos de linguagem se tornam dominantes, os critérios utilizados para classificar registros como correspondentes tornam-se menos transparentes. Sendo especialmente preocupante em domínios sensíveis, como justiça, saúde pública e análise demográfica, onde decisões automatizadas precisam ser auditáveis e justificáveis [Bhattacharya and Getoor 2007, Wilson 2011].

Além disso, a questão do viés algorítmico vem ganhando relevância. Modelos de linguagem pré-treinados são treinados em grandes corpora da web, sujeitos a estereótipos e desigualdades históricas [Gehman 2020, Peeters et al. 2023]. Quando utilizados em tarefas de RL, esses vieses podem se propagar e até se intensificar, impactando negativamente populações marginalizadas ou regiões com baixa representação nos dados de treinamento. Estratégias para mitigação de viés e avaliação de equidade nos out-

puts dos sistemas ainda são escassas e representam uma lacuna importante na literatura [Peeters and Bizer 2023, Peeters et al. 2023, Tang 2022].

Em síntese, a pesquisa em RL atravessa um momento de transformação. A incorporação de modelos de linguagem pré-treinados trouxe ganhos expressivos em termos de acurácia e adaptabilidade, ao mesmo tempo em que exigiu novas abordagens para lidar com escalabilidade, explicabilidade e robustez. O futuro do campo dependerá da capacidade da comunidade em enfrentar esses desafios de forma ética e técnica, desenvolvendo soluções que sejam não apenas eficazes, mas também confiáveis, transparentes e socialmente responsáveis.

Referências

[Abramitzky 2021] Abramitzky, R. e. a. (2021). Automated linking of historical data. *Journal of Economic Literature*, 59(3):865–918.

[Arora and Dell 2023] Arora, R. and Dell, R. (2023). Linktransformer: A unified package for record linkage with transformer language models.

[Barbosa et al. 2020] Barbosa, G. C. G., Ali, M. S., Araujo, B., Reis, S., Sena, S., Ichihara, M. Y. T., Pescarini, J., Fiaccone, R. L., Amorim, L. D., Pita, R., Barreto, M. E., Smeeth, L., and Barreto, M. L. (2020). Cidacs-rl: a novel indexing search and scoring-based record linkage system for huge datasets with high accuracy and scalability. *BMC Medical Informatics and Decision Making*, 20(1).

[Barreto et al. 2019] Barreto, M. L., Ichihara, M. Y., Almeida, B. A., Barreto, M. E., Cabral, L., Fiaccone, R. L., Carreiro, R. P., Teles, C. A. S., Pitta, R., Penna, G. O., Barral-Netto, M., Ali, M. S., Barbosa, G., Denaxas, S., Rodrigues, L. C., and Smeeth, L. (2019). The centre for data and knowledge integration for health (cidacs): Linking health and social data in brazil. *International Journal of Population Data Science*, 4(2).

[Barreto et al. 2022] Barreto, M. L., Ichihara, M. Y., Pescarini, J. M., Ali, M. S., Borges, G. L., Fiaccone, R. L., Ribeiro-Silva, R. D. C., Teles, C. A., Almeida, D., Sena, S., Carreiro, R. P., Cabral, L., Almeida, B. A., Barbosa, G. C. G., Pita, R., Barreto, M. E., Mendes, A. A. F., Ramos, D. O., Brickley, E. B., and Smeeth, L. (2022). Cohort profile: The 100 million brazilian cohort. *International Journal of Epidemiology*, 51(2):E27–E38.

[Bhattacharya and Getoor 2007] Bhattacharya, I. and Getoor, L. (2007). Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):5–es.

[Blake et al. 2022] Blake, H., Sharples, L., Harron, K., Van der Meulen, J., and Walker, K. (2022). Linkage of national clinical datasets without patient identifiers using probabilistic methods. *International Journal of Population Data Science*, 7(3).

[Bloom 1970] Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426.

[Brunner and Stockinger 2020] Brunner, U. and Stockinger, K. (2020). Entity matching with transformer architectures-a step forward in data integration. In *23rd International Conference on Extending Database Technology*, pages 463–473. OpenProceedings.

[Camargo and Coeli 2011] Camargo, K. and Coeli, C. (2011). Openreclink a free and open source solution for probabilistic record linkage. In *AMERICAN JOURNAL OF EPIDEMIOLOGY*, volume 173, pages S108–S108. OXFORD UNIV PRESS INC JOURNALS DEPT, 2001 EVANS RD, CARY, NC 27513 USA.

