Chapter 4

Learned Index: Perspectivas e Desafios na Gestão de Dados de Saúde

Guilherme Waldschimidt (IC-UFBA), Alberto Sironi (IC-UFBA), Laís Sacramento (IC-UFBA/Cidacs-Fiocruz), Mirlei Moura (IC-UFBA), Maíra Lima Souza (Cidacs-Fiocruz), Nívea Bispo (IME-UFBA), Robespierre Pita (IC-UFBA/Cidacs-Fiocruz)

Abstract

Healthcare-focused computational solutions, such as record linkage, image pattern recognition, data mining for diagnostic support, or natural language information extraction, often utilize indexing methods to optimize the retrieval and access of database records. This course introduces the concept of the Learned Index, which leverages machine learning models to create index structures that outperform traditional algorithms. The engagement of the Applied Health Computing community in this new research area has the potential to drive the development of various solutions and applications with a direct impact on clinical practice, health surveillance, management, and public policy formulation. This mini-course is designed for computing and healthcare students and researchers while offering valuable insights for database professionals and managers.

Resumo

Soluções computacionais voltadas à saúde, tais como record linkage, reconhecimento de padrões em imagens, mineração de dados para suporte a diagnósticos ou extração de informação em linguagem natural frequentemente recorrem a métodos de indexação para otimizar a recuperação e acesso aos registros de uma base de dados. Este curso apresenta o conceito de Learned Index, que emprega modelos de machine learning para produzir estruturas de índices capazes de superar algoritmos tradicionais. Acreditamos que a participação da comunidade de Computação Aplicada à Saúde neste novo tópico de pesquisa tem o potencial de levar ao desenvolvimento de diversas soluções e

aplicações com impacto direto na prática clínica, da vigilância em saúde, gestão e formulação de políticas públicas. Tendo um caráter teórico apoiado por demonstrações práticas de criação, uso e validação de índices, este minicurso se dirige a estudantes e pesquisadores da computação e saúde, além de trazer insights para profissionais e gestores de bancos de dados.

4.1. Introdução

Sistemas Gerenciadores de Bancos de Dados (SGBD) são coleções de programas que compartilham o objetivo de apoiar o armazenamento, acesso e recuperação de dados [Silberschatz et al., 2020]. Estes sistemas são responsáveis por suportar diversas tarefas de gestão e administração de dados, implementando todas as funcionalidades e restrições impostas pelos diferentes modelos. Um modelo de dados estabelece um conjunto de regras e abstrações para representação, descrição, relacionamento, semântica, consistência e coerência dos fenômenos registrados num mini-mundo. O modelo relacional define tabelas — ou relações — para orientar o armazenamento físico dos dados. Neste sentido, cada registro numa tabela descreve os fenômenos ou entidades observados usando um número fixo de atributos. Administradores de bancos de dados (DBA, do inglês *database administrador*) e Engenheiros de Dados são responsáveis por projetar, manter e manipular dados nestes sistemas.

Os SGBD tradicionais são comumente utilizados pelos Sistemas de Informação projetados para administração do Sistema Único de Saúde (SUS), seja na atenção primária, vigilância em saúde ou na administração de políticas públicas diversas[1,2]. Dentre eles figuram o Sistema de Informações sobre Nascidos Vivos (Sinasc), Registro de Eventos em Saúde Pública (Resp-Microcefalia), Sistema de Informação de Agravos de Notificação (Sinan), e-SUS Notifica, Sistema de Informações sobre Mortalidade (SIM), etc. Como descrito no sítio eletrônico do principal mantenedor das bases de dados de saúde do país, o Departamento de Informática do Sistema Único de Saúde (DATASUS), as diretrizes de gestão dos bancos de dados que suportam seus sistemas de informação são estabelecidas pela Metodologia de Administração de Dados (MAD), que normatiza regras gerais de uso e otimização. As especificações encontradas no MAD conformam os principais aspectos do modelo de dados, incluindo quais os SGBDs, definições de padrões, regras técnicas, normas de modelagem e dicionários de dados [DATASUS, 2024]. Fora deste ambiente de produção, versões pseudonimizadas destas bases relacionais são disponibilizadas pelo DATASUS para fins de transparência e suporte à pesquisa em saúde pública[3].

O Módulo de 'Notificação Covid-19' do SI e-SUS Notifica, por exemplo, concentra a coleta de dados sobre todos os casos leves de Síndrome Gripal (SG), execução e resultados dos testes laboratoriais, a classificação final em relação à

[1] https://www.gov.br/saude/pt-br/composicao/svsa/vigilancia-de-doencas-cronicas-nao-transmissiveis/sistemas-de-informacao-em-saude

[2] https://www.gov.br/saude/pt-br/assuntos/noticias/2023/julho/entenda-as-principais-caracteristicas-dos-sistemas-de-informacao-do-ministerio-da-saude

[3] https://datasus.saude.gov.br/informacoes-de-saude-tabnet/

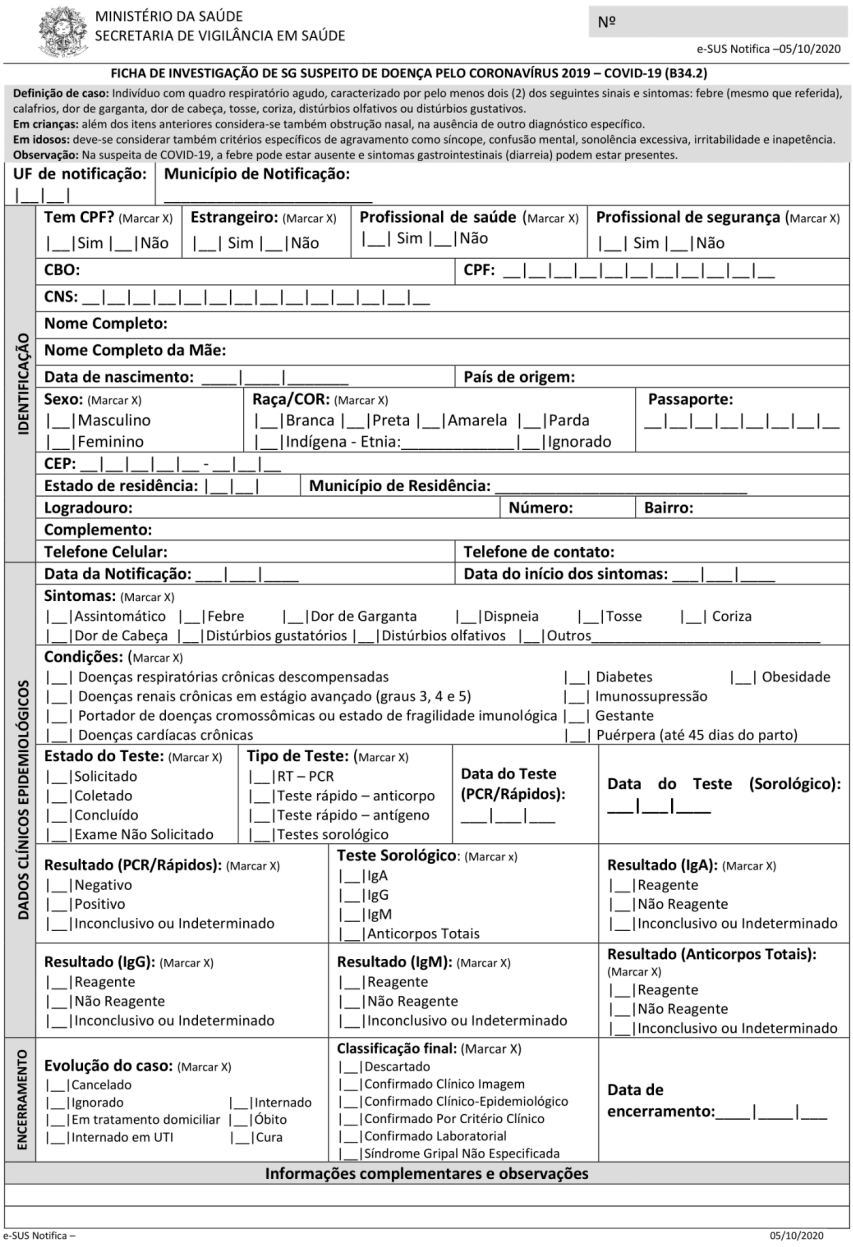
Covid-19 (suspeito, confirmado ou descartado) e o desfecho (cura ou óbito). Esta adaptação suportou a vigilância epidemiológica durante a pandemia, adaptando rapidamente um SI para cumprir com a notificação compulsória imediata da Covid-19, definida em maio de 2022 [Ministério da Saúde, 2022]. Na Figura 4.1 é possível verificar o formulário físico utilizado para coletar os dados primários. Em seu preâmbulo, a ficha inclui informações para a sua própria identificação, incluindo seu número único, unidade federativa e município de notificação. Os três segmentos subsequentes concentram dados de identificação do indivíduo, suas informações clínicas e epidemiológicas e desfecho do caso.

Para fins de demonstração, o modelo lógico exposto na Figura 4.2., denominado `covid_bd`, mimetiza o banco de dados do Módulo de 'Notificação Covid-19' do e-SUS Notifica, orientado a partir dos segmentos da ficha na Figura 4.1. O `covid_bd` é composto por quatro tabelas. A tabela `tbl_identificacao` concentra os dados derivados do primeiro segmento da ficha na Figura 4.1, incluindo nome, cadastro de pessoa física (CPF), a raça/cor autodeclarada, sexo e data de nascimento. Na `tbl_info_clinicas` estão as variáveis de aferição de temperatura, saturação de oxigênio no sangue, data da coleta, data do resultado, tipo e resultado de cada teste. Parte das informações que subsidiam a investigação epidemiológica estão na tabela `tbl_info_epidemiologicas`, que concentra variáveis binárias indicando se existem sintomas gripais, distúrbios olfativos ou gustativos e dispneia. Por fim, as variáveis de data de admissão, evolução do caso, diagnóstico/classificação final e data de encerramento estão na tabela `tbl_adm_encerramento`.

É importante ressaltar que a variável `id_caso`, original da tabela `tbl_identificacao` é exportada para as demais tabelas para garantir o relacionamento entre elas. Estes relacionamentos consistem na associação dos fenômenos e entidades fisicamente armazenadas sob orientação do SGBD. Por exemplo, a partir do `id_caso = 17781`, cujo paciente é Alfredo Casanova, é possível varrer toda a tabela `tbl_info_clinicas` para verificar se existem coletas e resultados de testes ou procurar pelo seu diagnóstico e desfecho desta infecção na `tbl_adm_encerramento`. Outro aspecto da notação estabelece entre colchetes as restrições de mínimo e máximo de associações nestas relações. No caso da relação entre as tabelas `tbl_identificacao` e `tbl_info_clinicas`, um mesmo caso pode ter nenhuma ou muitos testes associados, denotado pelo [0, N]. Contudo, cada teste nesta relação só pode ser registrado se já existe um caso cadastrado e estar associado a apenas um caso, representado pela notação [1, 1]. Esta restrição permite que haja dois testes para o `id_caso = 29819`.

Os SGBD que implementam o modelo relacional oferecem uma linguagem de consulta estruturada chamada (SQL, do inglês *structured query language*) usada para recuperar a informação armazenada pelo sistema [Silberschatz et al., 2020]. Em seu livro, Elmasri et al., 2016 argumenta que a maioria das consultas aos dados recupera apenas uma pequena porção de todos os seus registros, tornando ineficiente a estratégia de varrer todo o dado disponível para responder a cada busca. Os *datasets* resultantes de consultas SQL frequentemente suportam tarefas de processamento analítico online (OLAP, do inglês *online analytical processing*). Por exemplo, após a saída do SGBD para a consulta "*quais os casos confirmados e descartados mediante testes coletados*

entre 01/01/2022 e 31/12/2022?", diversas perguntas sobre os casos suspeitos de Covid-19 ou modelos analíticos podem ser elaborados. Analistas de dados podem



MINISTÉRIO DA SAÚDE
SECRETARIA DE VIGILÂNCIA EM SAÚDE

Nº

e-SUS Notifica –05/10/2020

FICHA DE INVESTIGAÇÃO DE SG SUSPEITO DE DOENÇA PELO CORONAVÍRUS 2019 – COVID-19 (B34.2)

Definição de caso: Indivíduo com quadro respiratório agudo, caracterizado por pelo menos dois (2) dos seguintes sinais e sintomas: febre (mesmo que referida), calafrios, dor de garganta, dor de cabeça, tosse, coriza, distúrbios olfativos ou distúrbios gustativos.
Em crianças: além dos itens anteriores considera-se também obstrução nasal, na ausência de outro diagnóstico específico.
Em idosos: deve-se considerar também critérios específicos de agravamento como síncope, confusão mental, sonolência excessiva, irritabilidade e inapetência.
Observação: Na suspeita de COVID-19, a febre pode estar ausente e sintomas gastrointestinais (diarreia) podem estar presentes.

UF de notificação: |__|__| **Município de Notificação:** _____

IDENTIFICAÇÃO

Tem CPF? (Marcar X) |__|Sim |__|Não	**Estrangeiro:** (Marcar X) |__| Sim |__|Não	**Profissional de saúde** (Marcar X) |__| Sim |__|Não	**Profissional de segurança** (Marcar X) |__| Sim |__|Não
CBO:		**CPF:** __|__|__|__|__|__|__|__|__|__|__	
CNS: __|__|__|__|__|__|__|__|__|__|__|__|__|__|__			
Nome Completo:			
Nome Completo da Mãe:			
Data de nascimento: ____|____|_____		**País de origem:**	
Sexo: (Marcar X) |__|Masculino |__|Feminino	**Raça/COR:** (Marcar X) |__|Branca |__|Preta |__|Amarela |__|Parda |__|Indígena - Etnia:_____|__|Ignorado		**Passaporte:** __|__|__|__|__|__|__|__|__
CEP: __|__|__|__|__ - __|__|__			
Estado de residência: |__|__|	**Município de Residência:** _____		
Logradouro:		**Número:**	**Bairro:**
Complemento:			
Telefone Celular:		**Telefone de contato:**	

DADOS CLÍNICOS EPIDEMIOLÓGICOS

Data da Notificação: ___|___|____		**Data do início dos sintomas:** ___|___|____	
Sintomas: (Marcar X) |__|Assintomático |__|Febre |__|Dor de Garganta |__|Dispneia |__|Tosse |__| Coriza |__|Dor de Cabeça |__|Distúrbios gustatórios |__|Distúrbios olfativos |__|Outros_____			
Condições: (Marcar X) |__| Doenças respiratórias crônicas descompensadas |__| Diabetes |__| Obesidade |__| Doenças renais crônicas em estágio avançado (graus 3, 4 e 5) |__| Imunossupressão |__| Portador de doenças cromossômicas ou estado de fragilidade imunológica |__| Gestante |__| Doenças cardíacas crônicas |__| Puérpera (até 45 dias do parto)			
Estado do Teste: (Marcar X) |__|Solicitado |__|Coletado |__|Concluído |__|Exame Não Solicitado	**Tipo de Teste:** (Marcar X) |__|RT – PCR |__|Teste rápido – anticorpo |__|Teste rápido – antígeno |__|Testes sorológico	**Data do Teste (PCR/Rápidos):** ___|___|___	**Data do Teste (Sorológico):** ___|___|____
Resultado (PCR/Rápidos): (Marcar X) |__|Negativo |__|Positivo |__|Inconclusivo ou Indeterminado	**Teste Sorológico**: (Marcar x) |__|IgA |__|IgG |__|IgM |__|Anticorpos Totais		**Resultado (IgA):** (Marcar X) |__|Reagente |__|Não Reagente |__|Inconclusivo ou Indeterminado
Resultado (IgG): (Marcar X) |__|Reagente |__|Não Reagente |__|Inconclusivo ou Indeterminado	**Resultado (IgM):** (Marcar X) |__|Reagente |__|Não Reagente |__|Inconclusivo ou Indeterminado		**Resultado (Anticorpos Totais):** (Marcar X) |__|Reagente |__|Não Reagente |__|Inconclusivo ou Indeterminado

ENCERRAMENTO

Evolução do caso: (Marcar X) |__|Cancelado |__|Ignorado |__|Internado |__|Em tratamento domiciliar |__|Óbito |__|Internado em UTI |__|Cura	**Classificação final:** (Marcar X) |__|Descartado |__|Confirmado Clínico Imagem |__|Confirmado Clínico-Epidemiológico |__|Confirmado Por Critério Clínico |__|Confirmado Laboratorial |__|Síndrome Gripal Não Especificada	**Data de encerramento:**____|____|___

Informações complementares e observações

e-SUS Notifica – 05/10/2020

Figura 4.1. e-SUS Notifica: Ficha de investigação utilizada para coletar dados de vigilância sobre casos de Sindrome Gripal suspeitos de doença pelo Covid-19.

estimar a média móvel de casos de síndrome gripal no município, calcular o volume de testes feitos, verificar a duração média dos casos, computar o risco de óbito com diferentes sintomas, características sociais ou regionais. Cientistas de dados podem conceber modelos preditivos capazes de auxiliar no diagnóstico precoce, estimar a quantidade de leitos ou insumos para atender a elasticidade da demanda ou encontrar

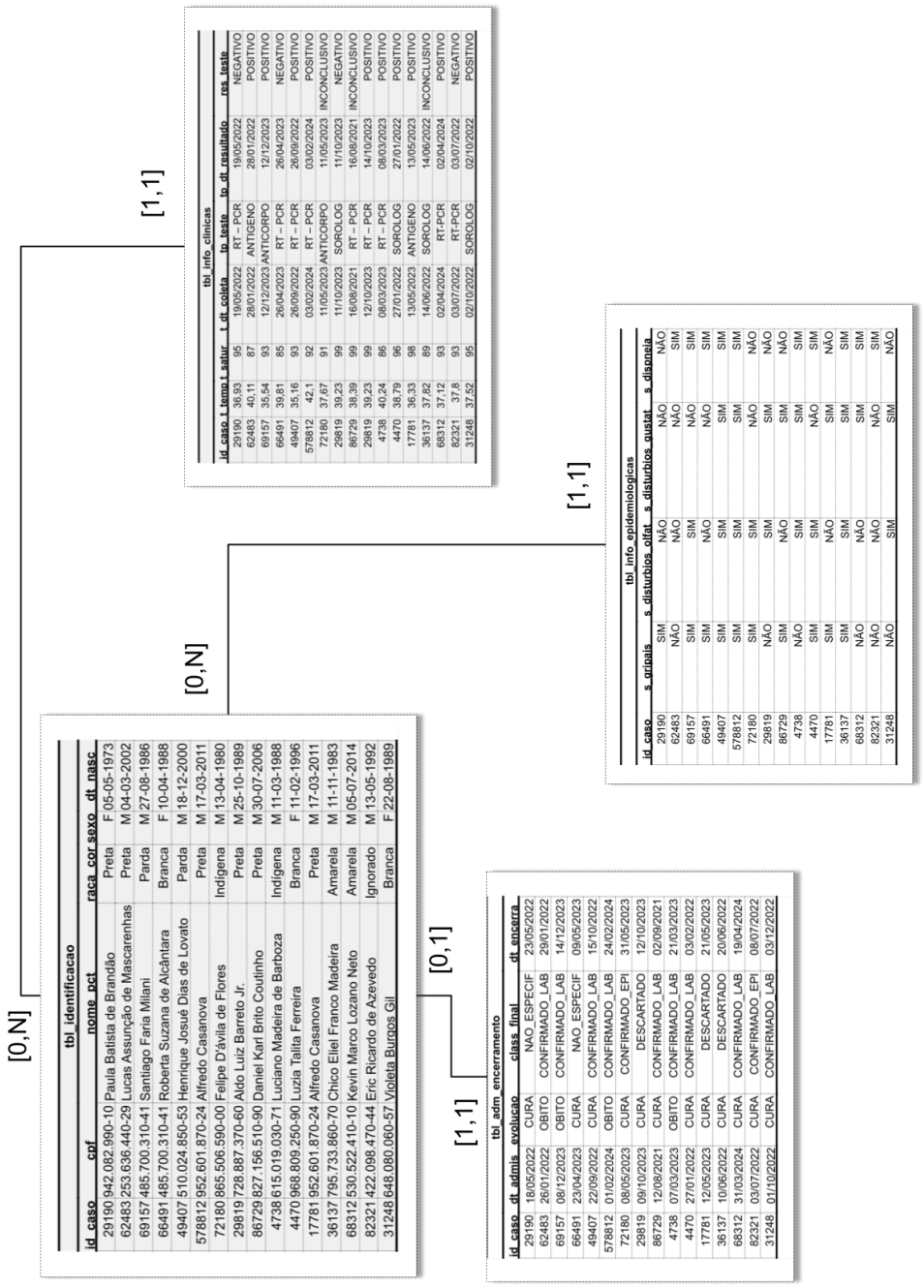


Figura 4.2. Modelo lógico de uma base de dados inspirada na Ficha de investigação de casos da Síndrome Gripal para doença pelo Coronavírus 2019.

padrões de casos através do tempo para definir estratégias de intervenção medicamentosa ou de assistência básica em saúde. Este mesmo *dataset* pode ser utilizado por pesquisadores interessados em estudar iniquidades em saúde, auxiliando na obtenção de conhecimento sobre o acesso desproporcional a testes por pessoas com diferentes marcadores demográficos, a exemplo da raça ou sexo. Alternativamente, o *dataset* pode ser enriquecido a partir da integração com outros sistemas de informação ou bancos de dados, possibilitando análises ainda mais complexas e valiosas. Estas

poderosas ferramentas de suporte à decisão, contudo, possuem alta complexidade computacional e frequentemente classificadas como rotinas intensivas de dados.

