Ziawasch Abedjan
Mahdi Esmailoghli
Sainyam Galhotra

Data Discovery in Data Lakes

Springer

Data Discovery in Data Lakes

Ziawasch Abedjan • Mahdi Esmailoghli •
Sainyam Galhotra

Data Discovery in Data Lakes

Springer

Ziawasch Abedjan
Berlin Institute for the Foundations of Learning and Data (BIFOLD)
Technische Universität Berlin
Berlin, Germany

Mahdi Esmailoghli
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, Canada

Sainyam Galhotra
Department of Computer Science
Cornell University
Ithaca, NY, USA

ISBN 978-3-032-30821-4 ISBN 978-3-032-30822-1 (eBook)
https://doi.org/10.1007/978-3-032-30822-1

© The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Switzerland AG 2027

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

If disposing of this product, please recycle the paper.

Preface

At the time of writing, data discovery had emerged as a highly active research area, with entire sessions at leading database conferences, such as SIGMOD, VLDB, and ICDE, devoted to new discovery methods, indexing techniques, and benchmarks. At the same time, we observed increasing fragmentation in terminology and communication, which made it progressively harder to compare results, relate contributions to one another, and place new work into a coherent broader context. This motivated us to collect and organize the relevant body of research known to us and to develop a tutorial on data discovery for VLDB 2025, which in turn led to the invitation to write this book.

This book builds on that effort by offering a deeper and more systematic treatment of the field. Our goal is not only to survey the large and rapidly growing body of research in data discovery but also to contextualize it, discuss its assumptions and limitations critically, and highlight directions that we believe are especially promising for future work. As the field is continuously growing, we acknowledge that we cannot claim to have covered all research in this area.

Berlin, Germany — Ziawasch Abedjan
Waterloo, ON, Canada — Mahdi Esmailoghli
Ithaca, NY, USA — Sainyam Galhotra
April 2026

Competing Interests The authors have no competing interests to declare that are relevant to the content of this manuscript.

Contents

1 Data Discovery: Motivation and Challenges 1
1.1 Introduction and Overview 1
1.2 Data Lakes: Background and History 3
1.3 Challenge: Ad Hoc Data Integration 5
1.4 Case Studies 6
1.4.1 Government Data Portals 7
1.4.2 Data Marketplaces 8
1.4.3 Pollution Explanation 13
1.5 Book Outline 15
References 15

2 Discovery Settings 19
2.1 Introduction 19
2.2 Data Lake 20
2.2.1 Centralized Setting 22
2.2.2 Federated Setting 23
2.2.3 Hybrid Setting 24
2.3 Privacy and Governance Constraints 25
2.4 Multimodal Settings 26
2.5 Temporal Dimension of Discovery 26
2.6 Summary of Discovery Settings 28
References 31

3 Discovery Operations 33
3.1 Introduction and Overview 33
3.2 Keyword Search 34
3.2.1 Keyword Search in Documents and Tables 34
3.2.2 Keyword Search Prototypes 35
3.2.3 Further Impact 37
3.3 Union Search 37
3.3.1 Semantics of Union Search 38
3.3.2 Unionability of Columns 39

3.3.3 Union Search Prototypes 41
3.4 JOIN Search 42
3.4.1 Joinability 44
3.4.2 Utility Augmented Joinability Scores 46
3.5 Operation Sequences and Compositions 48
3.5.1 Operation Sequences and Grammars 48
3.5.2 Domain-Specific Languages for Data Discovery 52
3.6 Natural-Language Queries 53
3.6.1 Natural-Language Questions Over Large Repositories 54
3.6.2 Multi-table QA 55
3.7 Operations in Distributed Repositories 56
3.7.1 Discovery Operations in Federated Environments 56
3.8 Additional Data Discovery Constraints 59
3.8.1 Privacy-Constrained Data Discovery 59
3.8.2 Multimodal Data Discovery 60
3.9 Conclusion 62
References 63

4 Indexing Strategies for Data Discovery 67
4.1 Introduction 67
4.2 Value-Based Indexes 69
4.2.1 Inverted Index 70
4.2.2 Inverted Index for Multicolumn Retrieval 75
4.3 Column-Based Indexes 78
4.3.1 MinHash and Locality-Sensitive Hashing for Column Similarity 80
4.3.2 Beyond Minhash LSH 87
4.4 Indexes for Discovering Correlating Columns 89
4.4.1 Sketch-Based Correlation Indexes 90
4.4.2 Exact Correlation Discovery 93
4.4.3 Discussion 96
4.5 Row-Based Indexes 96
4.6 Table-Based Indexes 97
4.7 Graph-Based Indexes 98
4.7.1 Similarity-Based Navigation 98
4.7.2 Topic-Driven Navigation 98
4.7.3 Knowledge Graph-Enhanced Discovery 99
4.7.4 Discussion 99
4.8 Embedding-Based and Foundation Model Approaches 99
4.8.1 ANN Indexes over Dense Column Embeddings 100
4.8.2 Hybrid Retrieval: Combining Full-Text and Vector Indexes 101
4.8.3 LLM-Augmented Discovery Pipelines with Lightweight Indexing 102

4.8.4 Self-Supervised Representation Learning for Table Relationships 102
4.8.5 Foundation Models as Semantic Reasoners 103
4.9 Coresets for Discovery Indexes 104
4.9.1 Distribution-Aware Discovery and the Coreset Construction Problem 106
4.10 Conclusion 107
References 109

5 Data Discovery Systems 115
5.1 Introduction 116
5.2 The Demand for End-to-End Discovery Systems 116
5.2.1 Survey by Esmailoghli et al. (2023) 116
5.2.2 Survey by Hulsebos et al. (2024) 118
5.3 Academic Systems 120
5.3.1 AURUM 120
5.3.2 AUCTUS 122
5.3.3 BLEND 124
5.4 Industrial Discovery Tools 125
5.4.1 Enterprise Discovery Platforms 126
5.4.2 Marketplace Infrastructures 129
5.5 Conclusion 130
References 132

6 Benchmarking Data Discovery Systems 135
6.1 Challenges in Evaluation 135
6.2 Effectiveness in Data Discovery 136
6.2.1 Background: Keyword Search in Information Retrieval 137
6.2.2 World Assumptions 138
6.2.3 Adaptation of Evaluation Metrics for Tabular Data Discovery 138
6.2.4 Evaluating JOIN Discovery 139
6.2.5 Evaluating UNION Discovery 140
6.3 Available Benchmarks 140
6.3.1 Synthesized Lakes for CWA 140
6.3.2 Limitations of CWA Benchmarks 141
6.3.3 Utility-Driven Data Discovery for OWA 142
6.4 Efficiency in Data Discovery 142
6.4.1 Deployment Costs 143
6.4.2 Query Runtime 144
6.4.3 Ease of Use 145
6.5 Conclusion 145
References 147

7 Conclusions 151
7.1 Summary 151
7.2 Future Directions 152
7.2.1 Holistic Data Discovery Systems 152
7.2.2 Time and Lake Versioning 154
7.2.3 Decentralized Data Discovery 154
7.2.4 Multimodal Data Discovery 155
7.2.5 Agentic Data Discovery 156
References 157

Chapter 1
Data Discovery: Motivation and Challenges



Abstract Data lakes have emerged as a dominant paradigm for retaining heterogeneous data in its original form, but their flexibility comes with a fundamental usability challenge: They are typically underspecified, weakly documented, and difficult to exploit in an ad hoc manner. This chapter introduces data discovery as the set of methods and systems that make such repositories accessible by enabling users to search for relevant datasets without requiring detailed prior knowledge of schemas or contents. We motivate data discovery through the characteristics and historical evolution of data lakes, tracing their roots to the rise of big data infrastructures and the shift from schema-on-write to retain-first data management. Against this background, the chapter positions data discovery as a bridge between information retrieval and data integration: Like information retrieval, it addresses user-specified needs through search. Unlike traditional retrieval, it must identify datasets that are not only relevant but also usable for downstream tasks such as augmentation, joining, unioning, and exploratory analysis. We then outline the core architectural view of a vanilla data discovery system, where user queries, table artifacts, and discovery operations are supported by dedicated indexes and matching techniques. Finally, the chapter frames the scope of the book by highlighting the main technical themes that follow, including discovery settings, operations, indexing methods, end-to-end systems, and benchmarking. Overall, the chapter establishes data discovery as a central systems problem for turning raw, heterogeneous data lakes into practically usable analytical assets.

Keywords Data discovery · Data lakes · Market places · Open data

1.1 Introduction and Overview

Data lakes, whether intentionally created or emerging as a byproduct of organizational processes or open data initiatives, share a common challenge: They are often underspecified and underdocumented (Fernandez et al. 2016; Abedjan et al. 2015). Unlike curated data warehouses, where the schema and purpose are clearly defined, data lakes typically lack predefined objectives. Despite this, there is significant

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027

Z. Abedjan et al., *Data Discovery in Data Lakes*,
https://doi.org/10.1007/978-3-032-30822-1_1

interest in leveraging these repositories for augmenting in-use datasets or pure exploratory purposes. To support such applications, data discovery as a process becomes essential, enabling users to interact with the data lake not by relying on detailed schema knowledge, but by using search and matching techniques. By searching with user-provided keywords or table artifacts, users aim to uncover datasets that are relevant for their use cases.

Data discovery is currently an active field of research, as building scalable solutions to effectively capture user intent, such as augmenting a given dataset fragment, presents various challenges. A key issue is the need to index thousands to millions of tables for fast value lookups and efficient alignment to evaluate joinability (Galhotra et al. 2023; Esmailoghli et al. 2022; Zhu et al. 2019) and unionability (Nargesian et al. 2018) with the user (query) table. After identifying relevant tables, assessing the usefulness of the retrieved results remains a complex problem. Often, users rely on proxy metrics, such as the downstream machine learning performance of the augmented dataset, to assess its effectiveness (Galhotra et al. 2023; Santos et al. 2021; Esmailoghli et al. 2021; Esmailoghli and Abedjan 2020). Finally, in distributed data lakes, additional constraints, such as privacy regulations (Huang et al. 2023) and pricing structures, further complicate the process. In such cases, the usefulness and relevance of datasets must often be assessed based on available metadata and summaries, rather than direct access to the full datasets.