[Camargo Jr and Coeli 2000] Camargo Jr, K. R. d. and Coeli, C. M. (2000). Reclink: aplicativo para o relacionamento de bases de dados, implementando o método probabilistic record linkage. *Cadernos de Saúde Pública*, 16:439–447.

[Camargo Jr and Coeli 2015] Camargo Jr, K. R. d. and Coeli, C. M. (2015). Going open source: some lessons learned from the development of openreclink. *Cadernos De Saude Publica*, 31:257–263.

[Camargo Júnior and Coeli 2002] Camargo Júnior, K. and Coeli, C. (2002). Reclink ii: Guia do usuário.

[Camargo Junior and Coeli 2006] Camargo Junior, K. R. d. and Coeli, C. M. (2006). Reclink 3: nova versão do programa que implementa a técnica de associação probabilística de registros (probabilistic record linkage). *Cad. saúde colet.,(Rio J.)*, pages 399–404.

[Cha 2008] Cha, S.-H. (2008). Taxonomy of nominal type histogram distance measures. *City*, 1(2):1.

[Christen 2011] Christen, P. (2011). A survey of indexing techniques for scalable record linkage and deduplication. *IEEE transactions on knowledge and data engineering*, 24(9):1537–1555.

[Christen 2012] Christen, P. (2012). *Data matching: Concepts and techniques for record linkage, entity resolution, and duplicate detection*. Springer Science & Business Media.

[Christen 2014] Christen, P. (2014). Privacy aspects in big data integration: Challenges and opportunities. In *Proceedings of the First International Workshop on Privacy and Secuirty of Big Data*, PSBD '14, page 1, New York, NY, USA. Association for Computing Machinery.

[Christen 2019] Christen, P. (2019). Data linkage: The big picture. *Harvard Data Science Review*, 1(2).

[Christen et al. 2020] Christen, P., Ranbaduge, T., and Schnell, R. (2020). *Linking Sensitive Data: Methods and Techniques for Practical Privacy-Preserving Information Sharing*. Springer Cham.

[Christen et al. 2025] Christen, P., Schnell, R., and Vidanage, A. (2025). Information leakage in data linkage. *arXiv preprint arXiv:2505.08596*.

[Christen et al. 2024] Christen, P., Ziyad, S., Vidanage, A., Nanayakkara, C., and Schnell, R. (2024). A single parameter method for secure privacy preserving record linkage. *International Journal of Population Data Science*, 9(5).

[Coeli et al.] Coeli, C., Pinheiro, R., Camargo Jr, K., et al. Achievements and challenges for employing record linkage techniques in health research and evaluation in brazil. epidemiol e serviços saúde. 2015.

[Coeli 2021] Coeli, C. M. e. a. (2021). Record linkage under suboptimal conditions for data-intensive evaluation of primary care in rio de janeiro, brazil. *BMC Medical Informatics and Decision Making*, 21(1):190.

[Damerau 1964] Damerau, F. J. (1964). A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176.

[Dillinger and Manolios 2004] Dillinger, P. C. and Manolios, P. (2004). Fast and accurate bitstate verification for spin. In Graf, S. and Mounier, L., editors, *Model Checking Software*, pages 57–75, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Doidge and Harron 2019] Doidge, J. C. and Harron, K. L. (2019). Reflections on modern methods: linkage error bias. *International journal of epidemiology*, 48(6):2050–2060.

[Dong and Srivastava 2013] Dong, X. L. and Srivastava, D. (2013). Big data integration. In *Proceedings of the International Conference on Data Engineering*, pages 1245–1248.

[dos Santos Filho 2008] dos Santos Filho, W. (2008). Algoritmo paralelo e eficiente para o problema de pareamento de dados.

[Dunn and Chief 1946] Dunn, H. L. and Chief, F. (1946). Record linkage.

[Durham et al. 2013] Durham, E. A., Kantarcioglu, M., Xue, Y., Toth, C., Kuzu, M., and Malin, B. (2013). Composite bloom filters for secure record linkage. *IEEE Transactions on Knowledge and Data Engineering*, 26(12):2956–2968.

[Dusetzina 2014] Dusetzina, S. B. e. a. (2014). *Linking Data for Health Services Research: A Framework and Instructional Guide*. Agency for Healthcare Research and Quality (US), Rockville (MD). Report No.: 14-EHC033-EF. Available at: https://pubmed.ncbi.nlm.nih.gov/25392892/. Accessed: 6 May 2025.

[Fellegi and Sunter 1969] Fellegi, I. P. and Sunter, A. B. (1969). A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210.