Aplicações intensivas de CPU (*central processing unit*) demandam arranjos computacionais de alta capacidade, frequentemente medida em FLOPS (*float operations per second*), como os supercomputadores da lista TOP500[4]. Por outro lado, aplicações *data-intensive* são desenvolvidas para endereçar a complexidade inerente ao dado, seja pelo seu volume, sua estrutura ou a velocidade com que as observações mudam [Kleppmann, 2017]. Em geral, sistemas computacionais capazes de suportar aplicações *data-intensive* são confiáveis, escaláveis e sustentáveis. A confiabilidade é invariavelmente alcançada por algoritmos e componentes de hardware tolerantes a falhas. A propriedade de escalabilidade está relacionada a capacidade de manter boas métricas de latência e vazão em cenários com volume variável de requisições. Por fim, a sustentabilidade é garantida pela incorporação adequada de boas práticas da Engenharia de Software, a exemplo da modularidade, reuso e interoperabilidade. SGBDs adequados para atender os requisitos destas aplicações devem oferecer métodos eficientes de recuperação e acesso aos registros de uma base de dados [Kaur, 2018]. Modelos analíticos e preditivos, como algoritmos de agrupamento de dados, sistema de recomendação, *record linkage* e extração de informação são exemplos de aplicações intensivas que demandam acesso eficiente aos registros de um banco de dados.

Segundo Silberschatz et al., (2020), as tarefas de acesso a registros em SGBD frequentemente extraem pequenas frações de tuplas armazenadas em discos ou alocados em memória. Especialmente quando estes sistemas de dados são usados para suportar aplicações *data-intensive*. O processamento destas consultas pode ser ineficiente se considerarmos varrer todas as observações, selecionando apenas aquelas que correspondem com o valor buscado. Neste caso, métodos de indexação são aplicados para auxiliar organizar os dados, orientando o armazenamento e otimizando operações de recuperação ou acesso, tais como consultas, filtros e varreduras.

Neste capítulo, exploraremos as propriedades das principais estruturas de índices, sua evolução e uso em aplicações intensivas de dados, especialmente na gestão de dados da saúde. No curso desta exposição, focaremos em soluções habilitadas por algoritmos de aprendizado de máquina, chamadas de Índices Aprendidos (LI, do inglês *Learned Index*) [Kraska et al., 2018]. Este novo conceito, apresentado em 2018 à comunidade de gestão de dados, é percursor de uma nova classe de sistemas de dados autoajustáveis, denominada *instance optimized data systems* (IODS). No cerne destes sistemas e estruturas de índices estão três premissas: i) algoritmos de gestão de dados são modelos; ii) é possível superar os algoritmos tradicionais a partir da obtenção de conhecimento, ou estimativa, da distribuição de uma coleção de dados; e iii) os inúmeros parâmetros para otimização da gestão de dados representa um volume de trabalho cada vez mais desafiador para os DBA engenheiros de dados. Sendo assim, o objetivo geral deste texto é apresentar o estado-da-arte de LI e suas aplicações na gestão de dados de saúde. Acreditamos que a participação da comunidade de Computação Aplicada à Saúde neste novo tópico de pesquisa tem o potencial de levar ao

[4] https://top500.org/lists/top500/

desenvolvimento de diversas soluções e aplicações com impacto direto na prática clínica, da vigilância em saúde, gestão e formulação de políticas públicas.

Este texto está dividido em três principais seções. Ainda na Introdução (Seção 4.1), abordaremos o papel da indexação nas atividades de governança e gestão de dados em saúde, focando em experiências documentadas de entidades governamentais e centros de dados nacionais. Na Seção 4.2, aprofundaremos a exposição do conceito de LI, seus avanços e limitações. Por fim, na Seção 4.4, apresentaremos uma discussão sobre tendências, desafios de pesquisa e aplicações para a saúde que podem ser endereçadas por métodos inovadores de indexação.

4.1.1 Indexação na gestão de dados em saúde

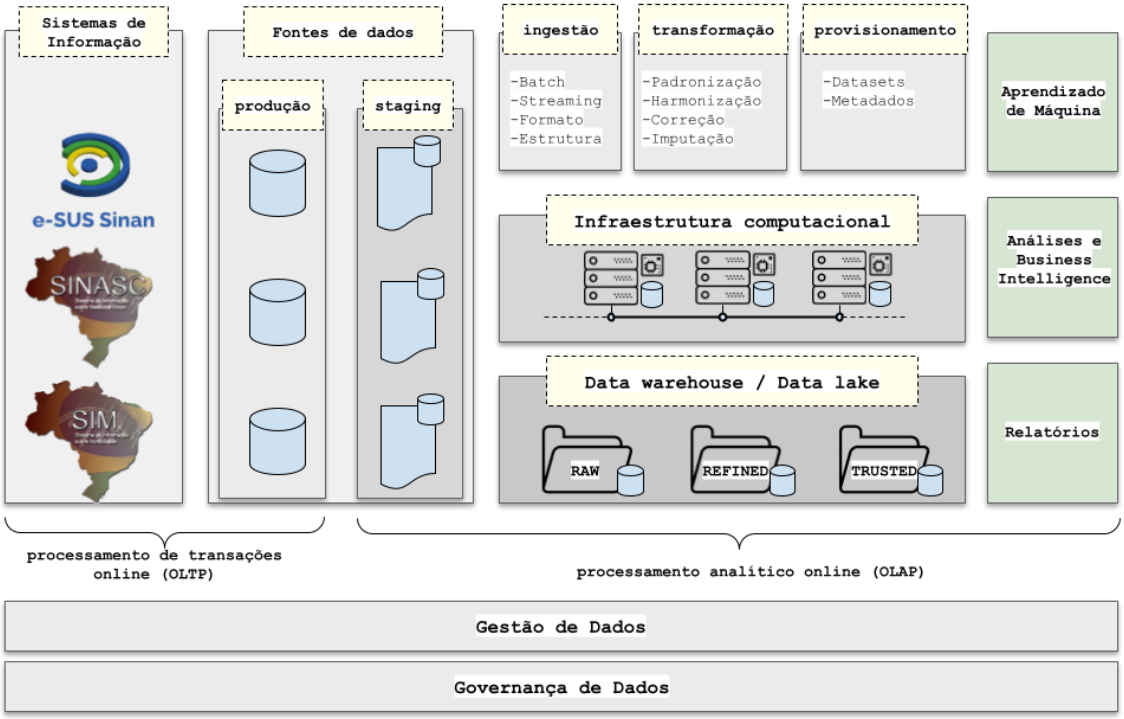


Figura 4.3. Modelo de arquitetura de dados e seus principais componentes, adaptado de [Reis et al, 2022;].

O termo governança de dados agrega diversas diretrizes e técnicas que devem ser adotadas por uma organização para garantir a qualidade, integridade, usabilidade e segurança dos dados que são produzidos ou coletados [Eryurek et al., 2021]. Estas diretrizes, quando bem aplicadas e suportadas pela cultura organizacional, tem o potencial de estabelecer processos dirigidos a dados que maximizam o valor das decisões ao passo em que estabelece uma plataforma de segurança adequada para a preservação da privacidade. Derivada da governança, a disciplina de gestão de dados consiste em práticas e conceitos aplicados no desenvolvimento, execução e supervisão do manejo da informação através do seu ciclo de vida [Dama, 2017]. Administradores de bancos de dados e engenheiros de dados são os principais responsáveis por desenhar,

planejar e materializar em práticas cotidianas todas as determinações de um modelo de gestão.

A arquitetura demonstrada da Figura 4.3 ilustra, conceitualmente, como se dá a gestão dos dados dos sistemas de informação da saúde, dirigida pelo DATASUS. As aplicações em produção demandam tecnologias e SGBD capazes de suportar *workloads* dinâmicos, onde as observações podem ser produzidas, coletadas, integradas, alteradas e incluídas. Este processamento de transações online (OLTP, do inglês *online transaction processing*) obedece a restrições muito bem definidas pela modelagem do sistema de informação e deve suportar uma alta taxa de transações com características de atomicidade, consistência, isolamento e durabilidade. Por outro lado, fora da produção, o DATASUS fornece acesso ao *backup* dos bancos de dados destes sistemas de informação em seus canais online, ou área de *staging*. Como uma demanda das diretrizes de governança que determinam a transparência e preservação da privacidade, estabelecidas pelas leis de acesso à informação e de proteção dos dados pessoais, estes dados disponibilizados são pseudonimizados, processo que extrai todas as variáveis com potencial de identificação direta dos indivíduos. A partir deste ponto, o DATASUS, gestores, pesquisadores, técnicos e organizações podem estabelecer ambientes auxiliares para efetuar suas análises. Neste ambiente são executadas as tarefas de processamento analítico online (OLAP, do inglês *online analytical processing*).

O modelo de gestão OLAP ilustrado na Figura 4.3 compreende as principais tarefas da engenharia de dados. Tarefas analíticas mais complexas, *data-intensive*, se beneficiam de arranjos computacionais dispostos em cluster ou computação em nuvem para garantir o armazenamento e processamento massivo de grandes volumes de dados. Contudo, rotinas extração, transformação e carga (ETL) antecedem estas análises, produzindo *datasets* e metadados que servirão de entrada para modelos e *dashboards*. Cabe ao modelo de gestão de dados estabelecer quais são as melhores práticas para o manejo de dados neste ambiente. Por exemplo, um engenheiro de dados deverá estabelecer políticas de *backup* ou de nomenclatura para documentos, pastas, bases e zonas no *data lake*. Estratégias de indexação são elementos centrais para garantir a eficiência, escalabilidade e confiabilidade de todas as etapas presentes na Figura 4.3. Engenheiros de dados e DBAs são responsáveis por aplicar os métodos adequados para cada tipo de dado, volume de operações, tecnologia ou recurso computacional disponível.

Como descrito no sítio eletrônico do principal mantenedor das bases de dados de saúde do país, o Departamento de Informática do Sistema Único de Saúde (DATASUS), as diretrizes de gestão dos bancos de dados que suportam seus sistemas de informação são estabelecidas pela Metodologia de Administração de Dados (MAD), que normatiza regras gerais de uso e otimização. As especificações encontradas no MAD conformam os principais aspectos do modelo de dados, incluindo quais os SGBDs, definições de padrões, regras técnicas, normas de modelagem e dicionários de dados [DATASUS, 2024]. Fora deste ambiente de produção, as tarefas de Extração, Transformação e Carga (ETL) e a construção de paineis analíticos são suportados pelas ferramentas PowerCenter, PowerBI e Qlik Sense.

Na saúde, a estratégia de gestão de dados, juntamente com a governança, desempenha papel crucial ante a diversidade dos sistemas de informação e seus modelos de negócio. Sistemas que quando combinados permitem o mapeamento da linha da vida, levantamento dos aspectos clínicos e farmacológicos de uma população, estudo sobre a estrutura genética e o percurso social e hospitalar de um indivíduo. Cada tema e finalidade de construção do Sistema de Informação para a saúde repercute na forma como a informação será estruturada. Há um aspecto ex ante que antecede a construção do sistema, e que repercute nele; e que deve ser considerado na estratégia ampliada de gestão, e na proposta metodológica de administração de dados. Essas considerações possibilitam montar uma organização informacional a partir de estruturação de domínios e combinações capazes de relacionar os dados tanto fisicamente, quanto logicamente.

A estratégia e métodos de análise e organização de dados para saúde, devem considerar um modelo de negócio que enfatiza tanto o ciclo de vida dos dados, quanto a noção de cauda longa do dado. A primeira estabelece as considerações sobre a evolução de acesso e compartilhamento dos dados, enquanto a segunda enfatiza a frequência de uso e consumo dos dados. Neste sentido, a arquitetura de governança e, por conseguinte, a de gestão deve considerar (a) um pipeline de dados que se retroalimenta e interage; (b) os requisitos de "*informatics*", segundo os termos de Gray, ou da ecologia computacional e informacional necessárias para o contexto e o campo intelectual do negócio [Gray et al., 2007; Hey, Tansley, Tolle, 2009] e (c) uma arquitetura de dados orientada a serviços que deve ser delineada com vistas às operações repetitivas, objetivando a troca e o reaproveitamento de aplicativos, recursos e fluxos de trabalho. O ecossistema debatido deve projetar uma administração de banco de dados que considere a gestão da informação, o modelo computacional, a orquestração do fluxo de dados e a estratégia de documentação, catalogação e publicação de metadados.

Considerando que a gestão de dados aborda a funcionalidade necessária para administração eficiente e eficaz dos dados e, além da coleta e manipulação de dados, inclui ações de controle e avaliação de qualidade, gestão de metadados, gestão de catálogo de dados; a indexação na gestão de dados em saúde atrela-se à organização e à categorização desses modelos informacionais e dos metadados. Uma indexação eficaz apoia a recuperação de dados, suporta iniciativas de pesquisa, facilita o fluxo de trabalho e a eficiência operacional, apoia a integração de dados, e contribui para manutenção da qualidade e integridade de dados.

As considerações sobre o impacto da indexação na gestão de dados em saúde ampliam-se quando consideramos as ações para a Saúde Digital. Iniciativa que estimula o debate em torno de confederações e formação de redes segundo uma perspectiva de compartilhamento de dados de saúde que envolve atores distintos.

Iniciativas na pesquisa, como o Centro de Integração de Dados e Conhecimentos para Saúde (CIDACS/Fiocruz), utilizam estas diversas bases de dados de saúde para compreender o impacto das iniquidades em saúde e avaliar políticas de proteção social na saúde populacional [Barreto et al., 2019]. As investigações conduzidas no CIDACS são suportadas por diversas etapas de computação intensiva de dados e conta com tecnologia de ponta de *big data*, computação de alto desempenho e segurança da

informação para apoiar a resposta das diversas questões levantadas pelos projetos do Centro.

A relevância das técnicas de indexação para garantir a gestão e manejo dos *datasets* em seu *data lake* podem ser ilustradas em três exemplos. Em primeiro lugar, sua principal ferramenta de integração de dados, o CIDACS-RL [Barbosa et al., 2020], tem suas propriedades de escalabilidade e desempenho baseadas numa solução eficiente de busca em bases indexadas para definir quais registros correspondem a mesma pessoa em bases de dados distintas. No segundo exemplo, gestão de grandes coleções de dados integrados, incluindo a Coorte de 100 Milhões de Brasileiros, a Coorte de Nascimentos [Paixão et al., 2021] e a Plataforma de Dados da Covid-19 [Junior et al., 2021] é apoiada pelo Dataverse[5], que utiliza indexação para garantir encontrabilidade e visibilidade dos datasets. Um terceiro exemplo consiste no aplicativo de visualização da Coorte de 100 Milhões de Brasileiros[6] que é capaz de sintetizar em gráficos uma quantidade considerável de dados sob demanda.

4.1.2. Um overview sobre indexação, sua evolução e uso prático em tarefas intensivas de dados

Índices são estruturas capazes de organizar os dados, orientando o armazenamento e otimizando operações de recuperação ou acesso, tais como consultas, filtros e varreduras [Kaur, 2018]. Um dos modelos mais populares, a Árvore-B, pode permitir consultas de intervalo, baseadas em ordenação e multidimensionais [Graefe et. al., 2010]. Este modelo consiste em extrair um conjunto de amostras para uma série de valores temporalmente decrescentes geometricamente. Por mais que a Árvore-B tenha a capacidade de responder a consultas top-k(t) em tempo linear e com um custo de E/S esperado, ao lidar com fluxos de dados online que apresentam comportamentos desconhecidos a utilização da Árvore-B se torna inviável. Seguindo com índices baseados em árvore, o modelo KR+ introduziu um conceito de divisão e mesclagem de nós fazendo com que o acesso a múltiplos atributos fossem melhorados e com uma capacidade de recuperação de dados espaciais deturpados. Nesse modelo já é considerado tratativas para dados em nuvem e fornece suporte às em buscas de intervalo e de vizinhos próximos [Gani et al., 2014]. Já os índices de bitmap são capazes de acelerar operações em data warehousing, processamentos de análises online e diversas tarefas de gerenciamento de dados, mas não conseguem responder às consultas em um tempo viável [Wu et al., 2010]. Na indexação de hash, a busca de similaridade aproximada para dados de alta dimensão utilizando codificação de binários compactados para obtenção de resultados mais rápidos é o principal foco [Wu et al., 2010].

Os índices estão disponíveis nativamente em diversos sistemas gerenciadores de bases de dados (SGBDs). Uma pessoa responsável por administrar SGBDs consegue incluir facilmente os algoritmos tradicionais no seu projeto de otimização de processamento de consultas, definindo uma ou mais colunas de cada relação. Estruturas baseadas em intervalos, como árvores-B, são capazes de produzir uma organização de dados auto-balanceada que ordena o dado em páginas dispostas hierarquicamente

[5] https://dataverse.cidacs.org/
[6] https://appcoorte.cidacs.org/home

[Taranpreet et al., 2018]. Desta forma, ao buscar uma chave, o índice retornará em tempo logarítmico a página onde aquele valor se encontra, permitindo que aplicações desenvolvam suas próprias estratégias para buscar o valor de interesse naquele intervalo. Soluções posicionais, como os índices baseados em hash, deterministicamente mapeiam pontos de dados num *array* finito. A eficiência desta alternativa está associada à sua capacidade de evitar colisões que promovam o desbalanceamento, incluindo muitos registros num *array* com espaço limitado. Estruturas baseadas em filtros de Bloom são capazes de informar se uma determinada chave de busca está presente num conjunto de dados, esta característica permite evitar buscas irrelevantes e com resultados livres de falsos negativos [Kraska et. al, 2018].

Consultas do mundo real envolvem frequentemente a busca por múltiplos atributos usando uma chave de busca composta em um SGBD. Esse contexto pode exigir múltiplos índices ou um índice construído a partir de uma chave de busca multiatributo, um índice multivariado ou multidimensional (MI) [Silberschatz et al., 2020]. Geralmente, é possível produzir um MI a partir de uma chave de busca composta recorrendo à ordenação lexicográfica, *hashing* particionado ou arquivos de grade. Índices ordenados podem depender da ordenação lexicográfica e oferecer suporte a consultas de intervalo em atributos de dados. O *hashing* particionado pode retornar prontamente a interseção de *buckets* de cada valor de chave de busca codificado envolvido na consulta. No entanto, essas estratégias anteriores podem falhar ou levar a operações de E/S desnecessárias em consultas de condição de comparação. Os arquivos de grade fornecem particionamento uniforme em todas as dimensões de dados, abordando as deficiências das soluções ordenadas e baseadas em hash. Devido às operações intensivas de OLTP de dados dinâmicos, a principal limitação dos arquivos de grade relaciona-se ao *overhead* de manter a distribuição uniforme de instâncias em células e *buckets* [Silberschatz et al., 2020]. Casos especiais de MI se esforçam para oferecer suporte a consultas em dados espaciais ou temporais. A R-tree estende a B+-Tree para indexar recursivamente caixas delimitadoras ou tuplas bidimensionais que representam polígonos, pontos ou séries temporais. Kraska et al. (2021) argumenta que a adoção de algoritmos em SGBD frequentemente requer eficiência, maturidade e capacidade de lidar com diversas situações. Ainda assim, os desafios do gerenciamento de big data impõem uma revisão do modelo relacional e de suas soluções de indexação tradicionais embutidas.