The emerging phenomena of data lakes and the necessity for data discovery naturally attracted broad attention from the data management research community. The sheer amount of novel algorithms, systems, and benchmarks lead to several efforts in systematic definition and categorization of the individual aspects of data discovery materialized as surveys and tutorials. Idreos et al. (2015) discuss efficient algorithms and systems for data exploration in the broader sense. Paton and Wu (2024) and Dong et al. (2021) cover various aspects of dataset search and navigation. Zhang and Balog (2019) discuss data discovery from the lens of information retrieval and give perspectives on Web table extraction. Nargesian et al. (2019) elevate the discussion toward discovery operations and metadata management for data lakes.

The Vanilla Data Discovery System

Figure 1.1 depicts a conceptual architecture of data discovery systems and its components for discovering relational artifacts. The user formulates a discovery task that typically aims at enhancing an existing table or table fragment through additional features, rows, or individual values. The formulation of the discovery task can either be the upload of a dataset whose relevant attributes for search have been marked or a more complex discovery query formulated around an input dataset fragment (Esmailoghli et al. 2025). A discovery task can range from exploratory, such as identifying all related tables, to more complex objectives, such as augmenting a dataset across multiple dimensions to improve its completeness and utility. Such tasks are then typically carried out on top of more basic operations, such as keyword search, join discovery, and union discovery. Such operations themselves

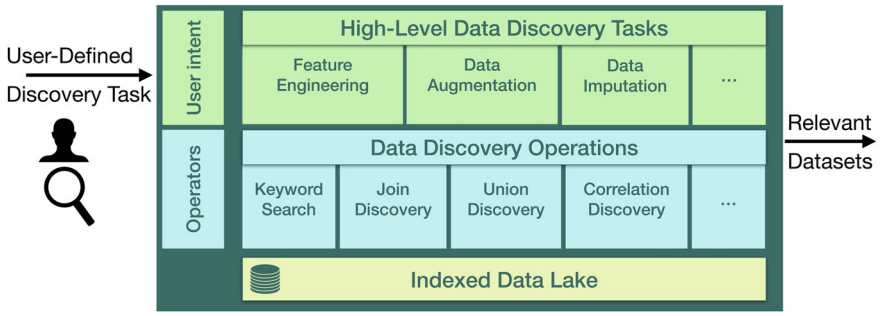


Fig. 1.1 Vanilla data discovery system architecture

are optimized retrieval algorithms tailored to the underlying dedicated indexes. The result of a query is a set of relevant datasets, potentially prepared for or already aligned with the input query dataset.

In this book, we have a strong focus on algorithmic details and how index structures and operations have evolved and can be categorized. We further project how these approaches relate and can extend to distributed scenarios and can be combined inside holistic systems. We cover recent advances and discuss the topic from the database management perspective.

Methodologically, this book is based on our previous study of data discovery use cases and our experience in building systems and algorithms that address some of its emergent challenges. Our goal is to give a structured overview of data discovery software components and dive into technical aspects that differentiate individual prototypes and approaches. A meta-goal is to bridge the language and perspective differences on what is counted as data discovery and how discovery systems in database research can be compared. While data discovery and data lakes encompass multimodal data structures, this book has a strong focus on covering recent advances in discovering relational tables. To understand the concepts that are introduced in this book, a solid background in database management and familiarity with machine learning and AI trends is recommended.

1.2 Data Lakes: Background and History

Hai et al. (2023) give a systematic timeline of how the term data lakes started to surface in the early 2010s and gradually boomed midway through the same decade. The advent of data lakes is closely tied to the rise of big data technologies, such as Hadoop,[1] Spark (Zaharia et al. 2012), and Flink (Alexandrov et al. 2014), in early 2010s and the resulting change in enterprise requirements. As organizations

[1] http://hadoop.apache.org/

began collecting high-volume, high-variety data such as logs, clickstreams, sensor data, and semi-structured records, the traditional data warehouse model increasingly struggled with long lead times and up-front modeling commitments, since data typically had to be transformed to fit a predefined structure before it could be stored and analyzed. In contrast, the big data stack made it practical to retain large amounts of data cheaply and process it at scale, shifting the emphasis from pre-modeling toward preserving raw data for unknown future questions. In this context, James Dixon, who was the CTO of Pentaho in 2010, popularized the term *data lake* to describe an alternative to "bottled" data marts: *Rather than packaging data immediately for a specific consumption pattern, data flows into a lake in a more natural state, from which different users can "examine, dive in, or take samples" depending on their needs* (Dixon 2010). Subsequent work has reinforced this framing by explicitly contrasting the warehouse's schema-on-write orientation with the lake's retain-first, interpret-later approach as a response to information loss and limited reuse under early transformation.

Over the course of time, data lakes are a natural organizational outcome of three forces arriving at the same time:

1. Data collection becoming ubiquitous through applications, logs, sensors, customer interactions
2. Storage becoming cheap enough to keep "everything" and
3. A belief that retained data will eventually yield business value, whether through analytics, reporting, personalization, forecasting, or new product capabilities

In practice, a data lake is often less a single deliberate project than an emergent pattern: Teams keep landing raw and semi-structured data in a central repository because it is fast, flexible, and feels future proof, even if the specific downstream use cases are not fully defined yet. That gravity of "store now, figure out later" makes data lakes both powerful and messy, especially as organizations scale and need governance, quality controls, and cost discipline.

The emergence of data lakes has created strong incentives for data processing and platform companies to productize the hard parts: ingesting diverse data, managing compute vs. storage economics, enabling SQL and machine learning workloads, enforcing security and governance, and supporting reliable pipelines without locking teams into fragile bespoke systems. Vendors such as Snowflake,[2] Databricks,[3] and AWS,[4] alongside others like Google Cloud[5] and Microsoft,[6] have built competing and complimentary approaches, such as cloud data warehouses and Lakehouse architectures, managed storage and query services, catalog/governance layers, and end-to-end analytics/AI platforms, because the market demand is

[2] https://www.snowflake.com/

[3] https://www.databricks.com/

[4] https://aws.amazon.com/

[5] https://cloud.google.com/solutions/data-lakehouse

[6] https://azure.microsoft.com/en-us/solutions/data-lake

Table 1.1 Representative definition statements for "data lake" in the literature

Source	Definition excerpt (verbatim)	Emphasis
Terrizzano et al. (2015)	"A centralized repository containing virtually inexhaustible amounts of raw (or minimally curated) data"	Centralization and accumulation of raw data
Nargesian et al. (2019)	"Current data lakes provide reliable storage for datasets ... along with a suite of tools for ... data discovery, extraction, cleaning, and integration"	Platform with operations for ad hoc integration
Sawadogo and Darmont (2021)	"A data lake is a large, raw data repository that stores and manages all company data bearing any format"	Enterprise scope; any format
Hai et al. (2023)	"Data lakes are repositories storing raw data in its original formats and providing a common access interface"	Original formats; unified access

consistent: Organizations want the optionality of a lake with the usability and performance of curated systems.

Table 1.1 contains several examples on how data lakes have been defined in the literature. While Terrizzano et al. (2015) put the focus on centralization, Nargesian et al. (2019) define data lakes as a combination of storage and processes. Sawadogo and Darmont (2021) highlight the relevant context of enterprise datasets, and Hai et al. (2023) emphasize the fact that data is provided in its raw and original format.

In this book, we adapt to a definition that connects the commonalities of these definitions. We consider a *data lake to be a repository that keeps heterogeneous data from many sources in its original form and defers imposing structure until use*. In this view, the distinguishing property is not a particular processing stack but rather the commitment to retain heterogeneous data "as-is" while relying on subsequent discovery, access, and governance capabilities to make it usable for analytics (Quix and Hai 2018; Hai et al. 2023; Sawadogo and Darmont 2021; Paton et al. 2024; Dixon 2010). We will give a more formal definition of data lakes in Chap. 2, where we talk about its fundamental settings.

1.3 Challenge: Ad Hoc Data Integration

So far, the aforementioned processing solutions by tech vendors serve exploratory analysis or well-curated data transformation workflows. As a result, the entirety of information is hardly accessible to a single individual without significant efforts in

exploration. A common problem that has so far been ignored is how to use the lake in an ad hoc manner. This is where data discovery as a discipline arises and attempts to build solutions that support not only the user who is aware of the content and schema inside a data lake but also those who only have implicit knowledge about the datasets stored.

What we mean with implicit knowledge is that the actual schema of tables and their constraints are not known, but the user might know that certain datasets might exist that cover an information need. At this point there is a strong initial resemblance of data discovery to information retrieval (IR).

In fact, information retrieval and data discovery address related but distinct problems. The goal of information retrieval is to satisfy an *information need* by retrieving relevant documents or passages, whereas data discovery targets an *information need or a dataset need*, i.e., helping users find data assets, such as tables, files, streams, and features, that can be analyzed or reused, not just read. Accordingly, information retrieval queries are typically expressed as unstructured *keywords*, while data discovery queries often combine *keywords with structured elements* such as schema terms, columns, tables, or quality constraints. Finally, information retrieval systems usually present a *ranked list of results* optimized for topical relevance, whereas data discovery systems commonly go beyond ranking to support *ad hoc integration*, through previewing, joining, sampling, imputing, and assembling candidate datasets to make them immediately usable in downstream analytics and decision-making.

With the goal for ad hoc data integration, data discovery faces several data integration related challenges that go beyond the performance challenges that are addressed by current vendors. The questions of relevance of discovered objects and their usefulness require heuristics-based solutions that rely on metrics, such as coverage and similarity. For each individual data discovery operation, such as Union or Join search, which we discuss in detail in Chap. 3 of this book, both exact matches of values and fuzzy matches are relevant. Further questions include how to holistically integrate such operations on top of unified data models, how to optimize data discovery tasks consisting of multiple operations holistically, and how to execute such operations in scenarios where data is distributed and access is subject to constraints.