[Ferreira et al. 2005] Ferreira, R. A., Meira, W., Guedes, D., Drummond, L. M., Coutinho, B., Teodoro, G., Tavares, T., Araujo, R., and Ferreira, G. T. (2005). Anthill: A scalable run-time environment for data mining applications. In *17th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'05)*, pages 159–166. IEEE.

[Fletcher et al. 2018] Fletcher, S., Islam, M. Z., et al. (2018). Comparing sets of patterns with the jaccard index. *Australasian Journal of Information Systems*, 22.

[Franke et al. 2021] Franke, M., Sehili, Z., Rohde, F., and Rahm, E. (2021). Evaluation of hardening techniques for privacy-preserving record linkage. In *EDBT*, pages 289–300.

[Gehman 2020] Gehman, S. e. a. (2020). Realtoxicityprompts: Evaluating neural toxic degeneration in language models. *arXiv preprint arXiv:2009.11462*.

[Grannis et al. 2002] Grannis, S. J., Overhage, J. M., and McDonald, C. J. (2002). Analysis of identifier performance using a deterministic linkage algorithm. In *Proceedings of the AMIA Symposium*, page 305.

[Guillen et al. 2017] Guillen, L. C., Domenico, J., Camargo, K., Pinheiro, R., and Coeli, C. (2017). Match quality of a linkage strategy based on the combined use of a statistical linkage key and the levenshtein distance to link birth to death records in brazil.: Ijpds (2017) issue 1, vol 1: 036, proceedings of the ipdln conference (august 2016). *International Journal of Population Data Science*, 1(1).

[Harron et al. 2016] Harron, K., Goldstein, H., and Dibben, C. (2016). *Methodological developments in data linkage*. Wiley.

[Herzog et al. 2007] Herzog, T. N., Scheuren, F. J., Winkler, W. E., Herzog, T. N., Scheuren, F. J., and Winkler, W. E. (2007). Phonetic coding systems for names. *Data Quality and Record Linkage Techniques*, pages 115–121.

[Jordão and Rosa 2012] Jordão, C. C. and Rosa, J. L. G. (2012). Metaphone-pt_br: the phonetic importance on search and correction of textual information. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 297–305. Springer.

[Junger 2006] Junger, W. L. (2006). Estimação de parâmetros em relacionamento probabilístico de bancos de dados: uma aplicação do algoritmo em para o reclink. *Cad. saúde colet.,(Rio J.)*, pages 225–232.

[Junior et al. 2018] Junior, A. A. G., Pereira, R. G., Gurgel, E. I., Cherchiglia, M., Dias, L. V., Ávila, J. D., Santos, N., Reis, A., Acurcio, F. A., and Junior, W. M. (2018). Building the national database of health centred on the individual: administrative and epidemiological record linkage-brazil, 2000-2015. *International Journal of Population Data Science*, 3(1):446.

[Karakasidis and Koloniari 2017] Karakasidis, A. and Koloniari, G. (2017). Phonetics-based parallel privacy preserving record linkage. pages 179–190.

[Kirsch and Mitzenmacher 2006] Kirsch, A. and Mitzenmacher, M. (2006). Less hashing, same performance: Building a better bloom filter. In *Proceedings of the European Symposium on Algorithms*, volume 4168, pages 456–467.

[Li 2020] Li, Y. e. a. (2020). Deep entity matching with pre-trained language models. *arXiv preprint arXiv:2004.00584*.

[Oliveira et al. 2016] Oliveira, G. P. d., Bierrenbach, A. L. d. S., Camargo Júnior, K. R. d., Coeli, C. M., and Pinheiro, R. S. (2016). Accuracy of probabilistic and deterministic record linkage: the case of tuberculosis. *Revista de saude publica*, 50:49.

[Paixao et al. 2021] Paixao, E. S., Cardim, L. L., Falcao, I. R., Ortelan, N., Silva, N. D. J., Rocha, A. D. S., Sena, S., Almeida, D., Ramos, D. O., Alves, F. J. O., Bispo, N., Ali, S., Fiaccone, R., Rodrigues, M., Smeeth, L., Brickley, E. B., Cabral, L., Teles, C., Costa, M. C. N., and Teixeira, M. G. (2021). Cohort profile: Centro de integração de dados e conhecimentos para saúde (cidacs) birth cohort. *International Journal of Epidemiology*, 50(1):37–38H.

[Peeters and Bizer 2023] Peeters, R. and Bizer, C. (2023). Using chatgpt for entity matching. In *European Conference on Advances in Databases and Information Systems*, pages 221–230. Springer Nature Switzerland.

[Peeters et al. 2023] Peeters, R., Steiner, A., and Bizer, C. (2023). Entity matching using large language models. *arXiv preprint arXiv:2310.11244*.