Avanços em alternativas de SGBD para endereçar os requisitos do big data produziram modelos de dados de documentos e vetores. Sistemas de gerenciamento de banco de dados vetoriais (VDBMS) [Kukreja et al., 2023] e sistemas de gerenciamento de banco de dados orientados a documentos (DODBMS) [Banker et al., 2016] são responsáveis por fornecer armazenamento eficiente, processamento de consultas, transações, privacidade e escalabilidade para um grande volume de registros e atributos que não possuem um esquema fixo. Com relação à indexação, DODBMS se beneficia das estruturas de *B+ -Tree* para permitir a recuperação rápida de informações [Banker et al., 2016], processamento de consultas de similaridade e intervalo. VDBMS, por outro lado, frequentemente suporta consultas em um conjunto mais amplo de modalidades de dados, armazenados após codificação em vetores de características n-dimensionais por meio de modelos de *embedding*. As estruturas de índice em

VDBMS são categorizadas como baseadas em tabelas [Datar et al., 2004], baseadas em árvores [Dasgupta et al., 2008] ou baseadas em gráficos [Harwood et al., 2016]. Essas soluções inovadoras enfrentam o custo da comparação de distâncias e a ausência de uma estrutura explícita para particionar vetores, permitindo buscas de similaridade e de vizinho mais próximo. Fora desses sistemas, os pesquisadores no campo de gerenciamento de dados têm discutido diversos índices adequados ao *big data*.

A mudança de técnicas de indexação para *big data* é dupla: extensões para algoritmos tradicionais e técnicas aprimoradas de aprendizado de máquina [Gani et. al, 2014]. As extensões de ponta são projetadas para produzir estruturas de MI sofisticadas para aplicações específicas, frequentemente focando em grandes volumes de dados não-estruturados e multimodais. Por exemplo, o índice T-PARINET combina particionamento de gráficos e a árvore *B+ -Tree* para permitir busca e recuperação rápida em consultas espaço-temporais [Sandu et al. 2014]. *KR+ -Tree* é implementado no Cassandra, um sistema de banco de dados distribuído de código aberto, para permitir acesso a dados espaciais através de consultas de faixa e vizinhança mais próxima em um ambiente de computação em nuvem [Wei et al. 2014]. Por outro lado, estruturas construídas a partir de computação suave são principalmente derivadas de métodos baseados em gráficos, usando decisões binárias aprendidas a partir de dados rotulados para navegar e escanear o índice [Gani et al., 2014]. Em [LAZARIDIS et al., 2013] um mecanismo de recuperação de informações, consultas de similaridade em bancos de dados multimídia, incluindo objetos 3D, imagens, vídeo e áudio, são permitidas. Sua metodologia abrange o uso de técnicas de aprendizado de variedades para extrair descritores de itens de mídia, construir uma estrutura de índice que vincula cada entrada de dados transformada ao objeto original respectivo e usar anotação de propagação para atribuir automaticamente conteúdo amigável ao humano aos registros. Apesar dos resultados de desempenho promissores, principalmente apoiados pela extensão do trabalho de [Amato et al., 2013], os autores apontam para a falta de suítes de *benchmark* e *baselines* para fundamentar a vantagem em relação às alternativas tradicionais e escaláveis.

Para fins de exemplo, considere um cenário hipotético em que um conjunto de gestores da saúde desejam construir um modelo preditivo capaz de antecipar o desfecho de um caso suspeito. Inicia-se então um *pipeline* OLAP, ilustrado na Figura 4.3. Neste contexto, uma pessoa DBA ou Engenheira de Dados deve: i) alinhar com especialistas quais são as bases de dados, atributos e filtros necessários para suportar este modelo; ii) executar tarefas de aquisição dos dados nos portais e serviços do DATASUS; iii) armazenar estas bases no *data lake* de seu domínio; escrever e executar rotinas de pré-processamento dos dados (padronização, harmonização e correção); e iv) tornar disponível em uma zona do *data lake* o *dataset* pronto para treinar e avaliar o modelo. Após algumas rodadas de discussão, decidiu-se então que as variáveis independentes devem incluir a temperatura e a saturação sanguínea obtidos na triagem, a quantidade de sintomas gripais e aspectos de temporalidade do caso. Os intervalos de tempo entre a admissão e os eventos de coleta, resultado do exame e encerramento do caso devem ser considerados. Os códigos SQL a seguir foram usados para produzir a nova variável que calcula a quantidade de sintomas gripais e o conjunto de dados — amostra ilustrada na Tabela 4.1 — que será usado para treinar o modelo, doravante denominado *Dataset 1*:


```
-- código SQL para adicionar uma nova coluna “qt_sintomas” à
-- tabela tbl_info_epidemiologicas contendo a contagem de
valores “SIM”
-- nas colunas "s_dispneia", "s_disturbios_gustat",
"s_disturbios_olfat"
-- e "s_gripais".

ALTER TABLE tbl_info_epidemiologicas
ADD COLUMN qt_sint INT;

UPDATE tbl_info_epidemiologicas
SET qt_sintomas =
    CASE WHEN s_dispneia = 'SIM' THEN 1 ELSE 0 END +
    CASE WHEN s_disturbios_gustat = 'SIM' THEN 1 ELSE 0 END +
    CASE WHEN s_disturbios_olfat = 'SIM' THEN 1 ELSE 0 END +
    CASE WHEN s_gripais = 'SIM' THEN 1 ELSE 0 END;
```

```
-- código de consulta SQL que recupera, para todos os casos em
2022,
-- 2023 e 2024:
--   - temperatura no momento da triagem (t_temp);
--   - saturação de oxigênio no sangue no momento da triagem
(t_satur);
--   - quantidade de sintomas gripais (qt_sint);
--   - a diferença entre as datas dt_admis e t_dt_coleta
(tmp_coleta);
--   - a diferença entre dt_admis e tp_dt_resultado
(tmp_result1);
--   - a diferença entre t_dt_coleta e tp_dt_resultado
(tmp_result1);
--   - a diferença entre dt_admis e desencerra (tmp_total)
--   - a evolução final do caso (evolucao)

CREATE TABLE dataset1 AS
SELECT
    i.id_caso,
    i.t_temp,
    i.t_satur,
    e.qt_sint,
    DATEDIFF(a.dt_admis, c.t_dt_coleta) AS tmp_coleta,
    DATEDIFF(a.dt_admis, c.tp_dt_resultado) AS tmp_result1,
    DATEDIFF(c.t_dt_coleta, c.tp_dt_resultado) AS tmp_result2,
    DATEDIFF(a.dt_admis, a.dt_encerra) AS tmp_total,
    a.evolucao
FROM
    tbl_identificacao i
JOIN
    tbl_info_epidemiologicas e ON i.id_caso = e.id_caso
JOIN
    tbl_adm_encerramento a ON i.id_caso = a.id_caso
JOIN
```



```
    tbl_info_clinicas c ON i.id_caso = c.id_caso
WHERE
    YEAR(i.dt_nasc) IN (2022, 2023, 2024) AND
    c.res_teste = 'POSITIVO';
```

Uma vez disponível na zona "REFINED" do *data lake*, o *Dataset 1* ilustrado na Tabela 4.1 pode ser submetido para uma análise de grupos por analistas ou cientistas. Esta estratégia objetiva segmentar os casos num número arbitrário de grupos, revelando relações escondidas entre eles [Kaufman et al., 2009]. O AGNES (*Agglomerative Nesting*) é um dos modelos de aprendizado não-supervisionado tradicionalmente usados para agrupamento de dados. Em resumo, o modelo estabelece numa primeira iteração que todos os objetos de dados são *singletons*, ou seja, grupos formados por apenas um registro. As iterações seguintes efetuam fusões sucessivas dos grupos de registros mais similares até que haja um único grupo. Cada iteração fornece uma quantidade decrescente de grupos, dispostos hierarquicamente, e cientista de dados é responsável por fazer inspeções que subsidiem a decisão do melhor número de grupos para um determinado conjunto de dados [Kaufman et al., 2009; Witten et. al, 2002].

Tabela 4.1. Dataset 1 resultado da consulta SQL para recuperar o perfil dos casos com resultado positivo do teste laboratorial no banco de dados da Figura 4.2.

id_caso	t_temp	t_satur	qt_sint	tmp_coleta	tmp_result1	tmp_result2	tmp_total	evolucao
62483	40,11	87	3	0	0	0	5	OBITO
69157	35,54	93	3	1	2	2	0	CURA
49407	35,16	93	4	1	0	0	2	CURA
578812	42,1	92	4	1	0	0	2	CURA
86729	38,39	99	2	0	0	0	5	CURA
4738	40,24	86	4	0	0	0	2	OBITO
4470	38,79	96	2	0	1	0	3	CURA
17781	39,33	98	4	0	0	0	2	OBITO
68312	37,12	93	2	0	0	0	3	OBITO
31248	37,52	95	2	0	0	0	2	CURA

Uma implementação do AGNES pode incluir uma estrutura de dados que auxilia a decisão sobre qual par de grupos deve ser mesclado a cada iteração. Esta estrutura mimetiza uma matriz de distâncias que consiste na combinação de todos os pares únicos

de registros, sua respectiva distancia euclideana. Uma coluna adicional *membership* do `id_caso_a`. A primeira estrutura é a matriz de distancias, de cada par de registro. Considerando uma combinação $C(n, 2)$, sendo o número de registros no Dataset 1 $n = 10$, então a quantidade de pares desta estrutura de dados, expressa na Tabela 4.2, será de $\frac{n \times (n-1)}{2} = \frac{10 \times (10-1)}{2} = 45$ registros.

```
-- Calcular a matriz de distâncias

CREATE TABLE matrix_distancias AS
SELECT
    a.id_caso AS id_caso_a,
    b.id_caso AS id_caso_b,
    a.id_caso AS membership,
    SQRT(
        POW(a.t_temp - b.t_temp, 2) +
        POW(a.t_satur - b.t_satur, 2) +
        POW(a.qt_sint - b.qt_sint, 2) +
        POW(a.tmp_coleta - b.tmp_coleta, 2) +
        POW(a.tmp_result1 - b.tmp_result1, 2) +
        POW(a.tmp_result2 - b.tmp_result2, 2) +
        POW(a.tmp_total - b.tmp_total, 2)
    ) AS distancia_euclideana,
    MOD(ROW_NUMBER() OVER (ORDER BY a.id_caso), 10) AS
membership
FROM
    dataset1 a
JOIN
    dataset1 b
ON
    a.id_caso < b.id_caso
ORDER BY
    a.id_caso, b.id_caso;
```

A Tabela 4.2 mostra o resultado do código SQL acima. Esta estrutura já traz o resultado da primeira iteração do AGNES, em que se atribui para cada registro, identificado pelo `id_caso_a`, um número de grupo contento apenas ele.

Tabela 4.2. Resultado da consulta SQL para produzir uma estrutura de dados com as distâncias de todos os pares únicos do Dataset 1 (Figura 4.1), sua distancia e cluster.

id_caso_a	id_caso_b	dist_euclideana	membership
62483	69157	9,40	0

62483	49407	9,16	0
62483	578812	6,98	0
62483	86729	12,08	0
62483	4738	3,18	0
62483	4470	9,62	0
62483	17781	11,61	0
62483	68312	7,47	0
62483	31248	9,11	0
69157	49407	2,83	1
69157	578812	7,56	1
69157	86729	8,54	1
69157	4738	10,91	1
69157	4470	7,21	1
69157	17781	8,77	1
69157	68312	4,03	1
69157	31248	3,74	1
49407	578812	7,55	2
49407	86729	10,87	2
49407	4738	5,72	2
49407	4470	7,72	2
49407	17781	10,63	2
49407	68312	5,20	2
49407	31248	7,17	2
578812	86729	13,61	3
578812	4738	6,27	3
578812	4470	9,68	3
578812	17781	12,40	3
578812	68312	8,11	3
578812	31248	10,30	3
86729	4738	16,08	4
4470	86729	3,00	5

17781	86729	2,65	6
68312	86729	7,21	7
31248	86729	7,34	8
4738	4470	9,11	9

Nas iterações subsequentes, utilizando o parâmetro de *single linkage*, os grupos mesclados serão aqueles que possuem registros de menor distância. O seguinte código SQL será, portanto, executado exaustivamente até que só reste um cluster.

```
-- encontrar os grupos a ser mesclados: registros com
menor distancia -- entre si que se encontram em grupos
diferentes (single linkage).

SELECT MIN(dm1.dist_euclideana)

FROM
    distance_matrix dm1
INNER JOIN
    distance_matrix dm2
ON
    dm1.id_caso_b = dm2.id_caso_a
AND
    dm2.membership <> dm1.membership
```

A consulta SQL acima é central para execução completa do algoritmo AGNES. Portanto, cabe entender quais as possibilidades de otimização do seu processamento. Diferentes SGBDs oferecem ferramentas que permitem DBAs e Engenheiros de Dados analisar o plano de execução de uma *query* complexa. Suponha que o grafo de processamento da Figura 4.4 expressa o plano de execução real de um SGBD.

As etapas realçadas em vermelho na Figura 4.4 concentram os trechos mais complexos do grafo de processamento. Em casos com um número maior de registros, varrer todas as tuplas da tabela `distance_matrix` para efetivar o filtro que suporta a função `INNER JOIN` é proibitivo, principalmente num contexto em que esta mesma query será executada inúmeras vezes.

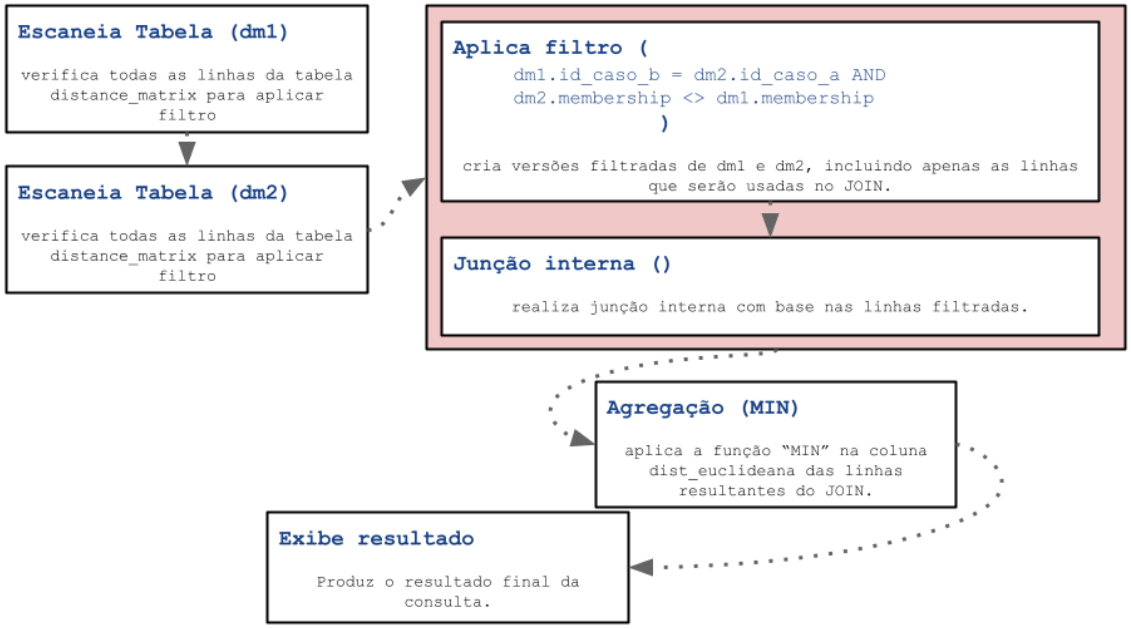


Figura 4.4. Grafo de processamento de query SQL que busca na tabela distance_matrix pelo par de registros com menor distância entre sí até então atribuídos a grupos diferentes.

```
-- Definindo índice uma estrutura de índice a partir do
atributo "id_caso_b" para auxiliar o processamento da
consulta SQL expressa na Figura 4.4.

CREATE INDEX idx_dm1_id_caso_b ON distance_matrix
(id_caso_b) USING BTREE;

-- Definindo índice uma estrutura de índice composta a
partir dos atributos "id_caso_a" e "membership" para
auxiliar o processamento da consulta SQL expressa na
Figura 4.4.

CREATE INDEX idx_dm2_id_caso_a_membership ON
distance_matrix (id_caso_a, membership) USING BTREE;
```

Após a definição do índice B+ -Tree — padrão para índices B-Tree no PostgreSQL e MySQL — nos atributos de interesse, o processamento ilustrado na Figura 4.4 pode ser sensivelmente otimizado [Silberschatz et al., 2020]. O principal ganho é a possibilidade de escanear toda a tabela distance_matrix duas vezes para efetivar o filtro. A estrutura de índice B+ -Tree dispõe os valores de uma ou mais colunas numa árvore balanceada, garantindo que a distância, expressa em saltos, entre o nó raiz e o qualquer nó folha seja a mesma. Cada nó deste índice multinível possui três componentes: i) um ponteiro que indica chaves, nó, ou nível com valores que antecedem

a chave corrente; ii) o valor da chave corrente; e iii) um ponteiro que indica chaves, nó, ou nível com valores que sucedem à chave corrente. O Algoritmo 1 ilustra um pseudo-algoritmo adaptado de [Silberschatz et al., 2020] para construção de uma estrutura B+ -Tree.

Algoritmo 1: pseudo-código para construção de estrutura B+ - Tree.

```
funcao insere(valor K, ponteiro p):
    se arvore == vazio:
        cria um nó folha L que também é o raíz
    senão:
        encontra nó folha L que deverá conter o valor K

    se L possui menos de n-1 chaves:
        insere_na_folha(L, K, P)

    ou se L já possui n-1 valores:
        cria-se o nó L'
        Copia L.P1 ... L.Kn-1 para um bloco T que pode segurar
n pares de chave-valor

        insere_na_folha(T,K,P)
        L'.Pn <- L.Pn
        deleta L.P1 até L.Kn-1 de L

        Copia T.P1 até T.K(n/2) de T em L iniciando em L.P1
        Copia T.P⌈n/2⌉+1 até T.Kn de T em L' iniciando em L'.P1
        O menor valor de K' fica em L'
        insere_no_pai(L,K',L')


funcao insere_no_folha(nó L, valor K, ponteiro P):
    se(K < L.K1):
        insere P, K em L após L.P1
    senão:
        Ki se torna o maior valor em L que é menor ou igual a
K
        Insere P,K em L após L.Ki

funcao insere_no_pai(nó N, Valor K', nó N'):
    se(N é o nó raiz):
        cria-se um novo nó R contendo N, K', N' #N e N' são
ponteiros
        R se torna a raiz da árvore
        return
```

```
    P <- pai(N)

    se (P possui menos de n ponteiros):
        insere(K', N') em P após N

    senão:
        Copia P para um bloco de memória T que pode receber P
e (K', N')
        Insere (K', N') em T após N
        Apaga todas as entradas de P
        Cria um nó P'
        Copia T.P' ... T.P⌜(n+1)/2⌝ em P
        K'' <- T.K ⌜(n+1)/2⌝
        Copia T.P⌜(n+1)/2⌝+1 ... T.Pn+1 em P'
        insere_no_pai(P, K'', P')
```

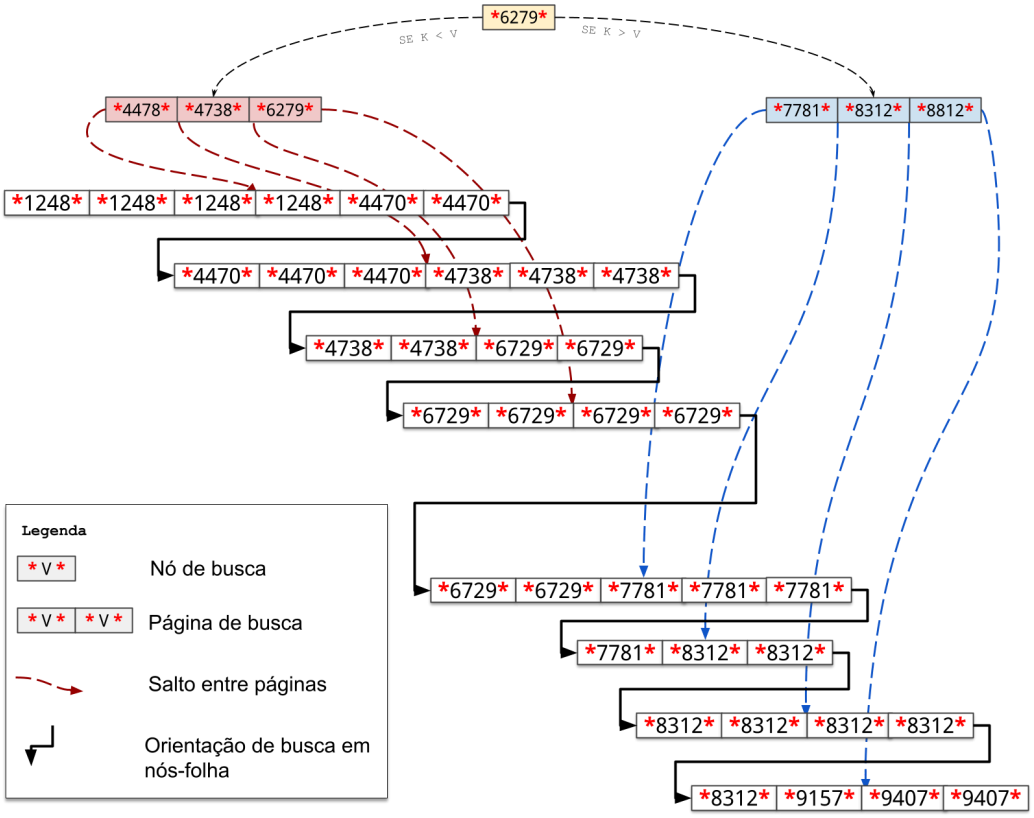


Figura 4.5. Estrutura de índice B+ -Tree produzida a partir do atributo id_caso_b.