1.4 Case Studies

To motivate the case for researching data discovery systems and operations, we provide three different examples of real-world scenarios where data discovery capabilities support end users and analysis. First we discuss the emergence of governmental open data frameworks and then the data market place landscape, and finally we conclude with a concrete example of ad hoc data integration for inference.

1.4.1 Government Data Portals

Open Government Data Portals (OGDPs) exist on multinational (EU) national (USA, UK, Canada, and Germany) and municipal (NYC) levels. They serve as open-data catalogs that are increasingly modeled as large, heterogeneous "open data lakes" of tables rather than as well-curated databases. For example, the US federal portal grew its indexed files by an order of magnitude since 2016. Further, a crawl of US/UK/Canada portals already contained 215,393 CSV[7] tables with millions of attributes, yielding a repository whose scale and publisher heterogeneity make a priori knowledge of available schemata unlikely for end users (Miller 2018).

- **Data.gov:** The US federal open-data portal[8] is an archetypal OGDP that foregrounds discovery and reuse at catalog scale: It presents itself as "the home of the U.S. Government's Open Data," explicitly emphasizing downstream reuse for research, application development, and visualization to inform decisions, drive innovation, and strengthen transparency. Operationally, its public-facing catalog is powered by CKAN[9] and exposes a Comprehensive Knowledge Archive Network-Application Programming Interface (CKAN-API). The platform documentation stresses that the CKAN end point is a *metadata* service—containing dataset descriptions and URLs rather than necessarily hosting the underlying data. So discovery often entails navigating from portal metadata to distributed agency systems. This architecture makes data.gov a representative case where data discovery quality depends on metadata completeness and consistency.
- **Data.europe.eu:** The European portal[10] aims to serve as a "single point of access to open data from European countries" and EU institutions, and it is explicitly engineered around semantic interoperability and cross-portal harvesting. A key technical feature is that portal metadata is stored as Resource Description Framework (RDF) and are queryable via a public SPARQL end point (Angles and Gutierrez 2008), aligning with the Data Catalogue vocabulary - Application Profile (DCAT-AP) profile that standardizes public-sector dataset descriptions across Europe. The design rationale and implementation details are documented in the EGOV 2019 paper by Kirstein et al., which describes the then European Data Portal as a central access point harvesting metadata from dozens of national providers and bridging conventional portal expectations regarding UI and API compatibility with native RDF storage and quality assessment (Kirstein et al. 2019). Conceptually, the European portal thereby illustrates a portal model where data discovery is supported not only through conventional catalog search but also

[7] comma-separated values

[8] data.gov

[9] https://ckan.org/

[10] data.europe.eu

through machine-actionable graph representations intended to enable federation, linkage, and quality monitoring at continental scale.

The dominant failure mode of search on the aforementioned portals is that meaningful relatedness is often expressed at the instance level, i.e., values and keywords, and via latent inter-table relationships rather than in reliable metadata. Accordingly, it is acknowledged that there is a "lack of consistent and reliable metadata" in open data and proposes interactive discovery of dataset linkages that can be navigated as joins, making "unrelated and unlinked dynamic collections" appear as a connected cloud that users can combine in real time (Zhu et al. 2017).

1.4.2 Data Marketplaces

Data marketplaces represent a distinct and commercially motivated instantiation of federated data discovery. Unlike government portals, where datasets are published for transparency and public reuse, data marketplaces are platforms where data is treated as a tradable commodity: Providers monetize datasets they have collected or curated, and buyers acquire them to enrich their own analytical pipelines. Figure 1.2 presents the key components of a data marketplace. At its core, every marketplace must solve a discovery problem: Given a buyer's need and a corpus of available datasets, how does the right data find the right buyer? The answer turns out to depend heavily on what the buyer knows about their need, what the seller is willing to expose before a transaction is committed, and what constraints (commercial, legal, or technical) govern the transfer. These factors vary dramatically across marketplace types. Below, we discuss examples of popular types of marketplaces.

1.4.2.1 Clean Rooms: Discovery Without Visibility

Enterprise data clean rooms are privacy-preserving environments in which two organizations collaborate over their combined data without either party seeing the

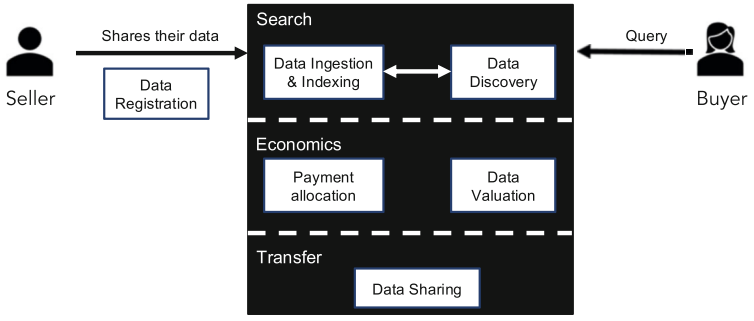


Fig. 1.2 Key components of a data marketplace

other's raw records. Popular clean rooms include Snowflake clean room,[11] AWS clean room,[12] Google BigQuery Analytics Hub,[13] and InfoSum.[14] As an example, Walmart may have a loyalty program and they collect in-store purchase data of their customers. Discover bank holds transaction and demographic data, and both want to measure whether advertising exposure drove purchases but PII regulations prohibit sharing raw customer records. A *clean room* mediates a join over anonymized keys, allowing both parties to compute sales lift and cross-channel attribution on the combined dataset.

From a discovery standpoint, clean rooms represent the most constrained setting: The buyer cannot inspect any of the seller's data before committing to collaboration. Discovery is reduced to catalog browsing and bilateral invitation, as the buyer must decide whether a potential partner holds relevant data based solely on the partner's identity, reputation, and a high-level schema description. The central question shifts from "does this dataset contain the values I need?" to "does this organization hold data about the same entities I care about?" This is a setting where traditional content-based discovery operations like join search, union search, and keyword search over values are unavailable and where the design of useful discovery mechanisms under zero-visibility constraints is an open problem. We return to privacy-constrained discovery in Chap. 2.

1.4.2.2 Labeling Markets: Discovery as Task Specification

Data labeling markets such as Scale AI[15] and Surge AI[16] invert the conventional discovery model. Rather than searching for data that already exists, the buyer specifies what data should exist, and the platform produces it. The company Waymo, for instance, generates millions of raw LiDAR frames from its autonomous vehicle fleet but needs structured 3D bounding boxes and semantic segmentation labels to train perception models. It submits task definitions and raw frames to Scale AI, whose labeler network and AI-assisted workflows produce the annotations to order.

This inversion is instructive for data discovery because it highlights a class of information need that retrieval-based discovery cannot satisfy: When the required data does not yet exist in any repository, no amount of indexing or search will find it. The discovery problem here is one of gap identification: recognizing that existing datasets do not cover the buyer's need and that new data must be created. Understanding when retrieval-based discovery is sufficient and when it must be

[11] https://docs.snowflake.com/en/user-guide/cleanrooms
[12] https://aws.amazon.com/clean-rooms/
[13] https://cloud.google.com/analytics-hub
[14] https://www.infosum.com/
[15] https://scale.com/
[16] https://surgehq.ai/

supplemented by data production is a recurring tension in practical data lake management.

1.4.2.3 Alternative Data Brokers

Alternative data brokers such as Thinknum,[17] Bloomberg,[18] and YipitData[19] curate datasets about companies, products, and markets, sourced from Web crawls, sensors, and proprietary feeds, and sell them to investors and analysts seeking informational advantages. Thinknum, for instance, continuously crawls hiring pages, app store rankings, and retail inventory listings, differences the data day-to-day, and sells cleaned change-signal feeds. A subscriber might monitor a competitor's engineering hiring rate as a proxy for product development pace or track retail pricing changes ahead of earnings announcements.

The discovery challenge here is particularly instructive: Buyers often do not know which datasets are relevant to a given hypothesis until they have examined the data, but the data is too commercially sensitive to expose fully before purchase. Platforms respond with curated thematic catalogs and team-based access workflows that allow analysts to explore related products before committing. This is a practical instance of the utility gap introduced in Chap. 2, the disconnect between what a buyer can learn from prepurchase metadata and what the data actually contains. Designing discovery mechanisms that close this gap without compromising seller interests is one of the central open problems in data marketplace research.

Dataset Search Engines: Metadata-Only Discovery at Web Scale Dataset search engines such as Google Dataset Search index metadata exposed by publishers using the Schema.org vocabulary and return links to dataset landing pages rather than the datasets themselves. Because no raw data is ever accessed, discovery operates entirely over declared descriptions and fields, making this the purest possible instance of metadata-only discovery.

The resulting discovery quality is often not ideal. Two datasets that share geographic identifiers and are therefore highly joinable will not be surfaced as related unless their metadata descriptions use compatible terminology. A dataset of European traffic counts and a dataset of European air quality measurements may share NUTS region codes as a natural join key, but no metadata-based search engine will surface the relationship if neither publisher mentioned it. Google Dataset Search's own designers identify metadata sparsity and inconsistency as the primary limitation of the system (Brickley et al. 2019). The system therefore defines the performance floor of data discovery, what is achievable with metadata alone, against which richer, content-aware discovery operations should be measured.

[17] https://www.thinknum.com/

[18] https://data.bloomberg.com/

[19] https://www.yipitdata.com/

Understanding where metadata-only discovery fails is as important for the field as understanding where it succeeds.

1.4.2.4 Model-as-Data: Discovery Through Task and Community Signals

Hugging Face[20] Datasets have redefined what it means to discover a dataset in the machine learning context. Rather than searching by content or schema, practitioners discover datasets through task tags, benchmark affiliations, and citation counts. A practitioner building a question-answering model searches for datasets that are tagged as question answering, filtered by language and license, and ranked by how frequently the dataset appears in published work.