[Pita et al. 2018] Pita, R., Pinto, C., Sena, S., Fiaccone, R., Amorim, L., Reis, S., Barreto, M. L., Denaxas, S., and Barreto, M. E. (2018). On the accuracy and scalability of probabilistic data linkage over the brazilian 114 million cohort. *IEEE Journal of Biomedical and Health Informatics*, 22(2):346–353.

[Population Data BC 2024] Population Data BC (2024). About us.

[Ranbaduge and Schnell 2020] Ranbaduge, T. and Schnell, R. (2020). Securing bloom filters for privacy-preserving record linkage. In *Proceedings of the 29th ACM international conference on information & knowledge management*, pages 2185–2188.

[Rasch et al. 2019] Rasch, A., Schulze, R., Gorus, W., Hiller, J., Bartholomäus, S., and Gorlatch, S. (2019). High-performance probabilistic record linkage via multi-dimensional homomorphisms. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, SAC '19, page 526–533, New York, NY, USA. Association for Computing Machinery.

[Sanni Ali et al. 2019] Sanni Ali, M., Ichihara, M. Y., Lopes, L. C., Barbosa, G. C. G., Pita, R., Carreiro, R. P., dos Santos, D. B., Ramos, D., Bispo, N., Raynal, F., Canuto, V., de Araujo Almeida, B., Fiaccone, R. L., Barreto, M. E., Smeeth, L., and Barreto, M. L. (2019). Administrative data linkage in brazil: Potentials for health technology assessment. *Frontiers in Pharmacology*, 10(SEP).

[Santos et al. 2007] Santos, W., Teixeira, T., Machado, C., Meira Jr, W., Ferreira, R., Guedes, D., and Da Silva, A. S. (2007). A scalable parallel deduplication algorithm. In *19th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'07)*, pages 79–86. IEEE.

[Schnell 2015] Schnell, R. (2015). Privacy-preserving record linkage. *Methodological developments in data linkage*, pages 201–225.

[Schnell et al. 2009] Schnell, R., Bachteler, T., and Reiher, J. (2009). Privacy-preserving record linkage using bloom filters. *BMC Medical Informatics and Decision Making*, 9:1–11.

[Schnell et al. 2011] Schnell, R., Bachteler, T., and Reiher, J. (2011). A novel error-tolerant anonymous linking code. Available at SSRN 3549247.

[Shukla 2015] Shukla, V. (2015). *Elasticsearch for Hadoop*. Packt Publishing Ltd.

[Tang 2022] Tang, J. e. a. (2022). Generic entity resolution models. In *NeurIPS 2022 First Table Representation Workshop*.

[Vaiwsri et al. 2018] Vaiwsri, S., Ranbaduge, T., and Christen, P. (2018). Reference values based hardening for bloom filters based privacy-preserving record linkage. In *Australasian Conference on Data Mining*, pages 189–202. Springer.

[Vatsalan et al. 2020] Vatsalan, D., Yu, J., Henecka, W., and Thorne, B. (2020). Fairness-aware privacy-preserving record linkage. In *International Workshop on Data Privacy Management*, pages 3–18. Springer.

[Vidal et al. 2006] Vidal, E., Coeli, C., Pinheiro, R., and Camargo, K. (2006). Mortality within 1 year after hip fracture surgical repair in the elderly according to postoperative period: a probabilistic record linkage study in brazil. *Osteoporosis international*, 17:1569–1576.

[Vidanage et al. 2023] Vidanage, A., Christen, P., Ranbaduge, T., and Schnell, R. (2023). A vulnerability assessment framework for privacy-preserving record linkage. *ACM Transactions on Privacy and Security*, 26(3):1–31.

[Vidanage et al. 2019] Vidanage, A., Ranbaduge, T., Christen, P., and Schnell, R. (2019). Efficient pattern mining based cryptanalysis for privacy-preserving record linkage. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1698–1701. IEEE.

[Vieira et al. 2017] Vieira, C., Coeli, C., Aguiar, F., Camargo Jr, K., Pinheiro, R., and Flores, P. (2017). Effect of short interdelivery interval between the first and second pregnancies in adolescence on low birth weight. *International Journal of Population Data Science*, 1(1):37.

[Williamson 2020] Williamson, E. J. e. a. (2020). Opensafely: Factors associated with covid-19 death in 17 million patients. *Nature*, 584(7821):430–436.

[Wilson 2011] Wilson, D. R. (2011). Beyond probabilistic record linkage: Using neural networks and complex features to improve genealogical record linkage. In *The 2011 International Joint Conference on Neural Networks*, pages 9–14. IEEE.