Um DBA ou Engenheiro de Dados é responsável por efetuar tarefas de *tuning* que inclui, entre outros parâmetros, avaliar quais atributos de cada tabela podem ser indexados para otimizar a maioria das consultas (*workload*) de uma determinada equação. Ao utilizar o pseudo-código do Algoritmo 1 na coluna `id_caso_b`, como sugere o código SQL acima, indicando um valor de $K = 7$, referente ao grau da

árvore a ser construída, obtemos uma estrutura de índice expressa na Figura 4.5. Note que, considerando o grafo de processamento da Figura 4.4, a estrutura de índice construída limitará o escaneamento de valores em id_caso_b e id_caso_a a um espaço de busca muito menor, tornando esta tarefa intensiva de dados ainda mais eficiente.

Estruturas de índices produzidas a partir de algoritmos tradicionais como árvores-B, hash e filtros de Bloom são capazes de organizar os dados de maneira a facilitar sua recuperação posterior, auxiliando o processamento de queries aos registros de uma determinada base de dados, provendo a eficiência necessária para diversas tarefas em casos de alta-dimensionalidade. Os índices estão disponíveis nativamente em diversos sistemas gerenciadores de bases de dados (SGBDs). Uma pessoa responsável por administrar SGBDs consegue facilmente incluir os algoritmos tradicionais no seu projeto de otimização de processamento de consultas, definindo uma ou mais colunas de cada relação. Estruturas baseadas em intervalos, como árvores-B, são capazes de produzir uma organização de dados auto-balanceada que ordena o dado em páginas dispostas hierarquicamente [Taranpreet et. al., 2018]. Desta forma, ao buscar uma chave, o índice retornará em tempo logarítmico a página onde aquele valor se encontra, permitindo que aplicações desenvolvam suas próprias estratégias para buscar o valor de interesse naquele intervalo. Soluções posicionais, como os índices baseados em hash, deterministicamente mapeiam pontos de dados num array finito. A eficiência desta alternativa está associada à sua capacidade de evitar colisões que promovam o desbalanceamento, incluindo muitos registros num *array* com espaço limitado. Estruturas baseadas em filtros de Bloom são capazes de informar se uma determinada chave de busca está presente num conjunto de dados, esta característica permite evitar uma buscas irrelevantes e com resultados livres de falsos negativos [Kraska et. al, 2018].

4.2. Learned index: usando machine learning para superar modelos tradicionais de indexação

Sistemas de dados de propósito geral demandam o ajuste de uma miríade de hiper-parâmetros, ou *knobs*, no esforço de otimizar seus componentes para atender conjuntos de dados ou *workloads* específicos. A estrutura de índice ilustrada na Figura 4.5 é um exemplo de como pequenas alterações na entrada do algoritmo podem alterar o resultado. Na figura, o DBA decidiu construir uma árvore de grau 7. Isso implica que cada nó da árvore deve conter, no máximo, 6 valores de chaves ordenadas. Uma rápida inspeção nos valores do atributo `id_caso_b` da Tabela 4.2 permite verificar que vários valores se repetem mais que 4 vezes. Esta duplicação implica numa fragmentação dos valores em mais de um nó de segundo nível, aumentando o espaço de busca e tornando o método menos eficiente. Por outro lado, outro algoritmo, como o *Hash* poderia ser aplicado neste caso e alcançar melhores resultados de latência, uso de memória e eficiência de armazenamento. Por décadas, SGBDs de propósito geral proviam versatilidade a partir destes domínios de intervenção, os algoritmos e os parâmetros de ajuste disponíveis para a adequação em certos cenários.

O uso de inteligência artificial em sistemas de dados não é uma prática recente. Diversos SGBDs ou soluções *ad hoc* já incorporam componentes inteligentes [Li et al. 2021]. As soluções inteligentes abordam, principalmente, aspectos de configuração e otimização. Consultores automáticos de indexação, *tuning* e *views* são exemplos de

componentes habilitados por aprendizado de máquina para configuração. Por outro lado, otimizadores inteligentes focam em obter um maior desempenho no processamento de *queries*, oferecendo estimativas de cardinalidade e ordenação para suportar operações de junção [Li et al. 2021].

Uma terceira vertente do uso de inteligência artificial em sistemas de dados foca no redesenho completo de seus componentes, tornando-os autoajustáveis para qualquer distribuição de dados ou padrão de carga de trabalho [Kraska, 2021; Ding et al., 2022]. Nas seções seguintes, apresentaremos com detalhes o Learned Index e seus principais avanços desde sua concepção em 2018. Contudo, Por mais que existiram alguns esforços para implementar os componentes otimizados por instância, como o Bourbon que substitui índices de blocos em uma árvore LSM por Learned Index e o Google que integrou índices aprendidos no *BigTable*, grande parte dos componentes otimizados foram criados e avaliados isoladamente [Ding et al., 2022].

O conceito de utilizar modelos de aprendizado de máquina para substituir algoritmos também foi explorado em diversos outros componentes de sistemas de dados. Soluções como otimizadores de consultas [Marcus et al., 2019; Mikhaylov et al., 2022], uso de aprendizado por reforço para escalonadores de tarefas [Mao et al., 2016], layout de dados [Nathan et al., 2020; Campero Durand et al., 2019] e ajuste adaptativo de parâmetros para tuning do sistema de dados [Zhao et al., 2023; Li et al., 2021]. Alguns desses desenvolvimentos alcançaram maturidade suficiente para serem integrados em serviços relevantes de computação em nuvem. Hussam et al., (2020) relata em seu trabalho a solução de LI capaz de armazenar sua estrutura em disco, em oposição à solução baseada em memória na proposta original. A Amazon incorporou componentes de sistemas de dados habilitados por ML em seus serviços da AWS e Redshift [Fanggohans et al., 2023; Armenatzoglou et al., 2022]. A Azure também incluiu estratégias de ML para efetuar tuning automático no SQL Server da Azure [Fritchey et al., 2022]. Além disso, a exploração desta lacuna na área de gestão de dados serviu de base para uma nova classe de soluções, chamadas de sistemas de dados otimizados para instância (IODS, do inglês instance-optimized data systems), capazes de se ajustar automaticamente a qualquer distribuição de dados ou workload. Vários IODS empregam ML em seus componentes principais para aliviar o trabalho quase proibitivo de administradores e engenheiros de dados, oferecendo sistemas de gerenciamento de dados autônomos [Ding et al., 2022; Li et al. 2019; Ma et al., 2022; Li et al., 2021; Zhou et al., 2021; Ma et al., 2021].

4.2.1. O conceito de Learned Index

Em Kraska et al. (2018) é proposto o conceito de Índice Aprendido (LI, do inglês *Learned Index*), uma estrutura de índices produzida a partir da estimação da distribuição dos dados para otimizar as consultas em banco de dados. O principal argumento para o uso de um método de indexação habilitado por ML é que a estimação adequada de uma distribuição de dados utilizando aprendizado de máquina (ML, do inglês *machine learning*) tem o potencial de superar a eficiência dos métodos tradicionais. Para fins de exemplo, considere um vetor $v_{.j} = \{0, 1, 2, 3, ..., N\}$ contendo N chaves inteiras ordenadas do $j^{-ésimo}$ atributo. Como mencionado na Seção 4.1.2, o uso de um algoritmo

de indexação, como o B+ -Tree, pode reduzir a complexidade da busca de uma determinada chave de $O(n)$ para $O(log\ n)$. O conhecimento sobre esta distribuição, contudo, pode permitir que o valor da própria chave de busca seja utilizado como offset, reduzindo a complexidade para $O(1)$. Kraska et al. (2018) argumenta que este tipo de otimização pode ser alcançado em outras distribuições mais complexas através do uso do LI.

O LI é trazido visando a substituição dos modelos tradicionais conhecidos (árvores, hash, bitmap e etc) para um novo que constroi índices a partir de funções que descrevem como os dados estão distribuídos. Por mais que seja difícil imaginar que modelos de aprendizagem de máquina sejam capazes de garantir a mesma eficiência e assertividade de modelos tradicionais, Kraska et al. (2018) conceitua uma árvore B como um "modelo", o qual tem a capacidade de "prever" a localização de um valor dentro de um conjunto com chaves ordenadas. Em uma base de dados analítica em memória, onde a chave primária está ordenada, a indexação mapeia uma posição dentro do vetor ordenado que contém um valor igual ou menor à da chave de consulta. Com isso, o modelo da Árvore B contém um erro máximo do tamanho da folha. Sendo assim, a ideia do LI é substituir esse modelo por outros, como redes neurais, desde que sejam capazes de fornecer garantias semelhantes ao erro máximo e mínimo.

Uma comparação realizada entre o modelo LI e a Árvore-B é feita em Kraska et al. (2018), onde é exemplificado que o modelo tradicional necessita re-equilibrar a disposição dos dados a cada inserção e, de forma semelhante, um modelo que utiliza de aprendizado de máquina para indexação necessitaria realizar um re-treinamento dos dados para continuar apresentando resultados satisfatórios de busca. Outra semelhança entre os modelos é que ambos realizam um mapeamento que retorna um intervalo onde o possível dado está (estará caso exista), porém com a abordagem de LI o espaço amostral do erro é consideravelmente menor, visto que a disposição dos dados é conhecida e a previsão do índice acaba apresentando um apontamento mais eficaz.

Outro ponto abordado por Kraska et al. (2018) é a busca pontual. Tradicionalmente, mapas de hash são utilizados para esse tipo de busca e usam uma função de hash para mapear as chaves para posições dentro do vetor. Porém esses modelos podem acabar gerando conflitos ao mapear várias chaves para a mesma posição dentro do mapa de hash, visando evitar esse tipo de problema, vários estudos foram desenvolvidos, inclusive soluções onde é utilizado aprendizado de máquina. Porém nenhuma dessas soluções traz consigo a utilização do aprendizado da disposição de dados subjacentes. Com isso em vista foi proposto a utilização da CDF como uma maneira de aprender uma função de hash, nesses casos os dados não precisam ser guardados de forma ordenada. O modelo utilizado para nessa nova versão de busca pontual continua sendo o recursivo e depende de dois fatores importantes para funcionar com um bom desempenho: a representação da CDF deve ser precisa e a arquitetura do mapa de hash deve ser bem pensada, onde características de resolução de conflitos e a quantidade de memória que pode ser alocada.

No que diz respeito às estruturas LI, estudos focados em investigar diferentes abordagens estão disponíveis para aproximar a CDF dos dados subjacentes [Kipf et al. 2020; Ding et al., 2020; Ferragina et al. 2020], a indexação de dados multidimensionais

[Nathan et al., 2020] e de strings [Wang et al., 2020], a escalabilidade fornecida por arquiteturas *multicore* [Tang et al., 2020] e de GPU [Zhong et al., 2022; Liu et al., 2024], e permitir a persistência da CDF aproximada no disco [Hussam Abu-Libdeh et al., 2020]. Nas próximas seções, traremos um overview sobre o conceito de função de distribuição cumulativa, aplicando para os dados sintéticos que apresentamos na Figura 4.1. Em seguida, focaremos nossa exposição nos LI que compõem o estado-da-arte, estendidos a partir da sua primeira publicação.

4.2.2. Estimando a distribuição dos dados

Para introduzir o conceito de função de distribuição acumulada (CDF, do inglês *Cumulative Distribution Function*), denotaremos X como uma variável aleatória, que pode ser contínua, ou seja, assume valores reais em um intervalo, como, por exemplo, a altura, em centímetros, de uma pessoa, ou discreta, onde assume valores finitos ou contáveis, como por exemplo, a idade de uma pessoa em anos completos [Ross, 2014]. Para cada variável aleatória X, seja ela discreta ou contínua, existirá uma função $p(x)$ ou $F(x)$, respectivamente, que irá associar a cada possível valor $x \in X$ uma probabilidade. Tal função será chamada de função de probabilidade ou função densidade de probabilidade, se certas condições forem satisfeitas.

Outro conceito importante associado à variável aleatória X é o da função de distribuição acumulada, comumente denotada por $F(x)$, que representa a probabilidade de que X assuma um valor menor ou igual a x. Formalmente, descrevemos $F(X) = \int_{-\infty}^{x} f(x)dx$, no caso contínuo e $F(X) = \sum_{i=1}^{x} p(x)$, para o caso discreto [Wong et al., 2001]. Como exemplo, consideremos que um dado não viciado é lançado, denotaremos como a variável X, o valor da face do dado, na tabela abaixo são mostrados os possíveis resultados, a probabilidade e a probabilidade acumulada:

Tabela 4.3: Tabela de resultados e probabilidades referente ao lançamento de um dado não viciado.

Resultados possíveis	$p(x)$	$F(x)$
1	1/6	1/6
2	1/6	2/6
3	1/6	3/6
4	1/6	4/6
5	1/6	5/6
6	1/6	6/6 = 1

Dados esse exemplo consideremos a seguintes questões:

1) Qual a probabilidade de sair a face 6?

$R: p(x = 6) = 1/6$

2)Qual a probabilidade do resultado ser no máximo 3?

$R: p(x \leq 3) = 3/6 = F(x = 3)$

3) Qual a probabilidade do resultado ser maior do que 3 ?

$R: p(x > 3) = 3/6 = 1 - F(x = 3)$

Agora, consideremos uma variável aleatória X que tem função densidade de probabilidade dada pela função:

$$f(x) = \{0,\ x < 1\ e\ \frac{2}{x^3},\ x \geq 1$$

Para podermos encontrar a função distribuição cumulativa precisamos dividir em duas partes:

Se x < 1 , temos $\int_{-\infty}^{x} f(t)dt = 0$

Se x ≥ 1, temos $\int_{-\infty}^{x} f(t)dt$

$$= \int_{1}^{x} f(t)dt$$

$$= \int_{1}^{x} \frac{2}{t^3}dt$$

$$= [-\frac{1}{t^2}\ ]_1^x$$

$$= -\frac{1}{x^2} + 1$$

Então:

$F(x) = \{\ 1 - \frac{1}{x^2},\ x \geq 1$ e 0,c.c

A função de distribuição acumulada possui algumas propriedades específicas, como:
i) é uma função não decrescente;

ii) é uma função contínua à direita;

iii) $\lim_{X \to -\infty} F(X) = 0$ e $\lim_{X \to +\infty} F(X) = 1$ [Magalhães, 2004].

Considerando as propriedades demonstradas anteriormente, é possível, desde que essa distribuição seja definida para as variáveis do banco de dados, treinar um determinado LI para que este, ao aprender com a distribuição desses dados, seja capaz

de executar as mesmas tarefas que já são realizadas pelos índices tradicionais de maneira mais eficiente e assertiva [Kraska et al., 2018].

Peguemos do exemplo inicial a variável referente a temperatura do paciente na hora da admissão na unidade hospitalar. Usando essas informações queremos estimar a função de distribuição de probabilidade dessa variável, para posteriormente utilizarmos essa informação para estimar a função de distribuição acumulado que servirá como base para o nosso índice aprendido. Essa implementação será feita no R, como mostrado abaixo:

```
library(ggplot2)

dados = data.frame(tem = c(36.93, 40.11, 35.54, 39.81, 35.16,
42.1, 37.67, 39.23, 38.23, 40.24, 38.79, 39.23, 40.24, 38.79,
36.33, 37.82, 37.12, 37.38, 37.52))

ggplot(dados) +

    aes(tem) +

   geom_histogram(aes(y= ..density..), fill="lightblue", colour
="black") +

          stat_function(fun = dnorm, args = list(mean =
mean(dados$tem), sd = sd(dados$tem)))

fn = ecdf(dados$tem)

plot(fn, main =" Distribuição Acumulada Temperatura" , xlab
= "Tempereratura", ylab="F(x)")
```

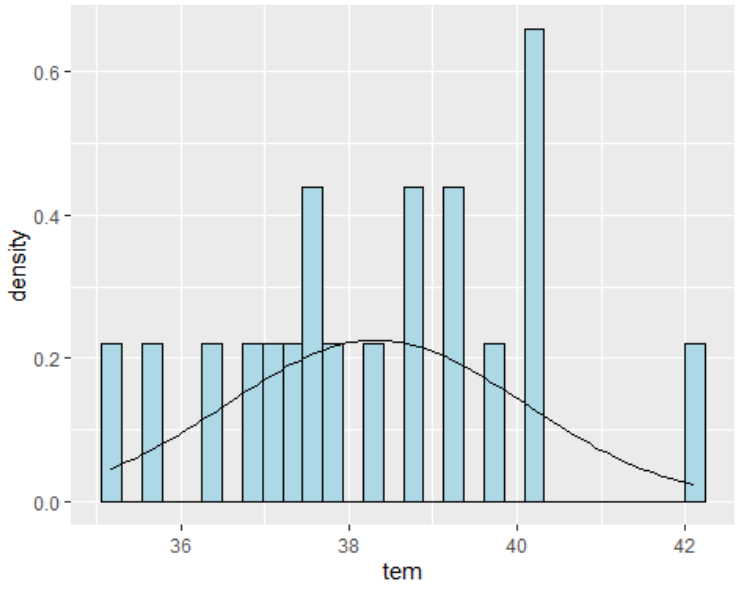


Figura 4.7. Distribuição de frequência da temperatura de 19 pacientes

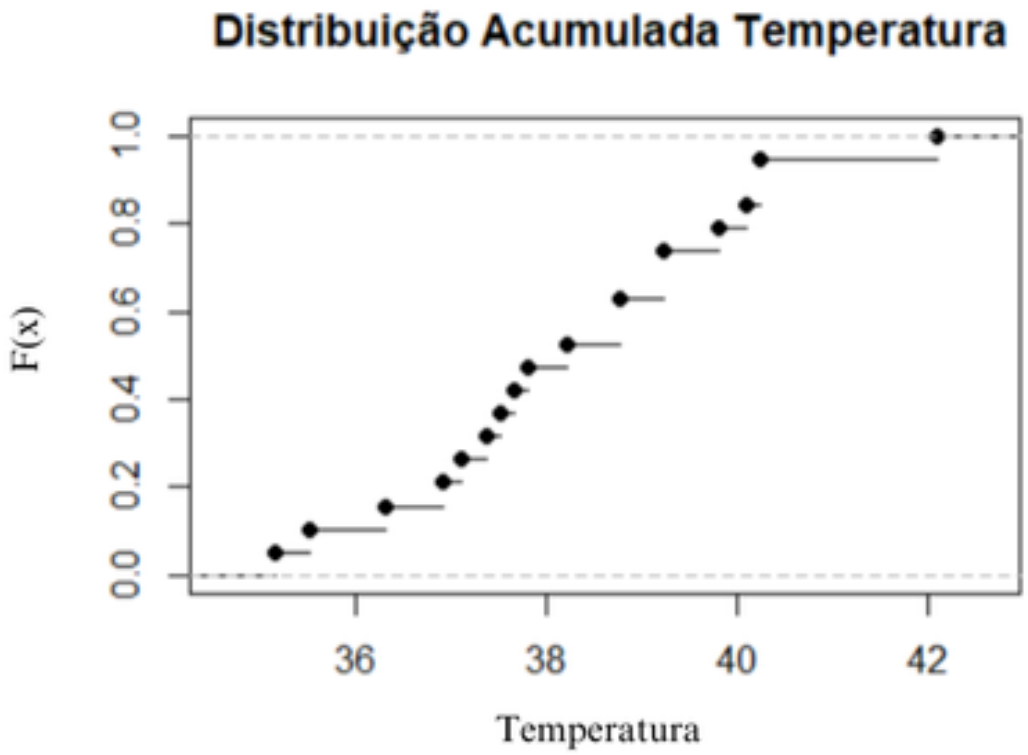


Figura 4.8. Distribuição Acumulada da temperatura de 19 pacientes

O formato no gráfico é diferente daquilo que conhecemos como formato de uma distribuição acumulado, isso porque nesse exemplo temos poucas observações o que não permite suavizar a curva da distribuição acumulado. Para tentarmos visualizar melhor a cara dessa distribuição, simularemos uma distribuição normal com média 38,33 e desvio padrão 1,77 (mesma distribuição dos dados de temperatura), mas aumentamos o tamanho amostral para 10.000.

```
amostra = rnorm(10000, 38.33, 1.77)

fn_amostra = ecdf(amostra)

plot(fn_amostra, main = "Distribuição Acumulada
Temperatura", xlab = "Temperatura", ylab="F(x)")
```