This modality exposes a discovery mechanism that content-based systems largely ignore: social and task-structural signals. A dataset that is well-documented, benchmarked, and cited in influential papers is discoverable not because of its content but because of its position in a community of practice. This is particularly relevant for data discovery in scientific and ML contexts, where the downstream task is well-defined and community consensus about which datasets are appropriate for a given task carries significant information. It also highlights a limitation of purely content-based approaches: Two datasets may be semantically equivalent by any value-based measure, but one is far more discoverable than the other because of the social infrastructure surrounding it.

1.4.2.5 Academic Repositories: Schema-Driven Discovery

Platforms such as ICPSR[21] catalog thousands of research datasets with standardized variable-level metadata variable names, value labels, question wording, and collection methodology: enabling discovery queries far more precise than keyword search. A researcher studying voter turnout can search not just for datasets mentioning "voting" but also for datasets containing a variable recording turnout as a percentage, collected via probability sampling, in US congressional districts, between 1990 and 2020.

This schema-aware discovery is only possible because Inter-university Consortium for Political and Social Research (ICPSR) enforces a metadata standard that captures the semantics of individual variables rather than just the dataset as a whole. It represents one of the few settings where the metadata is rich enough to support structured, predicate-based discovery: the kind of query that resembles a database WHERE clause more than a keyword search. The gap between what ICPSR achieves through careful human curation and what automated discovery systems achieve over uncurated data lakes is one of the most important motivating observations for the

[20] https://huggingface.co/

[21] https://www.icpsr.umich.edu/sites/icpsr/home

field: If all data lakes were documented to ICPSR standards, several of the discovery problems would not exist.

1.4.2.6 Synthetic Data: Discovery When Real Data Cannot Be Shared

Synthetic data platforms such as Gretel.ai[22] and MostlyAI[23] represent an emerging marketplace category in which the seller does not transfer raw data at all, but instead trains a generative model on the buyer's private data and returns synthetic records that preserve statistical properties without exposing real individuals. LendingClub,[24] for instance, holds loan application records that cannot be shared due to financial regulation; by uploading a sample to Gretel.ai, it can generate a synthetic dataset matching the distributional properties of its loan portfolio for use in testing and model development.

From a discovery perspective, synthetic data markets raise a fundamental question about what it means for data to be relevant. A synthetic dataset that faithfully reproduces the marginal distributions and pairwise correlations of the original may nonetheless fail to support a specific downstream join or causal analysis if higher-order structure is not preserved. Assessing whether a synthetic dataset is fit for a given discovery purpose, without access to the real data it was generated from, is a novel and largely unsolved variant of the utility assessment problem.

The Information Paradox as a Discovery Problem Across all of these categories of data markets, a common structural challenge recurs. The buyer must assess the utility of a dataset well enough to justify acquisition, but the seller restricts pre-acquisition inspection to protect commercial value, privacy, or competitive advantage. This tension, known in economics as Arrow's information paradox (Arrow 1962; Kennedy et al. 2022), is not merely a market design problem, but it is also fundamentally a discovery problem: how much can be inferred about a dataset's relevance and utility from the limited view that a seller is willing to expose?

Each marketplace category resolves this paradox differently and, in doing so, implicitly defines a different discovery setting. Clean rooms expose nothing but a schema. Alternative data brokers expose aggregate samples. Open data portals expose everything but document nothing. Synthetic data markets expose a privacy-safe proxy. Understanding which resolution mechanism is in play determines which discovery operations are applicable and how well they can perform. This diversity of settings is precisely what motivates the formal treatment of discovery settings in Chap. 2 and the breadth of operations and indexing strategies covered in Chaps. 3 and 4.

[22] https://www.nvidia.com/en-us/use-cases/synthetic-data-generation-for-agentic-ai/

[23] https://mostly.ai/

[24] https://www.lendingclub.com/

1.4.3 Pollution Explanation

As a concrete example for adata discovery use case, we want to present a prototype system for *explaining* local particle dust pollution peaks by enriching citizen-sensed air-quality readings with external contextual data (Esmailoghli et al. 2019). This work was a contribution to the Data Science Challenge 2019 by the German Computer Science Society (Meyer et al. 2019). The core dataset (Luftdaten) provides particulate matter concentration and spatiotemporal sensor metadata. However, the historical air-quality sensor logs alone do not contain enough descriptive variables to infer plausible contributors to pollution peaks, e.g., traffic, industrial activity, events, and meteorological conditions. Such connections are vital for a meaningful inference and causal *hypotheses* analysis about pollution drivers.

To address this problem, Esmailoghli et al. (2019) propose a system that integrates multiple external sources with information on weather, urban infrastructure/streets, air traffic, and public events and then applies stream-oriented outlier detection plus outlier explanation to surface descriptive features that discriminate high-pollution episodes from normal conditions. Note that in this setup, the approach is hypothesis-based in a sense that the researchers already knew what kind of datasets and sources might contain relevant information for explaining particle dust pollution spikes. Thus, the prototype depicted in Fig. 1.3 explicitly integrates four categories of external data to create a richer feature set:

- **Weather:** temperature, precipitation, humidity, pressure, wind speed, and wind direction (Deutscher Wetterdienst 2019)
- **Geo/street data:** neighborhood descriptors around sensors, such as, streets and intersections statistics, extracted from Open Street Map (OpenStreetMap contributors 2017)
- **Air traffic:** flight routes around airports obtained from Flight Radar,[25] used to identify overlaps between flight paths and polluted regions such as Berlin
- **Public events and holidays:** event name and start/end dates (holidays plus selected high-crowd events)

1.4.3.1 Findings Produced by the Prototype

The proposed architecture won the challenge and derived interesting data-driven insights that connect the outcome to data from all three-dimensional sources. For example, the authors report that weather features strongly modulate air quality:

- Higher *wind speed* tends to reduce particulate matter via dispersion.

[25] https://www.flightradar24.com

- *wind direction* can reveal directional pollution sources: For an example cluster, higher pollution is observed when wind direction is roughly ∼110–120°, which the authors associate with a nearby highway alignment in that sector.
- Annual patterns suggest higher pollution in winter, plausibly linked to heating systems and inversion phenomena.

Using Open Street Map (OSM)-derived features, the system highlights high-pollution areas around the *Berlin Ring* (a highway circling inner Berlin area). It also flags a notable latitude band (52.556) and links it to *Tegel Airport* operations and aircraft routing/holding patterns. The affected sensors also show a different seasonal signature higher pollution in summer and around New Year consistent with tourism peaks.

The system detects public event-linked spikes: A sensor cluster near *Potsdamer Platz*, a popular place in Berlin, Germany, shows increased particulate matter on the opening/closing days of the Berlin International Film Festival (Berlinale), and similar spikes occur on other public events/holidays, such as Valentine's Day.

1.4.3.2 Key Limitations of the Prototype

While the proposed prototype demonstrates the ability to obtain explanations by combining different sources to a given dataset, the system is *primed and maybe even biased by the chosen external sources*: Different enrichment sources can change the explanations. The architecture shown in Fig. 1.3 comprises four conceptual blocks out of which the first deals with extracting information from external sources. The proposed architecture resembles a federated architecture, which is the reasonable choice if the downstream use case is predictable and confined to the selected sources domains. Any approach to expand the sources or the use cases would require

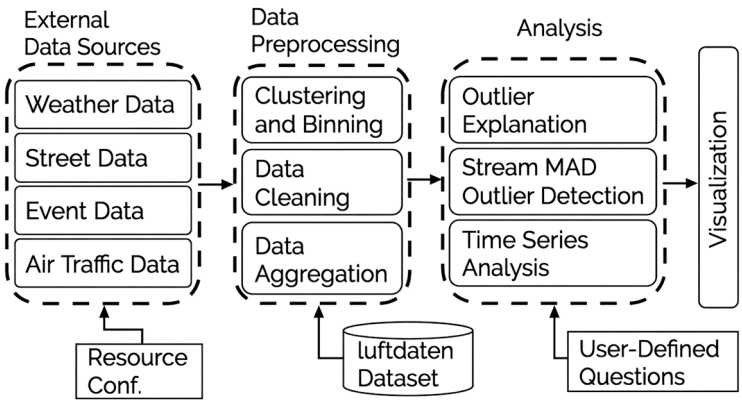


Fig. 1.3 System overview: external data integration → preprocessing → analysis (outlier detection and explanation and time series analysis) → visualization/user-defined questions

rewiring the pipeline architecture. Here the motivation for a more open and ad hoc integration-oriented pipeline becomes evident, where source and table selection can happen at query time and not at indexing time. In such a scenario, discovery systems would be needed that can find related and relevant tables through discovery operations.

1.5 Book Outline

In this book, we provide a comprehensive overview of state-of-the-art in data discovery research and system development by covering several entangled angles. In Chap. 2, we discuss different settings with regard to data locality, highlighting that discovery can be conduct on centralized lakes or decentralized infrastructures. In Chap. 3, we define and discuss common discovery operations in detail and highlight commonalities and computational and semantic challenges in implementing such operations. In Chap. 4, we present representative indexing techniques that enable the scalable execution of the discussed operations. In Chap. 5, we survey end-to-end discovery systems and highlight technical challenges in connecting the different operations and indexes for real-world discovery tasks. In Chap. 6, we dive into a discussion on how research in data discovery is conduct and how performance is measured, highlighting open challenges for the community. In the final chapter, we summarize the main messages of this book and outline future research in data discovery and how we expect this technology to develop in the upcoming years.

References

Abedjan Z, Golab L, Naumann F (2015) Profiling relational data: a survey 24(4):557–581

Alexandrov A, Bergmann R, Ewen S, Freytag J, Hueske F, Heise A, Kao O, Leich M, Leser U, Markl V, Naumann F, Peters M, Rheinländer A, Sax MJ, Schelter S, Höger M, Tzoumas K, Warneke D (2014) The stratosphere platform for big data analytics 23(6):939–964, URL https://doi.org/10.1007/s00778-014-0357-y

Angles R, Gutierrez C (2008) The expressive power of SPARQL. Springer-Verlag, Berlin, Heidelberg, ISWC '08, p 114–129, URL https://doi.org/10.1007/978-3-540-88564-1_8

Arrow KJ (1962) Economic welfare and the allocation of resources for invention. In: Readings in industrial economics: Volume two: Private enterprise and state intervention, Springer, pp 219–236

Brickley D, Burgess M, Noy NF (2019) Google dataset search: Building a search engine for datasets in an open web ecosystem. ACM, pp 1365–1375, URL https://doi.org/10.1145/3308558.3313685

Deutscher Wetterdienst (2019) Climate data center. FTP repository, URL ftp://ftp-cdc.dwd.de/pub/CDC/observations_germany/climate/, accessed 2019-02-06

Dixon J (2010) Pentaho, hadoop, and data lakes. URL https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/, blog post (accessed 2026-02-20)

Dong Y, Takeoka K, Xiao C, Oyamada M (2021) Efficient joinable table discovery in data lakes: A high-dimensional similarity-based approach. In: 37th IEEE International Conference on Data

Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021, IEEE, pp 456–467, URL https://doi.org/10.1109/ICDE51399.2021.00046

Esmailoghli M, Abedjan Z (2020) CAFE: constraint-aware feature extraction from large databases. www.cidrdb.org, URL http://cidrdb.org/cidr2020/gongshow2020/gongshow/abstracts/cidr2020_abstract86.pdf

Esmailoghli M, Redyuk S, Martinez R, Abedjan Z, Rabl T, Markl V (2019) Explanation of air pollution using external data sources. In: BTW

Esmailoghli M, Quiané-Ruiz J, Abedjan Z (2021) COCOA: correlation coefficient-aware data augmentation. OpenProceedings.org, pp 331–336, URL https://doi.org/10.5441/002/edbt.2021.30

Esmailoghli M, Quiané-Ruiz J, Abedjan Z (2022) MATE: multi-attribute table extraction 15(8):1684–1696, URL https://www.vldb.org/pvldb/vol15/p1684-esmailoghli.pdf

Esmailoghli M, Schnell C, Miller RJ, Abedjan Z (2025) BLEND: A Unified Data Discovery System . IEEE Computer Society, Los Alamitos, CA, USA, pp 737–750, DOI https://doi.org/10.1109/ICDE65448.2025.00061, URL https://doi.ieeecomputersociety.org/10.1109/ICDE65448.2025.00061

Fernandez RC, Abedjan Z, Madden S, Stonebraker M (2016) Towards large-scale data discovery: position paper. In: Roy SB, Stefanidis K, Koutrika G, Riedewald M, Lakshmanan LVS (eds) Proceedings of the Third International Workshop on Exploratory Search in Databases and the Web, ACM, pp 3–5, URL https://doi.org/10.1145/2948674.2948675

Galhotra S, Gong Y, Fernandez RC (2023) Metam: Goal-oriented data discovery. IEEE, pp 2780–2793, URL https://doi.org/10.1109/ICDE55515.2023.00213

Hai R, Koutras C, Quix C, Jarke M (2023) Data lakes: A survey of functions and systems 35(12):12571–12590, DOI https://doi.org/10.1109/TKDE.2023.3270101

Huang Z, Liu J, Alabi D, Fernandez RC, Wu E (2023) Saibot: A differentially private data search platform. Proc VLDB Endow 16(11):3057–3070, DOI https://doi.org/10.14778/3611479.3611508, URL https://www.vldb.org/pvldb/vol16/p3057-huang.pdf

Idreos S, Papaemmanouil O, Chaudhuri S (2015) Overview of data exploration techniques. In: Sellis TK, Davidson SB, Ives ZG (eds) Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015, ACM, pp 277–281, URL https://doi.org/10.1145/2723372.2731084

Kennedy J, Subramaniam P, Galhotra S, Fernandez RC (2022) Revisiting online data markets in 2022: A seller and buyer perspective. SIGMOD Rec 51(3):30–37, URL https://doi.org/10.1145/3572751.3572757

Kirstein F, Dittwald B, Dutkowski S, Glikman Y, Schimmler S, Hauswirth M (2019) Linked data in the european data portal: A comprehensive platform for applying DCAT-AP. In: Electronic Government (EGOV 2019), Springer, Lecture Notes in Computer Science, vol 11685, pp 192–204, DOI https://doi.org/10.1007/978-3-030-27325-5_15

Meyer HJ, Grunert H, Waizenegger T, Woltmann L, Hartmann C, Lehner W, Esmailoghli M, Redyuk S, Martinez R, Abedjan Z, et al. (2019) Particulate matter matters-the data science challenge@ btw 2019. Datenbank-Spektrum pp 1–18

Miller RJ (2018) Open data integration 11(12):2130–2139, DOI https://doi.org/10.14778/3229863.3240491, URL http://www.vldb.org/pvldb/vol11/p2130-miller.pdf

Nargesian F, Zhu E, Pu KQ, Miller RJ (2018) Table union search on open data 11(7):813–825, DOI https://doi.org/10.14778/3192965.3192973, URL http://www.vldb.org/pvldb/vol11/p813-nargesian.pdf

Nargesian F, Zhu E, Miller RJ, Pu KQ, Arocena PC (2019) Data lake management: challenges and opportunities 12(12):1986–1989, URL https://doi.org/10.14778/3352063.3352116

OpenStreetMap contributors (2017) Planet dump retrieved from https://planet.osm.org . https://www.openstreetmap.org

Paton NW, Wu Z (2024) Dataset discovery and exploration: State-of-the-art, challenges and opportunities. In: Tanca L, Luo Q, Polese G, Caruccio L, Oriol X, Firmani D (eds) Proceedings 27th International Conference on Extending Database Technology, EDBT 2024, Paestum, Italy,

March 25 - March 28, OpenProceedings.org, pp 854–857, URL https://doi.org/10.48786/edbt.2024.87

Paton NW, Chen J, Wu Z (2024) Dataset discovery and exploration: A survey 56(4):102:1–102:37, URL https://doi.org/10.1145/3626521

Quix C, Hai R (2018) Data lake. In: Sakr S, Zomaya A (eds) Encyclopedia of Big Data Technologies, Springer International Publishing, Cham, pp 1–8, URL https://doi.org/10.1007/978-3-319-63962-8_7-1

Santos ASR, Bessa A, Chirigati F, Musco C, Freire J (2021) Correlation sketches for approximate join-correlation queries. ACM, pp 1531–1544, URL https://doi.org/10.1145/3448016.3458456

Sawadogo P, Darmont J (2021) On data lake architectures and metadata management 56(1):97–120, URL https://doi.org/10.1007/s10844-020-00608-7

Terrizzano IG, Schwarz PM, Roth M, Colino JE (2015) Data wrangling: The challenging yourney from the wild to the lake. www.cidrdb.org, URL http://cidrdb.org/cidr2015/Papers/CIDR15_Paper2.pdf

Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin MJ, Shenker S, Stoica I (2012) Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI), USENIX Association, USA, p 2

Zhang S, Balog K (2019) Web table extraction, retrieval and augmentation. Association for Computing Machinery, New York, NY, USA, URL https://doi.org/10.1145/3331184.3331385

Zhu E, He Y, Chaudhuri S (2017) Auto-join: Joining tables by leveraging transformations. Proc VLDB Endow 10(10):1034–1045, DOI https://doi.org/10.14778/3115404.3115409, URL http://www.vldb.org/pvldb/vol10/p1034-he.pdf

Zhu E, Deng D, Nargesian F, Miller RJ (2019) JOSIE: overlap set similarity search for finding joinable tables in data lakes. ACM, pp 847–864, URL https://doi.org/10.1145/3299869.3300065

Chapter 2
Discovery Settings



Abstract The structural properties of the data repository fundamentally determine which discovery operations are feasible, how accurate their results can be, and where algorithms necessarily degrade. This chapter introduces a formal taxonomy of *discovery settings*: the environmental configurations within which all discovery operations and indexing strategies must function. We characterize discovery environments along three primary axes. *Locality* describes where data physically and administratively resides, ranging from fully centralized repositories under a single administrative domain to loosely federated networks of autonomous providers. *Visibility* describes how much of each dataset's content is accessible at discovery time, from full tuple-level access to metadata-only descriptions exposed through catalog interfaces. *Mutability* describes how the repository evolves over time, distinguishing static archives from batch-updated enterprise lakes and continuously streaming sensor platforms. Orthogonal to these axes, privacy and governance constraints impose policy-driven restrictions on data access even when datasets are physically colocated, motivating a family of privacy-preserving discovery mechanisms including differential privacy, secure multiparty computation, and synthetic data. Additionally, we discuss multimodal lakes, schema evolution, and the open-world assumption as structural properties that cut across all three axes. The chapter concludes by arguing that discovery algorithms cannot be meaningfully evaluated in isolation from the structural setting in which they are deployed.

Keywords Centralized discovery · Distributed discovery · Privacy and discovery

2.1 Introduction

Most existing work on data discovery implicitly assumes a centralized repository in which the discovery system has full access to the underlying tables (Fernandez et al. 2018; Miller 2018; Bogatu et al. 2020; Fernandez et al. 2018; Esmailoghli et al. 2025; Galhotra et al. 2023; Gong et al. 2024, 2023). However, real-world deployments increasingly operate under heterogeneous access constraints: datasets may reside across multiple organizations, expose only limited metadata, or evolve

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027

Z. Abedjan et al., *Data Discovery in Data Lakes*,
https://doi.org/10.1007/978-3-032-30822-1_2

continuously through streaming ingestion pipelines. These factors significantly influence which discovery operations are feasible and how accurate their results can be.

This chapter establishes a taxonomy of *discovery settings*: the structural configurations within which all discovery operations (Chap. 3) and indexing strategies (Chap. 4) must function. Rather than defining specific discovery operators, we characterize the environmental constraints that determine which operators are feasible, how they must be adapted, and where they necessarily degrade.