4.2.3. Recursive Model Index: o primeiro índice proposto

O framework proposto em Kraska et al. (2018) inclui o RMI, uma hierarquia de regressões estruturada para permitir que o modelo de cada nível receba a chave como entrada e a transfira para o modelo da camada seguinte mais apto a predizer qual a posição de saída referente à chave buscada. O funcionamento do RMI é baseado na observação da localização de uma chave em um vetor ordenado e a capacidade de conseguir calculá-la usando uma função de distribuição acumulada dos dados. Este método, portanto, é responsável por estimar a CDF de um conjunto de dados, mapeando uma chave de busca para a sua posição num vetor ordenado [Maltry and Dittrich 2021]. Com isso, cada modelo é capaz de fazer uma previsão com um certo erro para a posição da chave e com isso é escolhido o próximo modelo que é responsável por outra área do

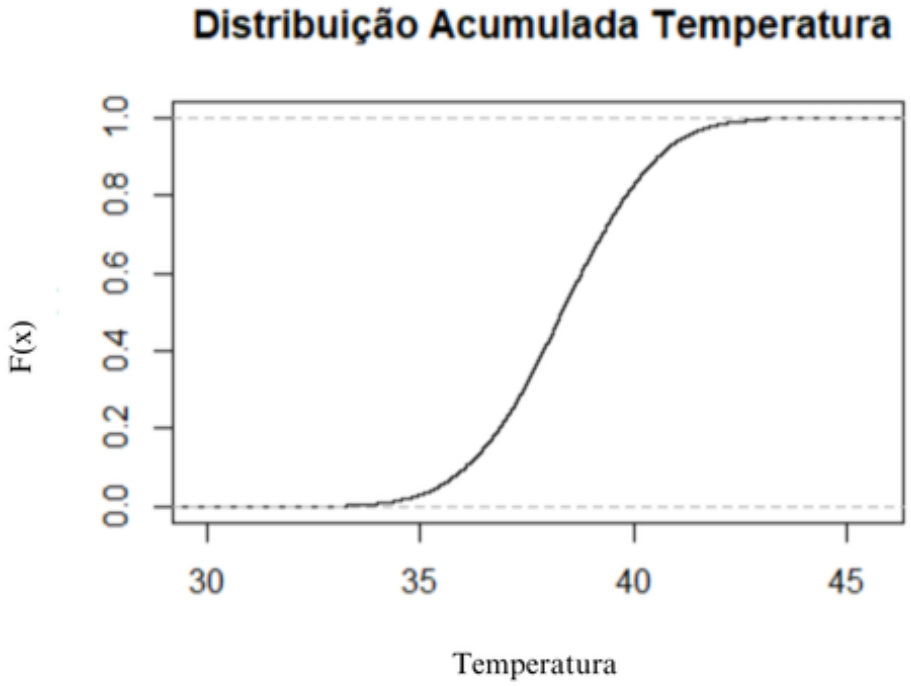


Figura 4.9. Distribuição Acumulada de dados simulados da temperatura de 10.000 pacientes

espaço de chaves e esse consegue fazer uma previsão mais assertiva com um erro menor que o modelo anterior. O RMI pode ser considerado como um grafo acíclico dirigido, diferentemente de um árvore, um nó (ou uma folha) do modelo recursivo pode ter múltiplos predecessores diretos e cada camada abaixo pode consistir em um número arbitrário de modelos [Maltry and Dittrich 2021].

A utilização do modelo RMI acarreta em alguns benefícios para o framework, Kraska et al. (2018) traz 4 deles, sendo: o tamanho e a complexidade do modelo são separadas para o custo de execução; o aprendizado da distribuição de dados é algo considerado fácil; o espaço de dados é dividido de maneira eficaz em subfaixas menores

(assim como na B-Tree) e não há uma busca entre os estágios dos modelos, fazendo com que a saída de um seja diretamente utilizada para selecionar o modelo da próxima etapa, fazendo com que o número de instruções gerenciais sejam reduzidas. Outra vantagem que a utilização de LI traz com o RMI é a capacidade de modelos híbridos. Por exemplo: em camadas superiores a utilização de uma RN ReLU pode se demonstrar como o mais eficaz, porém nas camadas intermediárias modelos de regressão linear podem se encaixar melhor por serem mais econômicos e, além desses modelos, a própria árvore B-Tree pode ser utilizada nos nós folhas, caso a distribuição dos dados apresente uma dificuldade de aprendizagem.

4.2.4. Índices para suportar OLTP

Por mais que os índices aprendidos apresentem uma boa resposta para atividades de leitura, as atividades de escrita eram um desafio para o primeiro modelo. Visando contornar esse problema, Ding et al. (2020) traz o ALEX, um modelo de learned index que oferece suporte a buscas pontuais, consultas de espaço, inserções, atualizações, exclusões e workloads, operações comuns em processamento de transações online (OLTP). Resumidamente, esse LI reserva espaços em memória para que novos valores possam ser inseridos sem a necessidade de deslocamento de chaves para cada inserção. O ALEX mantém os modelos precisos utilizando de mecanismos de expansão adaptativa e divisão dos nós atrelados com retreinamento de modelos, fazendo com que a estrutura seja possível de reagir a cargas de trabalho reais com mudanças dinâmicas.

Ding et al. (2020) traz algumas diferenças entre o LI e o ALEX. A primeira é a estrutura, onde o ALEX se assemelha com a árvore B+ para armazenar os dados nas folhas, utilizando apenas um nó por folha e com isso é possível realizar divisões e expansões mais facilmente para uma inserção. O ALEX usa de espaços livres nas folhas organizadas de forma estratégica para que a inserção e busca sejam mais eficientes. A próxima diferença é que o LI usa busca binária dentro dos limites fornecidos pelo modelo, enquanto o ALEX utiliza de busca exponencial para encontrar os valores dentro das folhas. Essa alteração permite uma busca mais rápida que a binária e também elimina a necessidade do armazenamento de limites de erro nos modelos do RMI. A terceira diferença é a inserção do ALEX ser feita na localização de onde a chave deve estar, essa é prevista pelo modelo e foi chamada de “Inserção Baseada em Modelo” (do inglês *model-based insertion*). A quarta diferença é que o ALEX consegue ajustar a forma e a altura do RMI de forma dinâmica. Por fim, a quinta diferença é que o ALEX não precisa ter seus parâmetros reajustados por carga de trabalho, diferentemente do LI que precisa ajustar o número de modelos.

O *layout* do ALEX apresenta dois tipos diferentes de nós: o folha e o interno. No nó folha (nós de dados), são armazenados os registros de dados. Esse nó possui um modelo de regressão linear que mapeia a chave para uma posição e possui dois vetores com lacunas, onde um mapeia a chave para uma posição e o outro é responsável pelas cargas de dados. Para não perder eficiência na busca, os vetores com lacunas ocupam os espaços vazios com a chave mais próxima à direita, fazendo com que a busca não seja afetada por esses espaços. Para passar pelas lacunas na busca, cada nó possui um bitmap que verifica se o próximo espaço está ocupado por uma chave ou se é um espaço vazio.

Já os nós internos são todos os que estão na estrutura do RMI, nesses são armazenados um modelo de regressão linear juntamente com um array contendo os ponteiros dos nós filhos. Os nós internos calculam a localização no array de ponteiros do próximo filho.

Já para a etapa de busca, o ALEX inicia no nó raiz do RMI e o modelo é utilizado de maneira iterativa para calcular uma localização nos ponteiros para o próximo filho, essa atividade é repetida até chegar no nó folha. Ao chegar no nó de dados é realizado uma previsão da chave de busca e — caso necessário — uma busca exponencial que retorna a posição real da chave. Caso essa seja encontrada, o valor correspondente é retornado e caso não o encontre, um registro nulo é retornado. Já para a busca de intervalo é realizada uma busca para a primeira chave onde o valor não é menor que o valor inicial do intervalo e depois é percorrido até chegar ao valor final.

Como explicado acima, o ALEX é um modelo capaz de realizar inserções no banco de dados e para essa etapa dois casos possíveis: em nós cheios e em nós não cheios. Na primeira, o algoritmo de busca é executado e ao achar o nó no qual o dado deveria estar, é feita a inserção. Nesse caso, caso o espaço seja uma lacuna, o dado é inserido e caso o espaço possua valor, um espaço vazio é criado e os elementos são empurrados em direção à lacuna mais próxima.

Na inserção de dados em nós cheios são executados alguns passos: primeiramente o nó é expandido criando um novo vetor de lacunas maior e o modelo é re-treinado ou escalado e os dados são inseridos nesse espaço. Ao realizar a divisão do nó, duas vertentes podem ser aplicadas: a lateral que se assemelha a B+ -Tree onde caso o nó pai ainda não tenha atingido o tamanho máximo, o nó de dados é dividido e o ponteiro do pai para o nó dividido é substituído para ponteiros para os novos nós de dados. E caso o nó do pai tenha atingido o tamanho máximo do nó, esse pode ser dividido e novos ponteiros são gerados para os nós filhos. Esse modelo de divisão se assemelha à B+ -Tree e a divisão pode chegar até o nó raiz. A segunda opção de divisão é a descendente, onde um nó de dados é convertido em um interno com dois filhos e esses são treinados com suas chaves.

Por fim, as exclusões e atualizações no ALEX são mais simples que as inserções, visto que para deletar um valor a busca é realizada para encontrar a chave e após isso faz-se a remoção. Essa atividade não realiza o deslocamento de chaves, tornando a operação mais simples que a inserção, caso o nó que possuía o valor acabe chegando ao limite inferior de densidade, o nó é retraído. Já as atualizações mesclam uma inserção com uma remoção.

4.2.5. Indexando dados multidimensionais

Índices multidimensionais são alternativas para os índices secundários no armazenamento de dados. Eles podem ser baseados em árvores (como a árvore k-d ou a árvore R) ou podem ser baseados em múltiplos atributos, como o Z-ordering. Sistemas como o Redshift, Spark SQL, Vertica e IBM Informix são alguns dos exemplos de banco de dados que usam técnicas de indexação de dados multidimensionais. Porém, esse modelo de indexação ainda apresentam desvantagens, como a dificuldade de ajuste (nesse caso os administradores de banco de dados precisam ajustar manualmente qual a

ordem de classificação e precisa saber quais colunas são acessadas juntas); outra desvantagem é que o índice multidimensional não possui uma universalidade de padrão, visto que esses variam conforme a distribuição dos dados e com a carga de *workload* [Nathan et al., 2020].

Com as limitações mostradas anteriormente, um framework de LI multidimensional em memória foi proposto em Natan et al. (2020). Nomeado de Flood, esse modelo objetiva buscar registros que correspondam a uma consulta de maneira

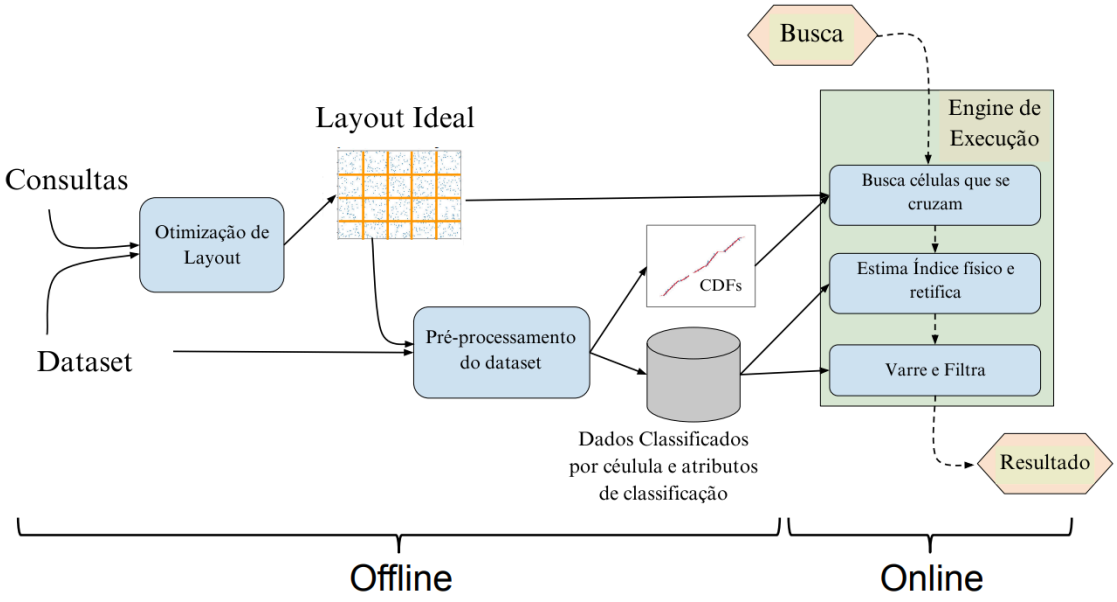


Figura 4.10. Arquitetura do Flood, é possível ver a distinção entre as partes offline e on-line. Fonte: Traduzido de [Natan et al, 2020]

mais rápida que os demais modelos de indexação existentes, para isso a estrutura do índice e o layout dos dados conseguirão ser ajustados automaticamente para uma distribuição de dados e consultas específicas. De maneira resumida, o Flood analisa uma carga de trabalho amostral para aprender a frequência que as dimensões são usadas, quais dessas são usadas juntas e quais são mais seletivas, com isso o layout do Flood é personalizado para otimizar a consulta dada. Outra característica do Flood é usar modelos empíricos de CDF para "projetar a distribuição de dados multidimensional e potencialmente enviesada em um espaço mais uniforme" Natan et al. (2020), com isso uma limitação do número de pontos pesquisados é aplicada e um melhor desempenho pode ser encontrado.

O Flood consegue suportar desde transações em memória OLTP à data warehouses em disco, porém o objetivo central do framework proposto é melhorar o desempenho de índices multidimensionais para um armazenamento em coluna em memória (os quais estão se tornando cada vez mais comum pela queda dos preços da memória RAM e aumento da memória principal alocada em máquinas). Outra característica do Flood é a otimização da leitura à custa da escrita, com isso a performance é maior em ambientes analíticos relativamente estáticos.

Por assumir que as consultas OR podem ser feitas através de diversas consultas de intervalo e que consultas de igualdade (ex: R.Z == f) pode ser reescrita como " f <= R.Z <= f ", o Flood foca em cláusulas AND [Natan et al, 2020]. Sendo assim, o framework funciona em duas partes: primeiramente há um pré-processamento offline em que um layout ótimo é escolhido e o índice será baseado neste. A segunda etapa é online, onde um componente é responsável por executar as consultas na medida que chegam [Natan et al, 2020]. A arquitetura do Flood pode ser vista na Figura 4.10.

Para a estruturação dos dados, levando em consideração que ao contrário de dados unidimensionais, os pontos em múltiplas dimensões não possuem uma ordem de classificação natural, o Flood inicia com tratativas para impor uma ordenação sobre os dados. Para isso, os d atributos do dado são classificados. Com isso, as d -1dimensões são são usadas na ordenação para sobrepor a grade (d-1)-dimensional dos dados, onde i-ésima dimensão na ordenação é dividida em c_i colunas igualmente espaçadas entre seus valores mínimos e máximos. Caso M_i e m_i sejam os valores máximo e mínimos dos dados na i-ésima dimensão, o intervalo da dimensão é $r_i = M_i - m_i + 1$ [Natan et al, 2020]. Com isso, a célula para o ponto p = (p_1 , … , p_d) é:

$$\text{célula}(p) = \left(\left\lfloor \frac{p_1 - m_1}{r_1} \cdot c_1 \right\rfloor, \ldots, \left\lfloor \frac{p_{d-1} - m_{d-1}}{r_{d-1}} \cdot c_{d-1} \right\rfloor \right)$$

Na fórmula apenas as d - 1 dimensões aparecem, pois a dimensão d é utilizada para ordenação dos pontos dentro da célula. Durante essa etapa, uma travessia em profundidade é utilizada nas células ao longo da ordenação das dimensões, sendo assim, as células são classificadas por cardinalidade dentro da tupla (1º valor, 2º valor, … , n-ésimo valor). Já dentro da célula os pontos são classificados pelo seu valor na

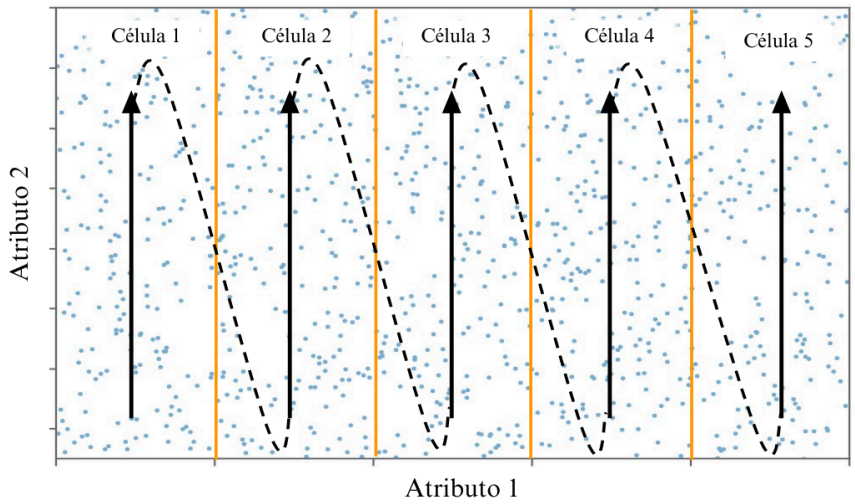


Figura 4.11. Estrutura de ordenação de 5 células em um cenário com dados de duas dimensões. Fonte: Traduzido de [Natan et al, 2020]

dimensão d. Na Figura 4.11 é possível ver essa movimentação da classificação [Natan et al, 2020].

O Flood opera em um fluxo de três etapas: projeção, refinamento e escaneamento. Nesse processo, uma entrada com um predicado de filtro de intervalos sobre um ou mais atributos (unidos por ANDs) é recebida. O Flood encontra e processa os pontos dentro do hiper-retângulo formado pela interseção dos intervalos de entrada [Natan et al, 2020]. A Figura 4.10 exemplifica esse fluxo.

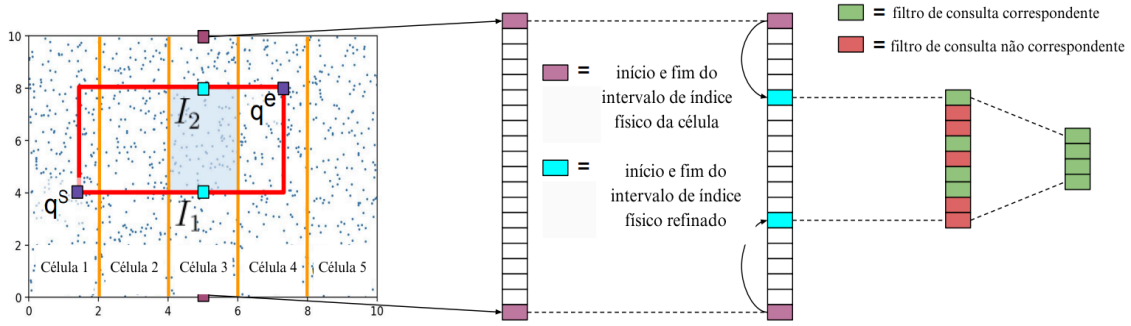


Figura 4.12. Passo-a-passo da operação do Flood

A etapa de projeção consiste em determinar quais pontos correspondem a um filtro. Para isso, o Flood inicia determinando quais as células que possuem os pontos correspondentes. Caso uma dimensão indexada não esteja na consulta, os pontos de início e fim são tomados como -∞ e +∞. Porém, caso a consulta inclua uma dimensão que não está indexada, o filtro é ignorado nessa etapa do processamento da consulta.