We organize discovery settings along three primary axes:

1. *Locality*, capturing where data physically and administratively resides on a spectrum from fully centralized to distributed across multiple data providers
2. *Visibility*, capturing how much of each table's content is accessible at discovery time, ranging from full raw-data access to metadata-only views
3. *Mutability*, capturing whether the repository is static and batch-indexed or evolves continuously through streaming ingestion

Orthogonal to these axes, *privacy and governance constraints* impose policy-driven restrictions on data visibility and access, even when datasets are physically colocated.

To ground the exposition, we use a running example throughout this chapter.

Example 2.1 A public health analyst who wishes to enrich a regional disease incidence dataset with demographic, environmental, and mobility data drawn from repositories they do not directly control. This scenario captures the essential tensions of data discovery: the analyst knows, at a high level, what kind of data they need but not precisely where it resides, and the repositories they must search impose varying degrees of access restriction.

The remainder of this chapter is organized as follows: Sect. 2.2 introduces the formal model of a data lake and defines the access function that underpins the taxonomy. Sections 2.2.1 through 2.2.3 characterize centralized, federated, and hybrid discovery settings. Section 2.3 discusses privacy and governance constraints as a cross-cutting overlay. Section 2.4 briefly addresses multimodal lakes. We conclude with a discussion of schema evolution and the open-world assumption, two structural properties that influence evaluation across all discovery settings.

2.2 Data Lake

In Chap. 1, we already defined *data lakes* as *a repository that keeps heterogeneous data from many sources in its original form and defers imposing structure until use*. A data lake stores datasets in their native formats with minimal upfront integration, enabling flexible downstream analytics but introducing significant challenges for dataset discovery and integration.

Formally, we define a data lake as

$$\mathcal{L} = \{T_1, T_2, \ldots, T_n\}.$$

Each table T_i is a relation

$$T_i = (A_i, R_i),$$

where:

- $A_i = \{a_{i1}, \ldots, a_{ik_i}\}$ is the attribute set.
- $R_i \subseteq \prod_{j=1}^{k_i} Dom(a_{ij})$ is the tuple set.

Discovery algorithms operate over $\mathcal{L}$ subject to structural constraints imposed by the repository setting.

Access Function Given this definition of a data lake, the key question for discovery is not only what $\mathcal{L}$ contains, but how much of it a discovery system can actually see and use to assist discovery. The central formal handle for characterizing discovery settings is the access function $\alpha : \mathcal{L} \to \{full, partial, metadata, none\}$, which assigns to each table the degree of inspection available to the discovery system at query time.

- $\alpha(T_i) = full$: Raw tuples are accessible.
- $\alpha(T_i) = partial$: Restricted access such as sampled rows or summaries.
- $\alpha(T_i) = metadata$: Only schema-level descriptions and annotations are visible; no tuple content is accessible.
- $\alpha(T_i) = none$: The table is not accessible to discovery.

Different discovery settings correspond to different distributions of access levels across the tables in $\mathcal{L}$.

Open World Assumption The access function characterizes what the discovery system can see *within* the lake. A complementary structural property concerns what lies *outside* it. A property that distinguishes data discovery from classical database querying is the *open-world assumption* (Abiteboul et al. 1995): A discovery system cannot know what tables are *not* in the lake. In a closed-world database, the absence of a query result implies the result does not exist. In a data lake, a null result may mean the data does not exist, or it may mean the data exists but is not indexed, not accessible, or described using different terminology.

This assumption has two important consequences.

First, recall is fundamentally difficult to measure as there is no ground truth set of all relevant tables in the universe of possible data sources. Second, discovery algorithms must be robust to incompleteness, returning useful candidates even when relevant datasets may lie outside the indexed repository.

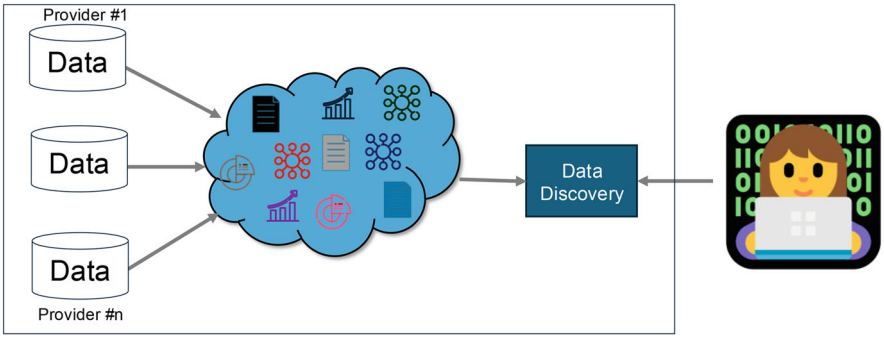


Fig. 2.1 Centralized data discovery

2.2.1 Centralized Setting

In the centralized setting shown in Fig. 2.1, a single system stores and manages all datasets. Formally,

$$\alpha(T_i) = full \quad \forall T_i \in \mathcal{L}.$$

This model underlies most traditional data lakes and enterprise analytics platforms. Centralization enables full access to raw data, rich metadata, and domain experts. As a result, the system can support sophisticated discovery operators, tightly integrate with downstream use cases, and perform deep inspection over dataset contents. Nearly all classical discovery operators like keyword search, join discovery, union search, task-aware search, and natural language interfaces were first studied in this setting.

The primary drawback is scope: Centralized discovery is typically confined to a single organization or administrative domain. Data sharing across organizational boundaries is limited by governance, legal, and privacy concerns.

The following example showcases the possibilities of the centralized setting:

Example 2.2 Consider a state health department that maintains a centralized analytics platform containing hospital admissions, vaccination records, census demographics, and environmental monitoring data. All datasets are stored in a unified data lake and can be inspected by the discovery system.

Suppose a public health analyst wants to study the relationship between air pollution and respiratory disease. A join discovery system can directly inspect the values in a hospital admissions table and an environmental monitoring table to identify a shared geographic identifier such as county code. Because tuple-level access is available, the system can compute value overlap statistics, profile attribute distributions, and rank candidate joins with high confidence.

2.2.2 *Federated Setting*

A federated discovery setting arises when datasets remain distributed across multiple providers and the discovery system has only restricted access to each repository.

In this setting,

$$\alpha(T_i) \in \{metadata, partial\}.$$

Note that certain data providers may provide full access to their data, but federated settings usually do not use all datasets for supporting data discovery. Federated discovery is increasingly common in cross-organizational analytics, public data markets, and regulated domains such as healthcare and finance. Data providers retain control over their datasets and expose only approved views such as schema descriptions, statistical summaries, or curated metadata as illustrated in Fig. 2.2.

Federation offers a natural alignment with real-world data governance structures. However, it significantly constrains discovery operations. Many discovery algorithms rely on inspecting raw data values to detect join relationships or semantic similarity between attributes. When only metadata is available, discovery must rely on schema-level signals, textual descriptions, or summary statistics.

A canonical example is Google Dataset Search (Brickley et al. 2019), where publishers expose metadata using the Schema.org vocabulary and discovery operates entirely over dataset descriptions. Several recent papers (Behme et al. 2024; Esmailpour et al. 2025) have studied data discovery in federated settings, where high-level data statistics are available.

Example 2.3 Returning to the public health scenario, suppose the analyst now wishes to incorporate mobility data collected by a transportation agency and



Fig. 2.2 Federated data discovery. The user can usually access summaries and statistics but not the raw data

pollution measurements maintained by an environmental regulator. Each agency stores its datasets independently and exposes only dataset descriptions and approved statistical summaries.

A federated discovery system may learn that a transportation dataset contains attributes such as *county_id* and *daily_traffic_volume*, but it cannot inspect individual records. To estimate joinability, which is a score to describe how well two tables join (we define it more formally in Chap. 3), the discovery system must rely on metadata and summary statistics such as value histograms or declared schema annotations.

Note that the federated setting is defined at the repository level: It characterizes an environment in which no provider grants unrestricted tuple-level access. Individual tables may expose varying levels of detail, but the absence of any fully accessible provider distinguishes this setting from the hybrid case (Sect. 2.2.3).

2.2.3 Hybrid Setting

Hybrid settings arise when a discovery system operates over repositories with heterogeneous access levels.

Some datasets are fully accessible, while others expose only restricted metadata or summaries:

$$\alpha(T_i) \in \{full, partial, metadata\}$$

Hybrid environments are common in open-data ecosystems and enterprise data platforms that combine internal datasets with external sources.

Hybrid discovery systems must reason across asymmetric visibility levels. Discovery operators that require tuple-level inspection may be applicable to some datasets but must fall back to metadata-based heuristics for others. Designing algorithms that gracefully degrade under reduced visibility remains an important research challenge.

Example 2.4 Consider a regional public health platform that combines an internal hospital admissions table (centrally managed, α =`full`) with mobility datasets from a commercial data broker (available as curated samples, α = `partial`) and demographic data from a national statistics office (accessible only as published metadata, α =`metadata`).

A hybrid discovery system must coordinate across these asymmetric visibility levels. For the internal table, it can compute exact value distributions and join overlap scores. For the broker dataset, it can estimate joinability from sampled rows, accepting some estimation error. For the statistics office metadata, it must rely entirely on schema annotations and declared value ranges. The overall discovery

result is therefore a composite of high-confidence and approximate candidates, and the system must communicate this uncertainty to the analyst.

While hybrid approaches increase coverage, they also inherit the limitations of federated systems: operators that require content-level inspection may fail or degrade gracefully when applied to metadata-only datasets.

2.3 Privacy and Governance Constraints

Privacy is a cross-cutting overlay on all three axes of the discovery setting taxonomy. A table in a centralized lake may still have $\alpha(T_i) \neq full$, if privacy regulations or organizational governance policies restrict access at the column or row level, even when the table is physically colocated with the discovery system.

Several technical mechanisms enable discovery despite restricted access, each corresponding to a different instantiation of the summary function in the federated model.

Differential Privacy (DP) (Dwork 2008) adds calibrated random noise to aggregate statistics (histograms, cardinality estimates, correlation coefficients) so that the presence or absence of any individual tuple cannot be inferred from the released summary. DP-protected summaries (Huang et al. 2023) can be used for approximate joinability and unionability estimation: a candidate table whose DP-noised value histogram shows significant overlap with the query table's histogram is likely joinable, but the noise introduces estimation error that grows with the privacy budget consumed.

Secure Multiparty Computation (MPC) (Lindell 2020) allows two parties to compute a joint function of their private inputs without revealing those inputs to each other. In the clean room setting described in Chap. 1, MPC enables two organizations to compute the intersection cardinality of their customer ID sets—a key input to joinability estimation, without either party seeing the other's actual IDs. MPC protocols for set intersection are computationally feasible for moderate set sizes (Kamara et al. 2014) but become expensive for the large-scale column comparisons required by bulk join search.

Synthetic Data replaces the original table with a statistically faithful synthetic replica that preserves marginal distributions and low-order correlations but cannot be linked to real individuals. Synthetic data allows full-access discovery operations to run against the synthetic replica as a proxy. The limitation, as noted in Chap. 1, is that synthetic data may not preserve higher-order structure needed for specific joins or causal analyses; a synthetic dataset that matches the original's column marginals may still fail a downstream join because it does not preserve the joint distribution of the join key and the target variable.

Understanding which privacy-preserving mechanism is appropriate for a given discovery setting is a design decision that depends on the sensitivity of the data, the computational resources available, and the acceptable estimation error in discovery

results. This is an active area of research at the intersection of data management and privacy engineering.

2.4 Multimodal Settings

Modern data lakes increasingly contain heterogeneous artifacts including relational tables, text documents, geospatial data, images, and graph structures. This book focuses primarily on relational table discovery, but the setting taxonomy should acknowledge multimodal lakes because the structural constraints of multimodal data affect both index design and operation feasibility.

Each modality induces a different representation space and a different notion of joinability or relevance. A relational table can be joined to a text corpus if the table contains entity names that appear in the documents; a time series can be correlated with a relational table if both are indexed on the same temporal dimension. Cross-modal discovery requires modality-aware indexes that can represent and compare artifacts across representation spaces.

Example 2.5 In addition to relational datasets, public health analysts often rely on heterogeneous data sources. For example, epidemiological studies may combine structured hospital admission tables with textual clinical reports, geospatial pollution maps, and mobility traces derived from transportation systems. A discovery system operating in a multimodal lake must therefore identify relationships across different representation spaces. A table containing county-level infection rates may need to be linked with geospatial raster data describing pollution concentrations or with textual reports describing local outbreaks. Supporting such cross-modal discovery requires indexes capable of comparing structured and unstructured artifacts.

The emergence of multimodal foundation models (large language models with vision and structured data capabilities) is beginning to make cross-modal discovery tractable by mapping heterogeneous data objects into a shared embedding space. Section 4.8 discusses these approaches in the context of embedding-based indexing. For the purposes of the setting taxonomy, it suffices to note that multimodal lakes impose additional heterogeneity on the access function.

2.5 Temporal Dimension of Discovery

While locality and visibility determine where data resides and how much of it can be inspected, the *temporal dimension* determines how the repository evolves over time. In many real-world environments datasets are not static artifacts but continuously evolving collections whose schema, values, and coverage change as new observations are ingested.

Temporal evolution affects both the correctness of discovery results and the design of discovery indexes. A join candidate that appears highly compatible with a query table today may become invalid tomorrow if the underlying distributions drift or if new data sources are introduced. Discovery systems must therefore reason not only about dataset content but also about the time at which that content is observed.

We distinguish four common temporal regimes in data discovery systems.

Static Repositories In the simplest setting, the repository is assumed to be static during index construction and query processing. This assumption underlies most existing research prototypes for dataset discovery (Nargesian et al. 2018; Bogatu et al. 2020; Fernandez et al. 2018). A static setting allows discovery systems to compute rich offline statistics such as value distributions, join overlap scores, or semantic embeddings without worrying about index staleness.

Batch-Updated Repositories Many enterprise data lakes evolve through periodic batch ingestion pipelines. New datasets or updated versions of existing tables are added at regular intervals (e.g., nightly or weekly). Discovery indexes must therefore be refreshed periodically to incorporate newly ingested data. According to Zhu et al. (2019), incremental index maintenance techniques can reduce recomputation costs by updating statistics only for affected tables.

Streaming Repositories In streaming environments, datasets are updated continuously through event streams or sensor feeds. Examples include IoT platforms, financial transaction logs, and environmental monitoring networks. In such settings, discovery systems must maintain approximate and incrementally updated statistics so that joinability or correlation estimates remain accurate over time.

Historical Repositories Many scientific and government repositories maintain historical snapshots of datasets rather than overwriting old data. Discovery systems must therefore reason about the *temporal validity* of datasets: two datasets may be structurally compatible but describe different time periods and therefore not be analytically comparable.

These temporal regimes introduce additional challenges for discovery systems:

- **Index staleness:** Statistics used to rank datasets may become outdated between index refreshes, causing discovery results to degrade in quality without any visible indication. For example, a nightly refreshed join index may rank a logistics dataset as highly compatible with a query table based on a shared `postal_code` column; if the logistics provider silently migrated to a different geocoding scheme during the day, that join candidate is stale but will not be flagged as such until the next index rebuild.
- **Concept drift:** The relationship between datasets may evolve over time, so a join candidate that was valid at index construction time may no longer be joinable at query time. A canonical example in public health is mobility data: A transportation dataset that strongly correlated with disease incidence during a pre-pandemic period may exhibit a completely different relationship

after behavioral patterns shifted, rendering previously discovered correlations misleading for a current analysis.

- **Temporal alignment:** Datasets may cover different time ranges or sampling frequencies, making them structurally compatible but analytically incomparable. For instance, a county-level air quality dataset sampled hourly and a hospital admissions dataset recorded monthly share the same geographic identifier and appear joinable at the schema level, but any correlation analysis that ignores their different temporal granularities will aggregate air quality readings in a way that obscures the short-term exposure effects the analyst is trying to measure.
- **Version management:** Multiple historical versions of the same dataset may coexist in the lake, and the discovery system must determine which version is most appropriate for the analyst's use case. A census bureau may publish population estimates annually, with each release superseding the last but all prior versions remaining accessible. A discovery system that returns the most recent version by default may produce incorrect results for an analyst who needs the vintage year that was current at the time their outcome data was collected.

Understanding the temporal properties of a repository is therefore essential for interpreting discovery results. A discovery algorithm that performs well on static repositories may produce unstable or misleading results in streaming environments if it does not account for evolving distributions.

2.6 Summary of Discovery Settings

The preceding sections introduced discovery settings as structural configurations that constrain how data discovery systems can operate. We characterized these settings along three primary dimensions: locality, visibility, and mutability. Locality captures where datasets reside and whether they are managed within a single administrative domain or distributed across multiple providers. Visibility captures how much of a dataset's content can be inspected by the discovery system, ranging from full tuple-level access to metadata-only descriptions. Mutability captures the temporal dynamics of the repository, distinguishing static repositories from those that evolve through periodic batch ingestion or continuous streaming updates. Formally, a discovery setting can therefore be represented as

$$Setting = (Locality, Visibility, Mutability).$$

Different real-world data platforms instantiate different combinations of these properties. Table 2.1 summarizes several representative systems discussed throughout the book and maps them to the discovery axes introduced in this chapter. Several observations follow from this summary. First, centralized environments provide the richest discovery capabilities because algorithms can inspect raw data and compute detailed statistics over dataset contents. The full tuple-level visibility

Table 2.1 Examples of real-world systems mapped to the discovery setting axes of locality, visibility, and mutability

System/example	Locality	Visibility	Mutability	Discovery characteristics
Enterprise Data Lake	Centralized	Full data access	Batch updates	Enables value-level discovery such as join and union search
Government Open Data Portals (Data.gov, EU Portal)	Hybrid/Federated	Metadata + external links	Periodic updates	Discovery relies heavily on metadata quality and portal harvesting
Google Dataset Search	Federated	Metadata only	Mostly static	Web-scale dataset search using schema.org metadata
Data Clean Rooms (Snowflake, AWS Clean Rooms)	Federated	Restricted/secure computation	Batch updates	Discovery based on partner identity and schema-level information
Alternative Data Brokers (Bloomberg, Thinknum)	Federated	Metadata + curated samples	Batch updates	Utility gap between preview metadata and full dataset
Labeling Markets (Scale AI, Surge AI)	Centralized platform	Task specification rather than datasets	Continuous tasks	Discovery becomes task specification rather than dataset retrieval
Academic Repositories (ICPSR)	Centralized	Rich variable-level metadata	Static/versioned	Schema-aware discovery over curated metadata
Dataset Platforms (HuggingFace Datasets)	Hybrid	Metadata + community signals	Continuous additions	Discovery through task tags, benchmarks, and citations
Synthetic Data Platforms (Gretel, MostlyAI)	Federated	Synthetic proxy data	Batch updates	Discovery operates over generated data rather than original datasets
IoT/Sensor Platforms	Centralized or hybrid	Full or partial access	Streaming	Discovery must account for evolving distributions and temporal alignment