A Figura 4.12 também mostra a formação do hiper-retângulo (etapa de projeção), onde o canto inferior esquerdo é $q^8 = (q^8_0 , \dots , q^8_{d-1})$ e o mesmo para o canto superior direito q^e. Sendo assim, as células que estão no intervalo do hiper-retângulo são representadas pelo conjunto:

$$\{C_i \mid \text{célula}(q^s)_i \leq C_i \leq \text{célula}(q^e)_i\}$$

E o Flood usa uma tabela de células que registra o índice físico do primeiro ponto em cada célula. O conhecimento das células que estão no intervalo delimitado se traduz em um conjunto de intervalos físicos de índices para escanear [Natan et al, 2020].

Quando uma consulta traz um filtro que usa da dimensão de ordenação (a dimensão d, relatada anteriormente), o Flood utiliza do fator de ordenação dos pontos da célula e refina ainda mais os intervalos de índice físico que precisam ser varridos. Por exemplo: em uma consulta que inclua um filtro sobre a dimensão de ordenação R.S onde a <= R.S <= b. O Flood encontrará os índices físicos do ponto I_1 onde R.S >= a e encontrará os índices físicos do ponto I_2 onde R.S <= b. Essa etapa ajuda com que o intervalo de busca de índices físicos para $[I_1,I_2]$ seja reduzido. Como os pontos em C são armazenados de forma classificada pela dimensão d, uma busca binária em C pode ser utilizada para encontrar o $[I_1,I_2]$.

Por mais que o Flood consiga trazer soluções concretas para a utilização de learned index em dados multidimensionais, algumas limitações foram encontradas na sua operação, como: existe uma restrição para se adaptar de maneira eficiente em cargas de trabalho com consultas enviesadas, onde a frequência de consulta e a seletividade dos filtros variam no espaço dos dados. Outra dificuldade é com dimensões correlacionadas,

o Flood não consegue manter células de grade de tamanho uniforme, o que acaba piorando o desempenho e o uso de memória do framework [Diang et al. 2020]. Visando superar essas limitações, Diang et al. (2020) propõe o Tsunami, um learned index otimizado para leitura na memória para dados multidimensionais.

O Tsunami consegue um ótimo desempenho para workloads de consultas enviesadas com a utilização de uma árvore de decisão leve que é chamada de Grid Tree (Árvore de Grade), a qual particiona o espaço em regiões não sobrepostas de forma que o enviesamento das consultas seja reduzido. Outro diferencial apresentado pelo Tsunami é a capacidade de atuar com correlações, através da indexação por regiões, alcança um alto desempenho em conjunto de dados correlacionados utilizando uma grade aumentada (essa usa mapeamentos funcionais e CDFs condicionais) [Diang et al. 2020].

Como relatado anteriormente, o Tsunami é uma composição de duas estruturas de dados independentes, árvore de grade e a grade aumentada. A árvore de grade é uma árvore de decisão de partição espacial e divide o espaço de dados com d dimensões em regiões não sobrepostas. Em cada das dimensões existe uma grade aumentada e essa indexa os pontos de sua região. Porém, caso nenhuma consulta faça interseção com sua região, essa não terá uma grade aumentada. Baseado no Flood, a grade aumentada é uma generalização do seu sistema de indexação, porém elas utilizam F: Y -> X e CDF(Y|X) enquanto o Flood utiliza CDF (Y) [Diang et al. 2020].

De forma mais detalhada, a Grid Tree é uma árvore de decisão de particionamento de espaço, a qual se assemelha com a árvore k-d. Cada nó interno da árvore de grade faz a divisão espacial com base nos valores de uma dimensão específica. Essa nova estrutura se difere da árvore k-d na granularidade dos nós, enquanto a k-d é uma árvore binária, os nós internos da árvore de grade podem se dividir em mais de um valor. No processamento de consultas, a árvore de grade é percorrida para encontrar todas as regiões que fazem interseção com os predicados da consulta. Caso haja um índice sobre os pontos nessa região, uma consulta é enviada para esse índice e os resultados são agregados e retornados. Caso não haja o índice nessa região, todos os pontos serão percorridos [Diang et al. 2020].

Com isso, Diang et al. (2020) apresentaram um índice aprendido multidimensional que utiliza de duas novas estruturas de dados (Árvore de Grade e Grade Aumentada) e é capaz de superar os demais learned index para dados multidimensionais em até 6x na taxa de transferência de consulta e 8x no espaço.

4.2.6. Indexando strings

Outra limitação que o primeiro framework de LI apresenta é a indexação de dados textuais. O suporte a workloads com dados desse tipo é custoso na inferência do modelo e no acesso dos dados que aumentam com o comprimento da chave. Outra questão é que a precisão do modelo diminui e a quantidade de acesso aos dados aumentam, isso pode ser resolvido utilizando modelos mais sofisticados como deep learning (aprendizagem profunda), porém esse benefício acaba por acarretar um tempo maior de treinamento e inferência, ocasionando um desempenho mais baixo que o esperado [Wang et al., 2020].

Wang et al. (2020) compara a indexação do XIndex com o Masstree para dados de string e o desempenho de escrita e de leitura caem significativamente quando o tamanho da chave é aumentado. Por exemplo: quando o Xindex possui uma chave de 128 bytes em uma carga de leitura, o desempenho cai em 70% quando comparado com uma chave de 8 bytes. Já em uma carga de leitura e escrita, o desempenho do XIndex cai em mais de 66%, enquanto o Masstree apresenta uma queda de apenas 34%.

Essas diferenças expressivas de desempenho em chaves de texto são causadas por alguns motivos, sendo eles: o custo de processamento do modelo aumenta significativamente com o comprimento da chave, até mesmo para um modelo simples como o linear, onde uma chave de 128 bytes pode apresentar um aumento de 23 vezes que a computação de uma chave de 8 bytes. Outro fator que influencia nessa queda de desempenho é o aumento do erro do modelo, onde aumenta de 24 para 68 quando a chave cresce de 8 para 128 bytes, essa característica faz com que os mais registros sejam acessados durante as buscas. Mesmo que modelos complexos como as redes neurais reduzem esse erro, o tempo de computação deles não apresenta melhora no desempenho. Outro fato é que a busca binária também acaba sendo custosa em buscas de strings e são afetadas pelo aumento dessas [Wang et al., 2020].

Visando superar essas dificuldades e buscando, o framework de índice aprendido escalável Sindex é proposto por Wang et al. (2020) com o objetivo de indexar eficientemente chaves de string. Em resumo esse modelo agrupa de forma gananciosa as chaves com prefixos comuns e em seguida cada grupo usa a parte única das chaves e a chave parcial para treinar o modelo e indexar os dados.

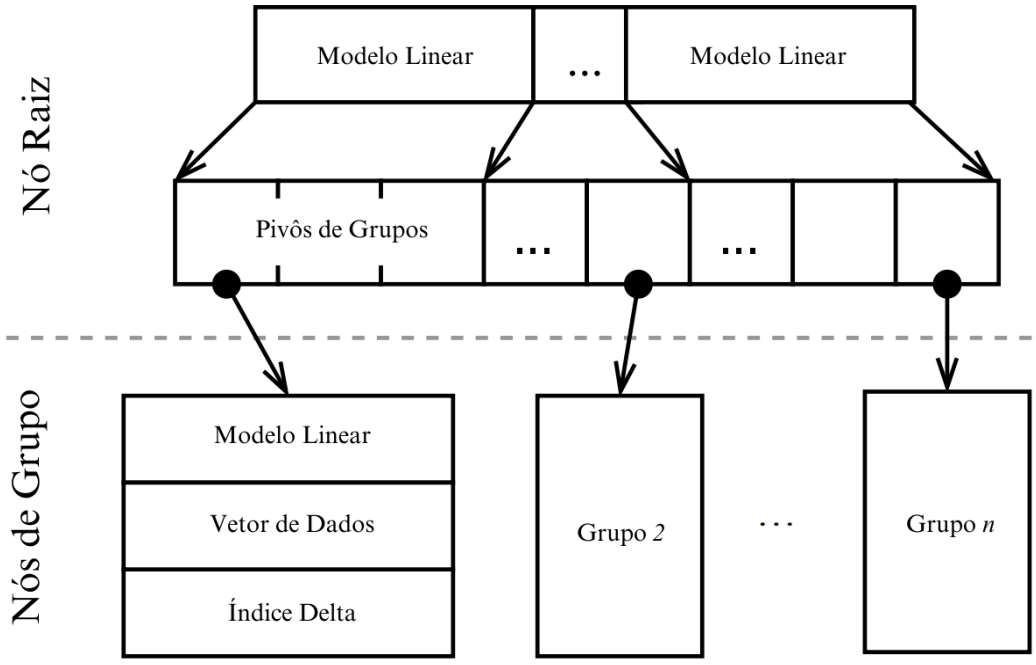


Figura 4.13. Arquitetura do Sindex. Traduzida de Wang et al. (2020)

O Xindex possui uma arquitetura separada em duas camadas, um nó raiz e vários grupos de nós na camada abaixo. Nesse layout cada grupo armazena registros de um

intervalo específico de chaves, onde ficam armazenados em um vetor ordenado e indexados por um modelo linear, ou ficam em delta buffers caso tenham sido oriundos de inserção. Já o nó raiz usa de vários modelos lineares para indexar os grupos e conta com o armazenamento das chaves pivô de cada grupo, que limitam o intervalo inferior inferior da divisão. Rodando em background, uma verificação contínua de todos os grupos é uma das atividades do SIndex, onde é analisado se o tamanho do índice delta de um determinado grupo está maior que o limite configurado pelo usuário. Caso seja encontrado valores que correspondam à essa busca, uma compactação é realizada e o índice delta é mesclado com o antigo vetor ordenado em um novo e os modelos passam por novo treinamento para se adequar à nova disposição dos dados [Wang et al., 2020].

A primeira etapa do XIndex é a extração das chaves parciais, as quais são substrings que possuem comprimento fixo extraídas das chaves originais de cada grupo. Com essa etapa é possível remover os prefixos comuns das chaves, ou seja, é possível eliminar a parte inicial das chaves que é igual entre todas, fazendo com que o comprimento da chave seja reduzido. Também é possível manter a distinção entre as substrings, fazendo com que essas possam ser suficientemente diferentes a fim de garantir a unicidade e ordenação correta em relação às demais. Para as chaves parciais pode-se utilizar a notação: para qualquer chave k em um grupo, a chave parcial é k[pl: pl+el] onde pl é o comprimento do maior prefixo comum e el é o menor comprimento de prefixo entre as chaves após a remoção que garanta que todas as chaves ainda sejam iguais [Wang et al., 2020]. Tomando o grupo [“cardiologista”, “cardiopatia”, “cardiograma”] como exemplo, o pl será “cardio” (6 letras). Já o menor comprimento necessário para distinguir as chaves restantes será el = 1, pois as substrings “l”, “p” e “g” são suficientes para diferenciá-las. As parciais melhoram a eficiência da computação do modelo e do seu treinamento, além de reduzir a complexidade da busca binária, fazendo-a apenas nas chaves parciais devido a preservação da ordem.

A próxima etapa é o particionamento dos grupos de dados, visando que o comprimento da chave parcial e os erros dos modelos estejam dentro dos limites especificados pelo usuário e para isso uma estratégia gananciosa é utilizada. Os parâmetros são: et - limite de erro do modelo; pt - limite de comprimento da chave parcial; fs - quantidade de registros inseridos por vez; bs - quantidade de registros removidos por vez. Essa etapa tem início na interação de todos os registros do conjunto de dados e em cada grupo é determinado se novos registros devem ser inseridos ou se valores já existentes devem ser removidos, com base nos limites de erro. Ao adicionar os registros, um modelo linear é treinado no grupo para calcular o erro médio. Caso a inserção fs cause uma violação no limite do comprimento da chave parcial ou no limite do erro do modelo, são removidos os bs registros mais recentes, essa etapa é repetida até o comprimento da chave parcial e o erro do modelo estejam dentro dos limites definidos. Com esses passos o SIndex ajusta dinamicamente os grupos de dados e otimiza a eficiência do modelo de indexação, garantindo que os grupos estejam formados de maneira a manter o erro do modelo e o comprimento da chave parcial dentro dos limites aceitáveis. Com isso é alcançado uma indexação mais eficiente e precisa, enquanto melhora o desempenho do sistema como um todo [Wang et al., 2020].

No nó raiz é utilizado um modelo linear segmentado que tem objetivo dividir o espaço de chaves em segmentos menores e mais gerenciáveis, onde cada segmento é

tratado por um modelo linear separado, fazendo com que a busca e a precisão da previsão sejam melhoradas para as chaves de string. O treinamento desses modelos é feito usando o pivô de cada grupo (esse é definido usando o mesmo algoritmo ganancioso do agrupamento, assim também é determinado o intervalo de chaves de cada modelo). Já a inferência é feita através de busca binária para encontrar qual o modelo linear que é responsável pelo intervalo de chaves da solicitação. O SIndex substitui o RMI pelo modelo linear segmentado visando solucionar duas dificuldades: no RMI, dois pivôs distantes podem ser classificados pelo mesmo modelo folha, resultando em um grande intervalo de chaves nesse nó, resultando em um intervalo grande e gerando dificuldade na remoção dos prefixos comuns que reduz o benefício das chaves parciais e acarreta em uma perda de eficiência.

4.2.7. Índices de alto desempenho

Visando superar desafios de atividades concorrentes, o XIndex foi desenvolvido para ser uma estrutura de learned index concorrente, acelerando a busca com modelos aprendidos e lidando com escritas concorrentes. Uma das características de sua funcionalidade é particionar os dados em diversos grupos e cada um está associado a um modelo aprendido para busca e a um índice delta para realizar inserções. Outra vantagem do XIndex é atualizar sua estrutura de acordo com a carga de trabalho recebida, para isso são realizadas fusões e divisões de grupos parametrizadas por valores (limite de erro, limite de tamanho do delta e etc) ajustados pelo usuário [Tang et al., 2020].

Assim como foi falado anteriormente, o primeiro modelo de LI não apresentava maneiras eficientes de escrita de dados, principalmente quando trata-se de operações concorrentes. Para isso o XIndex traz a utilização do índice delta para armazenar chaves inseridas, neste os dados recém inseridos são gravados e no momento da busca o delta é consultado apenas caso a chave não seja encontrada no learned index. Para compactar o índice delta, foi implementada uma Compactação de Duas Fases e essa é realizada em segundo plano, visando não bloquear operações concorrentes [Tang et al., 2020].

A estrutura do XIndex é semelhante à apresentada pelo SIndex, onde ambos contam com uma arquitetura de duas camadas. Na camada do nó raiz os nós de cada grupo da camada inferior são indexados através de um modelo de RMI. Já nos nós folha, modelos lineares são utilizados para indexar seus dados e um índice delta armazena temporariamente as inserções do seu referente grupo [Tang et al., 2020].

Com isso, é notável que o layout do XIndex consiste de três estruturas básicas: record_t que é a representação dos dados, consiste das chaves, valores e de metadados (is_ptr indica se o valor é real ou é uma referência de memória; removed que indica se o registro foi removido logicamente; lock e version que são usados para controle de concorrência, garantindo que as operações concorrentes serão executadas de maneira exclusiva e segura. A segunda estrutura é o root_t (nó raíz), esse guarda as informações sobre os grupos de dados e utiliza de um RMI para prever qual o grupo que possui uma chave específica. Ele guarda três informações sobre os grupos: endereço dos grupos (groups), menor chave de cada grupo (pivots) e o número total de grupos (group_n). Para realizar o treinamento do RMI, o XIndex faz uso dos pivots dos grupos e seus

índices. A última estrutura é o nó de grupo (group_t), o qual é responsável por gerenciar os dados reais, os modelos de indexação e os índices delta encarregados de receber as inserções. Os dados (data_array) armazenam todos os registros do grupo de maneira contínua. Os modelos (models) contém pelo menos um modelo linear para indexar os registros do data_array e o model_t inclui os parâmetros do modelo linear e a menor chave do intervalo de dados que o modelo cobre. Já o índice delta (buf) armazena todas as inserções temporárias, o buf_fronzen é o indicador de que o buf está congelado durante a compactação e o tmp_buf é um índice delta temporário utilizado durante a compactação. Todo esse layout auxilia o XIndex gerenciar operações de leitura e de escrita, enquanto mantém a integridade dos dados e alcança um desempenho otimizado [Tang et al., 2020].

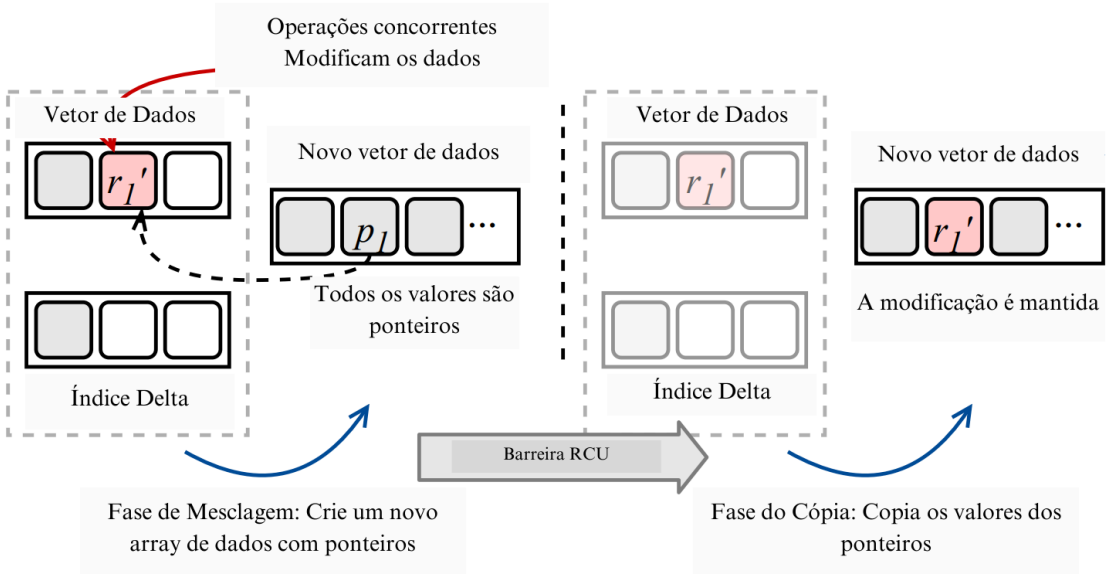


Figura 4.14. Compactação de duas fases do XIndex. Fonte: modificada de Tang et al (2020).

O XIndex também apresenta a capacidade de ajustar sua estrutura em tempo de execução, dessa forma um grupo que apresenta um alto erro de previsão recebe mais modelos lineares para uma melhora de performance. Caso um grupo esteja com muitos modelos ou o índice delta esteja sobrecarregado, o XIndex consegue dividi-lo em dois novos grupos. A mesclagem de modelos e grupos podem ser feitas desde que não apresentem queda da precisão de previsão e o re-treinamento do modelo de RMI pode ser treinado para se readequar às novas configurações dos grupos [Tang et al., 2020].

O XIndex apresenta as operações básicas de índices (get, put, remove e scan) que permitem a manipulação dos dados. No geral, todas as operações são realizadas tendo início com a identificação do grupo apropriado que contém o registro buscado, para isso o nó raiz prevê um grupo usando o RMI e em seguida ocorre uma busca binária nos pivôs dos grupos dentro de um intervalo limitado por erro. Caso o grupo encontrado pela operação get_group possua o ponteiro next, o XIndex seguirá em busca do grupo correto, pois os grupos podem ter sido recém criados. Quando o grupo é

achado, para a operação de busca no data_array, o XIndex usa o modelo linear apropriado para prever a posição do dado dentro do vetor e novamente uma busca binária é realizada com base no erro do modelo. Caso um registro seja encontrado e a operação seja get, o XIndex lê o valor, se o registro estiver logicamente removido, o XIndex continua a busca no buf do nó e talvez no temp_buf, um valor é retornado caso a busca encontre um resultado não vazio. Para as operações de put e remove, o XIndex

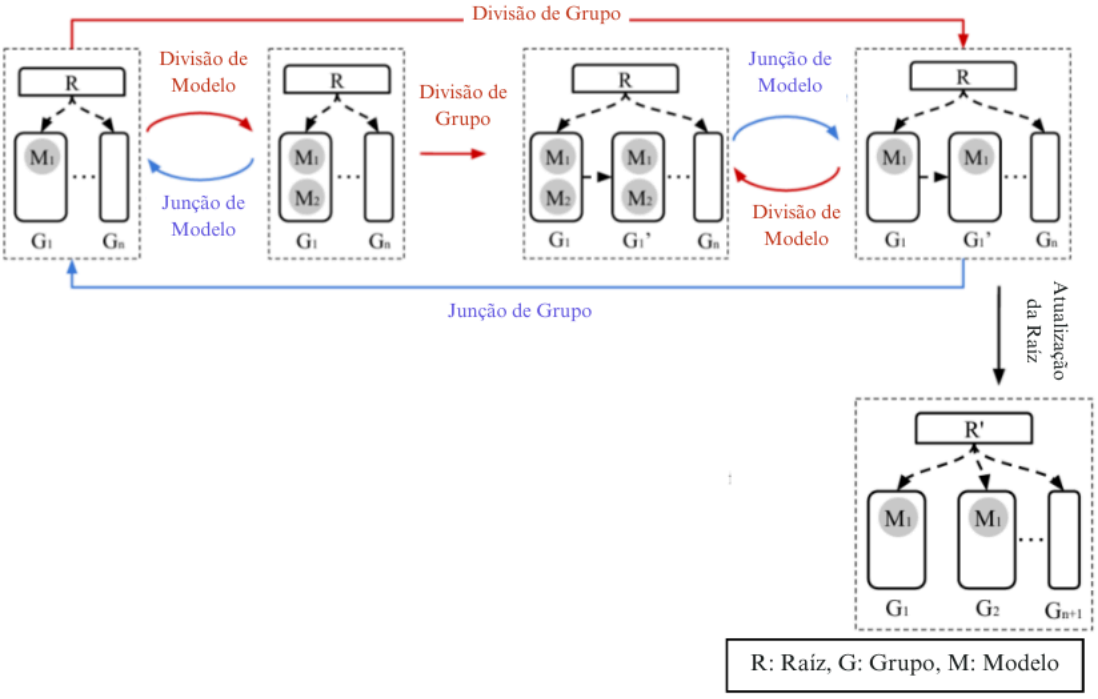


Figura 4.15. Exemplos das demonstrações de partição de grupo, nó e modelo do XIndex. Fonte: modificada de Tang et al. (2020).

tenta atualizar ou remover o valor diretamente no data_array e, caso a operação não possa ser feita diretamente, essa será realizada no buf ou no tmp_buf, caso a flag frozen_buf esteja ativa. Já a operação scan localiza o menor registro que é maior ou igual à chave solicitada e lê consistentemente um número definido de registros consecutivos [Tang et al., 2020].

Como dito anteriormente, o XIndex realiza uma compactação em duas etapas com o objetivo de suportar operações concorrentes. A primeira etapa é a mesclagem, onde o data_array de um grupo e o buf serão juntados em um novo array ordenado que possui ponteiros para os registros existentes. Para isso, o marcador (flag) buf_frozen é ativado para evitar novas operações de put no buf que está sendo mesclado e em seguida o tmp_buf é inicializado para armazenar inserções durante a compactação. Feito isso, um novo grupo é criado para armazenar o resultado da junção do data_array e do buf do grupo antigo em um novo data_array. No novo vetor de valores, cada valor de cada registro é uma referência para o registro correspondente do grupo antigo e a flag is_ptr de cada registro é atualizada para verdadeiro. Nesse novo grupo, a posição de buf será ocupada pelo antigo tmp_buf e após treinar os modelos lineares e finalizar de configurar os metadados do novo grupo, o XIndex substitui o grupo antigo pelo novo fazendo com

que a referência do grupo em groups seja atualizada. A segunda etapa é a cópia, onde cada referência do data_array é substituída pelo valor mais recente do registro, em paralelo, o XIndex usa da rcu_barrier para esperar que cada worker processe uma solicitação, fazendo com que o grupo antigo não seja acessado após a barreira. Finalizadas todas as etapas, a memória usada pelo grupo antigo pode ser liberada [Tang et al., 2020]. A Figura 4.14 demonstra a funcionalidade da compactação de duas etapas.

Outra das propriedades do XIndex que foi comentada anteriormente é a capacidade de atualizar sua estrutura em divisão ou fusão de modelos; divisão ou fusão de grupos e atualização no nó raiz. Na divisão de modelos o XIndex clona o nó responsável do grupo e ambos os nós estão referenciando os mesmos dados. Dando sequência, o número de modelos no novo nó é incrementado e os dados dos grupos são redistribuídos de maneira uniforme para cada modelo, feito isso todos são re-treinados e a referência no group do nó raiz é atualizada para apontar para o novo grupo. Já a fusão de modelos é a operação inversa da separação. A divisão de grupos consiste em criar dois grupos lógicos que compartilham os mesmos dados, porém possuem buf e tmp_buf diferentes para armazenar as novas inserções durante a divisão. Depois disso a atualização dos metadados é realizada e as referências de pivô e organização dos ponteiros entre os grupos mudam. A segunda etapa da divisão de grupos inicia com a mesclagem dos data_array e buf em um novo data_array, dividindo-o em dois novos grupos com base no pivot. Finalizado isso, as referências do data_array são substituídas pelos valores reais e os ponteiros de referência são atualizados. A fase de fusão tem um processo semelhante à divisão de grupos onde os data_arrays e os bufs são mesclados e as referências são substituídas pelos valores reais, finalizado essas etapas, o nó raiz é atualizado com as novas informações da fusão. Por fim, a etapa de atualização do nó raiz passa pelo achatamento da estrutura dos grupos para reduzir a latência de acesso dos ponteiros, o modelo RMI é re-treinado para garantir precisão nas previsões e o ponteiro global do nó raiz é substituído para garantir a consistência e integridade dos dados. Essas modificações estruturais estão representadas graficamente na Figura 4.15.

Visando tratar operações concorrentes de escrita com a mesma chave, o XIndex adiciona um bloqueio por registro ao data_array e a operação concorrente é realizada no buf. Ou seja, primeiramente as escritas são endereçadas ao vetor, porém, se a atualização nesse não estiver disponível, o escritor recorre ao índice delta, que é protegido por um bloqueio de leitura-escrita. Já para a leitura e escrita, o XIndex garante que a leitura sempre alcança um resultado consistente quando executado concorrente à uma escrita utilizando bloqueios e versionamento no data_array e no buf. Com isso, o get tentará ler um valor do vetor e realizar uma captura do número da versão antes da leitura. Quando o valor é obtido, é verificado se o bloqueio ainda existe e se o número da versão atual corresponde ao número da captura. Caso haja um número errado de versão, o get repete o procedimento até uma validação bem sucedida, fazendo com que resultados inconsistentes ou obsoletos sejam descartados. Caso a leitura do data_array seja impossibilitada, o buf será lido [Tang et al., 2020].

Sendo assim, esse é um modelo de Learned Index que consegue um bom desempenho em uma abordagem multicore trazendo a Compactação em Duas Fases junto com algumas abordagens clássicas de concorrência.

Saindo de abordagens de software e buscando uma diretriz de hardware, Zhong et al (2022) traz o modelo de learned index apresentado anteriormente (PGM) em um enfoque de execução em GPU, diferindo da tradicional que realiza suas tratativas em CPU. Uma das desvantagens amplamente citadas em diversos trabalhos sobre Learned Index e reforçada nos tópicos anteriores é a limitação do poder de processamento computacional da CPU. Porém, as Unidades de Processamento Gráfico (GPU) e as Unidades de Processamento Tensorial (TPU) se mostraram indispensáveis na área de pesquisa em Machine Learning e Deep Learning, porém, mesmo diante desse padrão, os LI acabam por rodar apenas em CPU e aderem modelos simples como regressão linear ou redes neurais de duas camadas.

Enquanto a arquitetura da CPU é composta por alguns núcleos com cache para lidar com uma vasta série de tarefas, a GPU é composta por centenas de núcleos que operam apenas com tarefas simples. Por outro lado, o Learned Index substitui a busca tradicional por uma formada por computação de modelos de aprendizado de máquina, o que se encaixa na arquitetura da GPU. Por mais que exista esse nexo entre processamento em GPU e LI, existem algumas dificuldades encontradas na junção de ambos: a latência da transferência de dados entre memória principal e recuperação dos resultados da memória da GPU é alta; a perda de desempenho ocasionada por sincronização de operações entre CPU e GPU pode ser considerável; a arquitetura do LI precisa se adaptar ao espaço de memória hierárquico da GPU (ZHONG et al., 2022).

Portanto, a proposta do PGM-GPU é ser armazenado totalmente na memória da GPU, evitando problemas de desempenho causados por sincronização entre CPU e GPU. Antes de iniciar as consultas, um lote de chaves de busca é coletado na CPU e, caso as solicitações não sejam concorrentes, as chaves são coletadas dentro de um intervalo de tempo caracterizado por “lote”. Como as consultas de entrada são realizadas na memória principal, o processo do PGM-GPU é iniciado com a trans ferência destas para a memória da GPU [Zhong et al., 2022].

O GPU-PGM é beneficiado com a capacidade de execuções paralelas da GPU, onde é atribuído consultas aos kernels para execuções e cada um é responsável por iniciar threads que realizam as atividades de consultas, permitindo que várias operações sejam executadas simultaneamente. Quando *threads* do mesmo kernel operam de forma coordenada e sincronizada, eles acessam o PGM armazenado na memória global da GPU de maneira ordenada e executam as mesmas instruções de consulta. Ao finalizar todas as operações, os *threads* escrevem os resultados nas localizações de memória designadas simultaneamente. Ao finalizar as operações de busca, as posições exatas dos resultados das consultas são transferidas da GPU para a memória do sistema e a CPU utiliza as posições recebidas para acessar diretamente os registros específicos na base de dados. Ou seja, toda a tarefa intensiva de busca é realizada pela GPU e a CPU finaliza a operação acessando os dados específicos [Zhong et al., 2022]. Na Figura 4.16 há um esquema visual do funcionamento descrito acima.

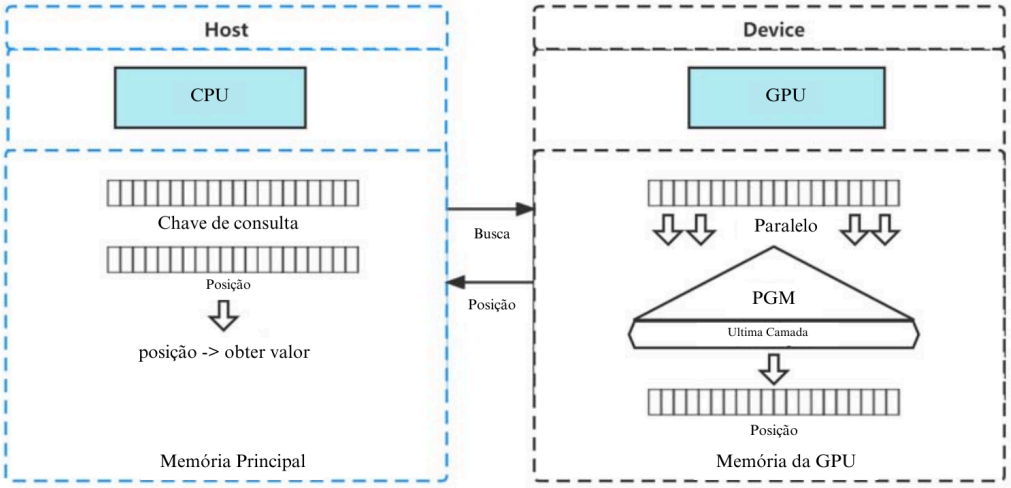


Figura 4.16. Funcionamento entre CPU (esquerda) e GPU (direita). Fonte: Traduzida de Zhong et al., 2022

Para montar o índice, a primeira etapa é a criação de um modelo linear por partes, onde o erro não exceda um limite pré-estabelecido. Com isso, o algoritmo garante que o modelo linear por partes seja ótimo. A principal ideia do algoritmo é reduzir o problema de construir o fecho convexo do conjunto de pontos (conjunto {key_i, $pos(key_i)$} crescido para i = 0, …, n-1. Enquanto o fecho convexo puder ser fechado em um retângulo de altura não maior que 2, o índice i é incrementado e o conjunto estendido. Quando o retângulo que envolve o casco é maior que 2, a construção é interrompida e um segmento é determinado abraçando a linha que divide o retângulo em duas metades de tamanho iguais. O algoritmo é iterado a partir dos pontos de entrada restantes. Ao finalizar essa etapa, o próximo passo é construir a estrutura de índice hierárquica, o qual inicia com o modelo construído sobre todo o vetor de entrada e a primeira chave do vetor coberta por cada segmento é extraída. Agora um novo modelo linear é construído por partes sobre esse conjunto reduzido de chaves. Todo esse processo continua de forma recursiva até que o modelo final consista em apenas um segmento [Zhong et al., 2022]. A estrutura do índice PGM pode ser vista na Figura 4.17.

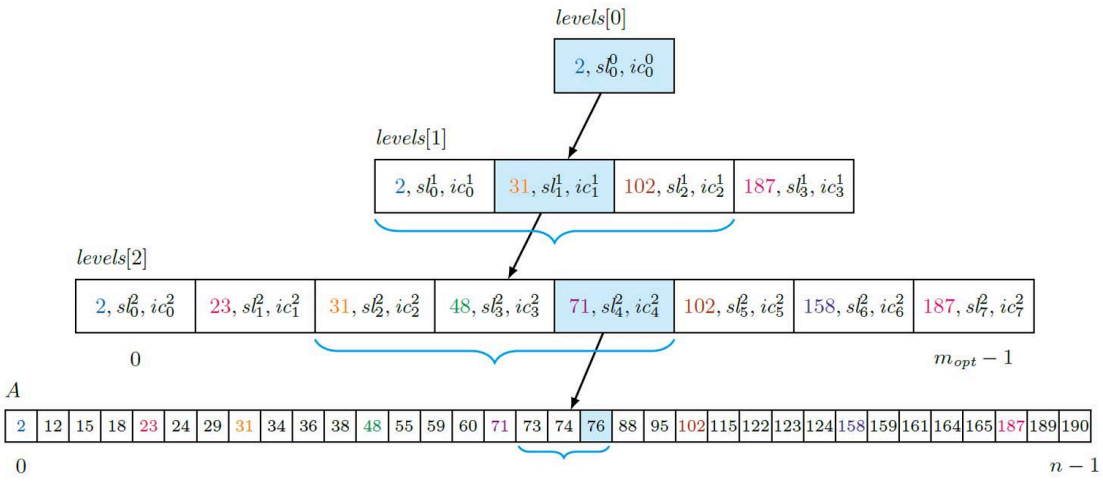


Figura 4.17. Estrutura do índice PGM. Fonte: [Zhong et al., 2022]

Já que o algoritmo de construção do PGM-Index necessita percorrer todo o conjunto de dados e não é possível realizar otimização paralela, o índice é construído na CPU e em seguida a estrutura de índice hierárquica é organizada como um array compacto e o deslocamento por nível é registrado. Finalizado esse processo, o índice é transferido para a GPU e o GPU-PGM é obtido.

Na etapa de consulta, o lote de chaves de entrada é enviado para a GPU e distribuído para um número designado de threads. Durante a operação é necessário verificar o número de threads no multiprocessador de streaming na GPU quando as threads são distribuídas, uma vez que a execução sincronizada é a maneira mais eficiente de execução na GPU, ou seja, threads do mesmo bloco executam as mesmas instruções de consulta e acessam a memória global de maneira síncrona e alinhada.

Quando cada thread recebe a chave de consulta correspondente, a busca funciona estimando a posição da chave e em cada nível é usado o segmento referente ao nó visitado para fazer essa estimativa entre as chaves do nível inferior. A posição real é encontrada por uma busca binária de tamanho dois que tem como centro a posição estimada. Portanto, como cada chave do próximo nível é a primeira chave coberta por um segmento, o próximo segmento que será consultado é encontrado e esse processo é repetido até que o último nível seja alcançado. Tendo os resultados, esses são escritos na localização especificada e são transferidos de volta para a memória principal.

4.3. Tendências, desafios de pesquisa e aplicações para a saúde

Desde sua concepção, o LI apresenta suas limitações como principais lacunas para investigação, desenvolvimento e avanço de suas soluções. Apesar de ser tema assíduo nas principais conferências sobre gestão de dados, big data e banco de dados, são inúmeras as oportunidades de evolução do conceito e suas aplicações. Neste minicurso, apresentaremos estes desafios e oportunidades em duas categorias: i) a de pesquisa básica, que inclui esforços para avanços teóricos, de assimilação de estrutura de dados inovadoras e escalabilidade; e ii) a de aplicação, que explora as otimizações já demonstradas empiricamente das diversas estruturas de LI já propostas nas outras áreas do conhecimento. A seguir, focaremos nossa apresentação em pesquisa aplicada, principalmente sob a ótica da Computação Aplicada à Saúde. É relevante mencionar, contudo, que diversos problemas de pesquisa relacionados com a gestão de dados desta comunidade ainda demandam avanços teóricos e um acúmulo mais robusto de evidências dos trabalhos de LI presentes na literatura.

4.3.1. Gestão de Big Data

Apesar dos vários artigos acrescentando evidências empíricas da superioridade dos modelos habilitados por ML para suportar grandes volumes de dados. Ainda existe uma lacuna evidente para a concepção de modelos de LI capazes de suportar a gestão de *big data*. Seja pela maturidade das estruturas de índice ou pela disponibilidade para o uso pelo público amplo, os modelos de LI escaláveis através de arquiteturas *multicore*, GPU

ou TPU ainda não são capazes de suprir a demanda da comunidade de gestão de big data. Por outro lado, estratégias de construção, uso e atualização de índices quando estes já não cabem em memória estão restritas a grandes provedores de computação em nuvem.

4.3.2. Dados multimodais

Do ponto de vista de pesquisa básica, especificamente no que diz respeito ao domínio de dados, ainda há uma carência muito grande de modelos existentes de LI capazes de lidar com o volume e a variedade das coleções de dados dos sistemas de informação em saúde. Por exemplo, a variedade pode ser definida através de uma determinada tipologia de dados, que estabelece escalas de mensuração e as operações matemáticas adequadas para cada uma delas. No contexto do Big Data, contudo, esta definição se mostra limitada. A variedade das variáveis numa coleção de dados pode ser avaliada também pelas modalidades presentes ou pela sua orientação de dependência (ex.: séries temporais).

Os dados que estão relacionadas à saúde do paciente podem apresentar diversas modalidades, a exemplificação na Figura <> mostra como um paciente pode estar associado aos diferentes tipos de dados, como os dados de imagens, série temporal ou sinal, dados genéticos, notas clínicas e informações laboratoriais e demográfica [Jensen et. al., 2012].

Ao agregar informações de diversas fontes e modalidades temos as bases de dados multimodais. Uma abordagem integrada das informações oferece uma visão mais abrangente dos pacientes, permitindo uma compreensão mais profunda dos casos em análise, bem como, melhores prognósticos e intervenções mais personalizadas [Acosta et. al., 2022]. Por exemplo, considere um sistema de diagnóstico, que recebe dados de registros médicos enriquecido de dados de imagem. Essa diversidade de informações permite que o sistema ofereça diagnósticos mais precisos com *insights* mais profundos.

A literatura apresenta trabalhos que utilizam conjunto de dados multimodais com objetivo de identificar e diagnosticar doenças. Nie et. al., 2017 propôs o uso de estruturas de deep learning para extrair características de imagens cerebrais multimodais (ou seja, ressonância magnética T1, fMRI e DTI) de pacientes com tumor cerebral. Esses dados foram utilizados juntamente com as principais características clínicas dos pacientes com o objetivo de prever se o paciente terá um tempo de sobrevida global longo ou curto, os experimentos apresentaram uma precisão de até 89,9%.

Pillai et. al., 2017, combinam características extraídas de imagens cerebrais com dados do líquido cefalorraquidiano para distinguir pacientes com doença de Alzheimer de indivíduos saudáveis. El-Sappagh et. al., 2020 buscam detectar a progressão da doença Alzheimer utilizando dados de diferentes modalidades, para isso, é proposta uma arquitetura de deep learning multimodal multitarefa.

A utilização de dados multimodais pode aprimorar a detecção precoce da mortalidade e elevar os índices de sobrevivência em pacientes com sepse na unidade de terapia intensiva (UTI), como evidenciado por Shin et al. (2021). Esse estudo integra informações demográficas, medidas fisiológicas e anotações clínicas para aplicar

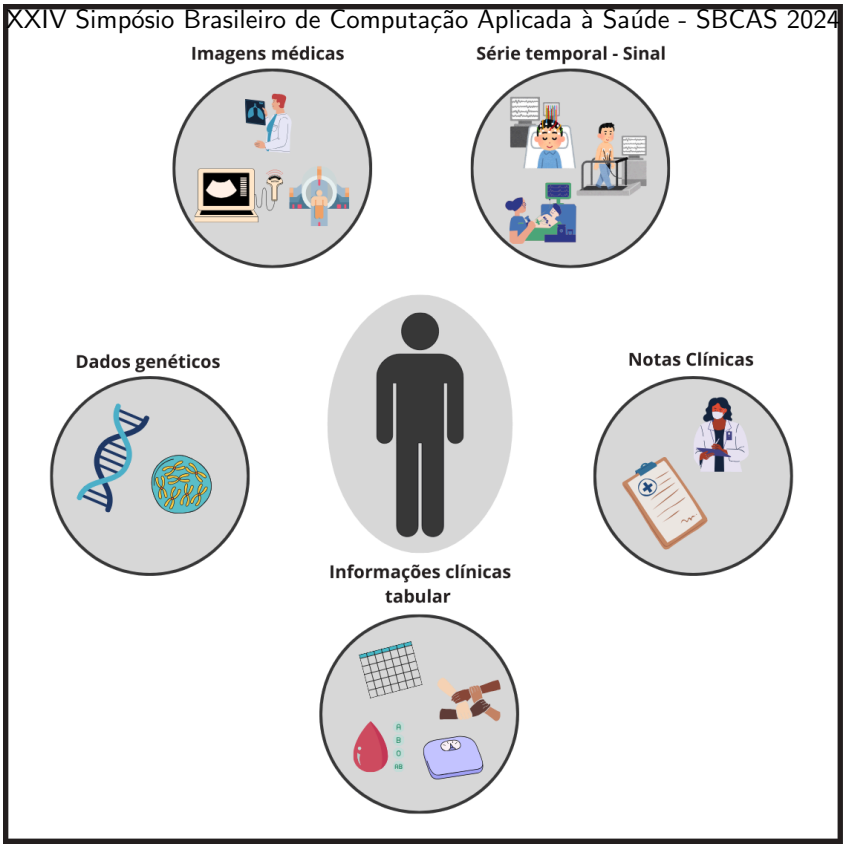


Figura 4.18. Exemplificação de dados multimodais em saúde

modelos de aprendizado de máquina visando prever o risco de mortalidade hospitalar nesses pacientes. Por fim, o estudo sugere que a integração de características clínicas estruturadas e não estruturadas pode ser efetivamente aplicada para auxiliar os médicos na identificação do risco de mortalidade em pacientes com sepse na UTI.

A fim de coletar e converter dados em informações e apoiar os profissionais na gestão de saúde, seja na esfera pública ou privada, existem os Sistemas de Informação em Saúde (SIS), os quais alimentam as bases de dados de saúde. No Brasil, os principais sistemas são Sistema do Cadastro Nacional de Estabelecimentos de Saúde (SCNES), Sistema de Informações Ambulatoriais (SAI/SUS), Sistema de Informações Hospitalares (SIH/SUS), Sistema Nacional de Agravo de Notificação (SINAN), Sistema de Informações sobre Nascidos Vivos (SINASC) e Sistema de Informação sobre Mortalidade (SIM) [Saraiva et al., 2021]. Esses dados se constituem como dados administrativos governamentais, recolhidas pelo serviço público como parte de seus processos contínuos e cotidianos.

4.3.2. Índices compostos ou multivariados

Índices multidimensionais consiste em estruturas de LI capazes de aprender uma mistura de CDFs ou um particionamento uniforme de um conjunto de dados com mais de uma dimensão, permitindo mimetizar a indexação multicoluna implementada em SGBDs tradicionais [Nathan et al., 2023]. Apesar dos diversos avanços nesta área, várias lacunas ainda devem ser exploradas nos próximos anos, principalmente para endereçar a ordenação monotônica que é desafiada quando se usa modelos de codificação das dimensões num único espaço dimensional, a escolha adequada de modelos de ML, concorrência e *benchmarking* [Al-Mamun et al., 2024]

4.3.1. Aplicações e oportunidades

Diversas iniciativas de gestão de dados em saúde incorporam modelos de indexação para garantir eficiência e escalabilidade das suas aplicações, como se pode observar no Modelo de Administração de Dados (MAD) do DATASUS e do Plano Diretor de Tecnologia de Informação (PDTI) do Ministério da Saúde. Várias destas soluções são proprietárias e oferecem pouca transparência sobre seus resultados ou confiabilidade de suas implementações. Neste sentido, o domínio de novas técnicas de indexação, especialmente pela comunidade de Computação Aplicada à Saúde, tem o potencial de promover uma adoção por soluções mais adequadas para as distribuições de dados existentes, ao passo que favorece a soberania de dados do país.

Aplicações de estruturas de índices habilitados por aprendizado de máquina na catalogação e curadoria de dados pode permitir uma maior maturidade na adoção dos princípios FAIR, perseguidos por centros de dados e pesquisa. Métodos de indexação estão diretamente ligados à melhoria de propriedades dos dados localizáveis (*Findable*), acessíveis (*Accessible*), interoperáveis (*Interoperable*) e reutilizáveis (*Reusable*). Por exemplo, serviços que suportam o modelos FAIR, como o SolR e o Dataverse, utilizam diversas técnicas avançadas de indexação para catalogar e encontrar *datasets*.

A integração de grandes volumes de dados é possível a partir da aplicação de técnicas poderosas de indexação. Por exemplo, em seu trabalho, Barbosa et al. (2020) argumenta que a escalabilidade e acurácia do CIDACS-RL — ferramenta de *record linkage* nacional responsável pela criação da Coorte de 100 Milhões de brasileiros — foi possível graças ao uso do Apache Lucene ou do Elasticsearch, duas soluções de indexação adequadas para sistemas de dados orientados a documentos (DODBMS). Contudo, novas técnicas de indexação aplicadas a ferramentas como estas podem garantir melhores resultados de desempenho, acurácia e mitigação de viéses.

Referências

Acosta, J. N., Falcone, G. J., Rajpurkar, P., and Topol, E. J. (2022) “Multimodal biomedical AI” Nature Medicine, 28(9), 1773-1784.

Al-Mamun, Abdullah, et al. "A Survey of Learned Indexes for the Multi-dimensional Space." arXiv preprint arXiv:2403.06456 (2024).

Amato, Giuseppe, and Pasquale Savino. "Approximate similarity search in metric spaces using inverted files." 3rd International ICST Conference on Scalable Information Systems. 2010.

Armenatzoglou, Nikos, et al. "Amazon Redshift re-invented." Proceedings of the 2022 International Conference on Management of Data. 2022.

Banker, Kyle, et al. MongoDB in action: covers MongoDB version 3.0. Simon and Schuster, 2016.

Barbosa, G. C., Ali, M. S., Araujo, B., Reis, S., Sena, S., Ichihara, M. Y., ... & 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, 1-13.

Barreto, M. L., Ichihara, M. Y., Almeida, B. D. A., Barreto, M. E., Cabral, L., Fiaccone, R. L., ... 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).

Campero Durand, Gabriel, et al. "Automated vertical partitioning with deep reinforcement learning." New Trends in Databases and Information Systems: ADBIS 2019 Short Papers, Workshops BBIGAP, QAUCA, SemBDM, SIMPDA, M2P, MADEISD, and Doctoral Consortium, Bled, Slovenia, September 8–11, 2019, Proceedings 23. Springer International Publishing, 2019.

Dama, International. DAMA-DMBOK: data management body of knowledge. Technics Publications, LLC, 2017.

Dasgupta, Sanjoy, and Yoav Freund. "Random projection trees and low dimensional manifolds." Proceedings of the fortieth annual ACM symposium on Theory of computing. 2008.

Datar, Mayur, et al. "Locality-sensitive hashing scheme based on p-stable distributions." Proceedings of the twentieth annual Symposium on Computational Geometry. 2004.

DATASUS. (2024) "Orientações para Construção de Modelo de Dados com Qualidade". Coordenação de Gestão de Banco de Dados. Disponível em: https://ced.saude.gov.br/arquivo/download?path=%2Fuploads%2F&file=daaed-cartilha-ad-20180815140452.pdf. Acesso em: 08 de março de 2024.

Ding, J., Marcus, R., Kipf, A., Nathan, V., Nrusimha, A., Vaidya, K., van Renen, A. and Kraska, T. (2022). “Sagedb: An instance-optimized data analytics system” Proceedings of the VLDB Endowment, v. 15, n. 13, p. 4062-4078.

Ding, J., Nathan, V., Alizadeh, M., and Kraska, T. (2020). “Tsunami: a learned multi-dimensional index for correlated data and skewed workloads” Proceedings of the VLDB Endowment, 14(2), 74-86.

Ding, Jialin, et al. "ALEX: an updatable adaptive learned index." Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 2020.

Elmasri, R., & Navathe, S. B. (2016). “Fundamentals of Database Systems” 7th ed.

El-Sappagh, S., Abuhmed, T., Islam, S. R., and Kwak, K. S. (2020). “Multimodal multitask deep learning model for Alzheimer’s disease progression detection based on time series data”. Neurocomputing, 412, 197-215.

Eryurek, Evren, et al. Data governance: The definitive guide. " O'Reilly Media, Inc.", 2021.

Fanggohans, Dean. Defio: Instance-Optimized Fusion of AWS Database Services. Diss. Massachusetts Institute of Technology, 2023.

Ferragina, Paolo, and Giorgio Vinciguerra. "The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds." Proceedings of the VLDB Endowment 13.8 (2020): 1162-1175.

Ferreira, J. E. D. S. M., Oliveira, L. R. D., Marques, W. S., Lima, T. S. D., Barbosa, E. D. S., Castro, R. R. D., and Guimarães, J. M. X. (2020). “Sistemas de Informação em Saúde no apoio à gestão da Atenção Primária à Saúde: revisão integrativa” RECIIS - Revista Eletrônica de Comunicação, Informação e Inovação em Saúde, Rio de Janeiro, v. 14, n. 4, p. 970-982.

Fritchey, Grant. "Automated Tuning in Azure and SQL Server." SQL Server 2022 Query Performance Tuning: Troubleshoot and Optimize Query Performance. Berkeley, CA: Apress, 2022. 657-675.

Gani, A. et al. A survey on indexing techniques for big data: Taxonomy and performance evaluation. Knowledge and Information Systems, v. 46, 04 2015.

Graefe, G. (2010). A survey of B-tree locking techniques. ACM Transactions on Database Systems, 35(3), Article 16. DOI: 10.1145/1806907.1806908.

Gray, J. (2007) “eScience -- A Transformed Scientific Method” eScience Talk at NRC-CSTB meeting, 11 January 2007. Apresentação do Power Point. Disponível em: https://jimgray.azurewebsites.net/talks/NRC-CSTB_eScience.ppt. Acesso em: 21 de dez. 2023.

Hadian, A., Ghaffari, B., Wang, T., and Heinis, T. (2023). “COAX: Correlation-Aware Indexing” In 2023 IEEE 39th International Conference on Data Engineering Workshops (ICDEW) (pp. 55-59). IEEE.

Harwood, Ben, and Tom Drummond. "Fanng: Fast approximate nearest neighbor graphs." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.

Hussam Abu-Libdeh, Deniz Altinbüken, Alex Beutel, Ed H. Chi, Lyric Doshi, Tim Kraska, Xiaozhou, Li, Andy Ly, and Christopher Olston. 2020. Learned Indexes for a Google-scale Disk-based Database. arxiv: 2012.12501

Hey, T., Tansley, S., and Tolle, K. M. (2009). “Jim Gray on eScience: a transformed scientific method” The Fourth Paradigm: Data-Intensive Scientific

Discovery. Redmond, Washington: Microsoft Research. 284pp. ISBN978-0982544204.

Jensen, P. B., Jensen, L. J., and Brunak, S. (2012). Mining electronic health records: towards better research applications and clinical care. Nature Reviews Genetics, 13(6), 395-405.

Junior, E. P. P., Normando, P., Flores-Ortiz, R., Afzal, M. U., Jamil, M. A., Bertolin, S. F., ... & Khalid, S. (2023). Integrating real-world data from Brazil and Pakistan into the OMOP common data model and standardized health analytics framework to characterize COVID-19 in the Global South. Journal of the American Medical Informatics Association, 30(4), 643-655.

Kaufman, Leonard, and Peter J. Rousseeuw. Finding groups in data: an introduction to cluster analysis. John Wiley & Sons, 2009.

Kaur, T. (2018). Learned Index Structures: Practical Implementations and Future Directions.

Kleppmann, M. (2017). Designing Data-Intensive Applications. Beijing: O'Reilly. ISBN: 978-1-4493-7332-0.

Kraska, T., Beutel, A., Chi, E. H., Dean, J., & Polyzotis, N. (2018, May). The case for learned index structures. In Proceedings of the 2018 international conference on management of data (pp. 489-504).

Kraska, Tim. "Towards instance-optimized data systems." Proceedings of the VLDB Endowment 14.12 (2021).

Kipf, Andreas, et al. "RadixSpline: a single-pass learned index." Proceedings of the third international workshop on exploiting artificial intelligence techniques for data management. 2020.

Kukreja, Sanjay, et al. "Vector Databases and Vector Embeddings-Review." 2023 International Workshop on Artificial Intelligence and Image Processing (IWAIIP). IEEE, 2023.

Li, Guoliang, Xuanhe Zhou, and Lei Cao. "Machine learning for databases." Proceedings of the First International Conference on AI-ML Systems. 2021.

Li, G., Zhou, X., & Cao, L. (2021). AI Meets Database: AI4DB and DB4AI. Proceedings of the 2021 International Conference on Management of Data. doi:10.1145/3448016.3457542

Ma, Chaohong, et al. "FILM: a fully learned index for larger-than-memory databases." Proceedings of the VLDB Endowment 16.3 (2022): 561-573.

Ma, Lin, et al. "MB2: decomposed behavior modeling for self-driving database management systems." Proceedings of the 2021 International Conference on Management of Data. 2021.

Mao, Hongzi, et al. "Resource management with deep reinforcement learning." Proceedings of the 15th ACM workshop on hot topics in networks. 2016.

Marcus, Ryan, et al. "Neo: A learned query optimizer." Proceedings of the VLDB Endowment 12.11 (2019).

Magalhães, M. N. “Probabilidade e variáveis aleatórias” São Paulo: IME-USP, 2004.

Maltry, M., and Dittrich, J. (2021). “A critical analysis of recursive model indexes” arXiv preprint arXiv:2106.16166.

Mikhaylov, Artem, et al. "Learned Query Optimizers: Evaluation and Improvement." IEEE Access 10 (2022): 75205-75218.

Ministério da Saúde, PORTARIA Nº 4.279, DE 30 DE DEZEMBRO DE 2010. Estabelece diretrizes para a organização da Rede de Atenção à Saúde no âmbito do Sistema Único de Saúde (SUS). Gabinete do Ministro. Brasília, DF.

Ministério da Saúde. PORTARIA GM/MS Nº 1.102, DE 13 DE MAIO DE 2022. Dispõe sobre a inclusão da covid-19 na Lista Nacional de Notificação Compulsória de doenças, agravos e eventos de saúde pública e Altera o Anexo 1 do Anexo V à Portaria de Consolidação GM/MS nº 4, de 28 de setembro de 2017. Diário Oficial da União, Brasília, DF, 16 maio 2022.

Nathan, Vikram et al. Learning Multi-Dimensional Indexes. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. New York, NY, USA: Association for Computing Machinery, 2020. p. 985–1000.

Nie, D., Zhang, H., Adeli, E., Liu, L., and Shen, D. (2016). “3D deep learning for multi-modal imaging-guided survival time prediction of brain tumor patients” Medical Image Computing and Computer-Assisted Intervention–MICCAI 2016: 19th International Conference, Athens, Greece, October 17-21, 2016, Proceedings, Part II 19 (pp. 212-220). Springer International Publishing.

Liu, Jiesong, et al. "G-Learned Index: Enabling Efficient Learned Index on GPU." IEEE Transactions on Parallel and Distributed Systems (2024)

LAZARIDIS, Michalis et al. Multimedia search and retrieval using multimodal annotation propagation and indexing techniques. Signal Processing: Image Communication, v. 28, n. 4, p. 351-367, 2013.

Paixao, E. S., Cardim, L. L., Falcao, I. R., Ortelan, N., Silva, N. D. J., Rocha, A. D. S., ... & Teixeira, M. G. (2021). “Cohort profile: Centro de Integração de Dados E Conhecimentos para Saúde (CIDACS) birth cohort” International journal of epidemiology, 50(1), 37-38.

Pillai, P. S., Leong, T. Y., and Alzheimer's Disease Neuroimaging Initiative. (2015). “Fusing heterogeneous data for Alzheimer's disease classification” MEDINFO 2015: eHealth-enabled Health (pp. 731-735). IOS Press.

Reis, Joe, and Matt Housley. Fundamentals of Data Engineering. " O'Reilly Media, Inc.", 2022.

Ross, S. (2013), Simulation (Fifth Edition) - Chapter 2 - Elements of Probability, Pages 5-38.

Ross, S. (2014), Introduction to probability models. Academic press.

Sandu Popa, Iulian, et al. "Indexing in-network trajectory flows." The VLDB Journal 20 (2011): 643-669.

Saraiva, L. I. M., Ramos, F. A. S., dos Santos, G. F., and Vetorazo, J. V. P. (2021). “Sistemas de informação em saúde, o instrumento de apoio à gestão do SUS: aplicabilidade e desafios” Revista Eletrônica Acervo Enfermagem, 9, e6418-e6418.

Shin, J., Li, Y., and Luo, Y. (2021). “Early prediction of mortality in critical care setting in sepsis patients using structured features and unstructured clinical notes” In 2021 IEEE International Conference on Bioinformatics and Biomedicine (BIBM) (pp. 2885-2890). IEEE.

Silberschatz, Abraham, Henry F. Korth, and S. Sudarshan (2020). “Database System Concepts” 7th ed.

TANG, Chuzhe; WANG, Youyun; DONG, Zhiyuan; HU, Gansen; WANG, Zhaoguo; WANG, Minjie; CHEN, Haibo. XIndex: a scalable learned index for multicore data storage. In: ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 25., 2020, San Diego, California. Proceedings [...]. New York, NY, USA: Association for Computing Machinery, 2020. p. 308–320. Disponível em: https://doi.org/10.1145/3332466.3374547

Wang, Y., Tang, C., Wang, Z., & Chen, H. (2020). “SIndex: a scalable learned index for string keys” In Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems (pp. 17-24).

Wei, Ling-Yin, et al. "Indexing spatial data in cloud data management." Pervasive and Mobile Computing 15 (2014): 48-61.

Witten, Ian H., and Eibe Frank. "Data mining: practical machine learning tools and techniques with Java implementations." Acm Sigmod Record 31.1 (2002): 76-77.

Wong, D. W. (2001). Location‐Specific Cumulative Distribution Function (LSCDF): An Alternative to Spatial Correlation Analysis. Geographical Analysis, 33(1), 76-93.

Wu, K., Shoshani, A., & Stockinger, K. (2010). Analyses of Multi-Level and Multi-Component Compressed Bitmap Indexes. ACM Transactions on Database Systems, 35(1), 1-52. DOI: 10.1145/1670243.1670245.

Zhao, Xinyang, Xuanhe Zhou, and Guoliang Li. "Automatic database knob tuning: a survey." IEEE Transactions on Knowledge and Data Engineering (2023).

Zhou, Xuanhe, et al. "Dbmind: A self-driving platform in opengauss." Proceedings of the VLDB Endowment 14.12 (2021): 2743-2746.

Zhong, X., Zhang, Y., Chen, Y., Li, C., and Xing, C. (2022). “Learned index on GPU” In 2022 IEEE 38th International Conference on Data Engineering Workshops (ICDEW) (pp. 117-122). IEEE.