available in enterprise data lakes enables precise computation of join overlap scores, value distribution profiles, and semantic embeddings over column content. However, the scope of centralized discovery is typically bounded by organizational or administrative boundaries, limiting coverage to datasets that have been formally ingested into the managed repository. Second, federated environments are increasingly common due to governance constraints, privacy regulations, and the emergence of data marketplaces. Federated settings fundamentally restrict discovery systems to operate over metadata or statistical summaries, and this restriction is not merely a technical inconvenience but a structural property that determines which discovery operations are feasible. Join discovery algorithms that require value-level overlap computation cannot be directly applied; they must be replaced by schema-matching heuristics, textual similarity over column descriptions, or statistical inference from declared cardinality bounds. The precision of discovery results degrades accordingly, and this degradation is often invisible to the analyst unless the system explicitly communicates the visibility level of each candidate. Third, hybrid environments—where internal datasets coexist with external or partially accessible repositories—represent the most common real-world scenario encountered in practice. Enterprise analytics platforms routinely combine a centrally managed data lake with licensed external datasets available only as curated samples, and with public data portals accessible only through metadata application programming interface (APIs). Discovery systems operating in hybrid environments must implement a tiered strategy: applying full content-level operators to internally managed tables, estimation-based operators to sampled external datasets, and metadata-only operators to catalog-exposed sources. The resulting ranked list mixes candidates of heterogeneous reliability, and communicating this heterogeneity to the analyst is itself a design challenge that existing systems address inconsistently. Fourth, the temporal dimension of repositories introduces additional complexity that interacts with all three primary axes. A centralized repository with full visibility but continuous streaming ingestion still requires discovery systems to maintain adaptive statistics and incremental indexes, because a join candidate identified against a stale index snapshot may no longer be valid at query time. Similarly, a federated repository that exposes only metadata is further constrained if that metadata is refreshed infrequently: Schema annotations may describe a dataset's structure as it existed months ago, while the underlying data has evolved. The combination of low visibility and high mutability is therefore particularly challenging, as the discovery system has neither the content access to detect drift directly nor the metadata freshness to infer it indirectly. A fifth observation concerns the interaction between privacy constraints and the three primary axes. Privacy overlays can reduce the effective visibility of a table below what its locality would suggest: A table that is physically colocated with the discovery system in a centralized lake may still be accessible only through differentially private summaries if it contains sensitive individual-level records. This means that the access function α is not determined by locality alone but by the combined effect of physical access, governance policy, and privacy regulation. Discovery systems that treat locality as a proxy for visibility will systematically overestimate the discovery operations available to them.

Taken together, these observations illustrate that discovery algorithms cannot be evaluated independently of the repository setting in which they operate. The structural constraints imposed by locality, visibility, and mutability fundamentally determine what discovery operations are feasible, how reliable their results can be, and where they will silently degrade. The following chapters therefore examine discovery operations and indexing strategies in light of these structural settings, returning to the taxonomy introduced here whenever a new algorithm's applicability depends on assumptions about the underlying repository.

Table 2.1 summarizes several representative systems discussed throughout the book and maps them to the discovery axes introduced in this chapter.

References

Abiteboul S, Hull R, Vianu V (1995) Foundations of databases, vol 8. Addison-Wesley Reading

Behme L, Galhotra S, Beedkar K, Markl V (2024) Fainder: A fast and accurate index for distribution-aware dataset search. Proc VLDB Endow 17(11):3269–3282, https://doi.org/10.14778/3681954.3681999, URL https://www.vldb.org/pvldb/vol17/p3269-behme.pdf

Bogatu A, Fernandes AAA, Paton NW, Konstantinou N (2020) Dataset discovery in data lakes. In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE, pp 709–720, URL https://doi.org/10.1109/ICDE48307.2020.00067

Brickley D, Burgess M, Noy NF (2019) Google dataset search: Building a search engine for datasets in an open web ecosystem. In: Liu L, White RW, Mantrach A, Silvestri F, McAuley JJ, Baeza-Yates R, Zia L (eds) Proceedings of the International World Wide Web Conference (WWW), ACM, pp 1365–1375, URL https://doi.org/10.1145/3308558.3313685

Dwork C (2008) Differential privacy: A survey of results. In: International conference on theory and applications of models of computation, Springer, pp 1–19

Esmailoghli M, Schnell C, Miller RJ, Abedjan Z (2025) BLEND: A Unified Data Discovery System . In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE Computer Society, Los Alamitos, CA, USA, pp 737–750, https://doi.org/10.1109/ICDE65448.2025.00061, URL https://doi.ieeecomputersociety.org/10.1109/ICDE65448.2025.00061

Esmailpour A, Galhotra S, Raychaudhury R, Sintos S (2025) A theoretical framework for distribution-aware dataset search. Proc ACM Manag Data 3(2):90:1–90:26, URL https://doi.org/10.1145/3725227

Fernandez RC, Abedjan Z, Koko F, Yuan G, Madden S, Stonebraker M (2018) Aurum: A data discovery system. In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE Computer Society, pp 1001–1012, URL https://doi.org/10.1109/ICDE.2018.00094

Galhotra S, Gong Y, Fernandez RC (2023) Metam: Goal-oriented data discovery. In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE, pp 2780–2793, URL https://doi.org/10.1109/ICDE55515.2023.00213

Gong Y, Zhu Z, Galhotra S, Fernandez RC (2023) Ver: View Discovery in the Wild . In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE Computer Society, Los Alamitos, CA, USA, pp 503–516, https://doi.org/10.1109/ICDE55515.2023.00045, URL https://doi.ieeecomputersociety.org/10.1109/ICDE55515.2023.00045

Gong Y, Galhotra S, Fernandez RC (2024) Nexus: Correlation discovery over collections of spatio-temporal tabular data. Proceedings of the ACM on Management of Data (PACMMOD) 2(3):154, URL https://doi.org/10.1145/3654957

Huang Z, Liu J, Alabi D, Fernandez RC, Wu E (2023) Saibot: A differentially private data search platform. Proc VLDB Endow 16(11):3057–3070, https://doi.org/10.14778/3611479.3611508, URL https://www.vldb.org/pvldb/vol16/p3057-huang.pdf

Kamara S, Mohassel P, Raykova M, Sadeghian S (2014) Scaling private set intersection to billion-element sets. In: International conference on financial cryptography and data security, Springer, pp 195–215

Lindell Y (2020) Secure multiparty computation. Communications of the ACM 64(1):86–96

Miller RJ (2018) Open data integration. Proceedings of the VLDB Endowment (PVLDB) 11(12):2130–2139, https://doi.org/10.14778/3229863.3240491, URL http://www.vldb.org/pvldb/vol11/p2130-miller.pdf

Nargesian F, Zhu E, Pu KQ, Miller RJ (2018) Table union search on open data. Proceedings of the VLDB Endowment (PVLDB) 11(7):813–825, https://doi.org/10.14778/3192965.3192973, URL http://www.vldb.org/pvldb/vol11/p813-nargesian.pdf

Zhu E, Deng D, Nargesian F, Miller RJ (2019) JOSIE: overlap set similarity search for finding joinable tables in data lakes. In: Boncz PA, Manegold S, Ailamaki A, Deshpande A, Kraska T (eds) Proceedings of the International Conference on Management of Data (SIGMOD), ACM, pp 847–864, URL https://doi.org/10.1145/3299869.3300065

Chapter 3
Discovery Operations



Abstract In this chapter, we describe the most common data discovery operations for finding and integrating tabular datasets. We organize these operations into retrieval-oriented keyword discovery, union-oriented discovery, and join-oriented discovery, and for each, we outline the central questions that determine practical usefulness: how candidate data is identified, how compatibility or usefulness is established, and how results are ranked and validated in terms of expected integration quality and downstream utility. We then compare how these operations behave in central versus distributed environments, emphasizing how data placement, access constraints, latency, and privacy requirements shape what evidence can be examined, what matching signals can be computed, and how confidence and provenance are communicated across boundaries. Building on this, we discuss when discovery requires a sequence of operations, and we motivate optimization techniques for operation sequences that trade off cost, accuracy, and robustness while maintaining transparency about uncertainty and integration effects. Finally, we outline emerging directions in natural-language discovery operations, where intent is expressed in free-form queries and systems must translate intent into executable discovery pipelines.

Keywords Discovery operations · Data lakes

3.1 Introduction and Overview

Data discovery in a data lake is ultimately defined by the operations the lake exposes. Those operations determine what a user can ask for, what form the answers take, and how easily a user can move from an initial question to an actionable, integrated dataset. In that sense, discovery is not just about locating datasets but also about providing a small, well-understood vocabulary of interactions that lets users search, relate, and extend data in a controlled way. Over time, several such operations have become established as the practical building blocks for table-centric discovery: keyword search, union search, and join search. These primitives capture recurring user intents and serve as the foundation for more complex, task-specific

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027

Z. Abedjan et al., *Data Discovery in Data Lakes*,
https://doi.org/10.1007/978-3-032-30822-1_3

workflows. Combinations of operations and natural-language-induced operations are active field of research.

The chapter is organized around these operation types and their practical implications. Section 3.2 introduces keyword search as the basic retrieval-oriented operation for finding tables relevant to a concept. Section 3.3 then presents union search, focusing on discovery aimed at vertical augmentation by finding tables that can be combined by union. Section 3.4 covers join search as the corresponding operation for horizontal enrichment through joinable tables. Section 3.5 broadens the view from single operators to operation sequences and compositions, motivating when pipelines are needed and what it means to optimize them under different criteria. Section 3.6 discusses natural-language queries as an emerging interface that maps user intent to discovery operations. Section 3.7 examines how these operations behave in distributed repositories, where constraints such as access boundaries and limited visibility change the assumptions behind discovery. Finally, Sect. 3.8 discusses additional constraints that shape discovery in practice, beyond the core mechanics of retrieval and integration.

3.2 Keyword Search

A fundamental use case in data discovery is retrieving all tables related to a given concept, typically represented by a set of keywords. While declarative languages have been the dominant interface for accessing in databases, their use for exploration is rather limited as one has to exactly specify in which tables and which attributes to search for specific keywords. With the rise of the World Wide Web, concepts such as retrieval and ranking also found their ways into the realm of structured data and relational databases (Chaudhuri and Das 2009).

Keyword-based retrieval focuses on identifying tables that contain the specified keywords either within their metadata or within their content. Consequently, traditional information retrieval techniques for document search can be directly applied by treating tables as plain documents (Idreos et al. 2015; Agrawal et al. 2002). Keyword search over databases has been studied since the early 2000s and gained renewed interest with the rise of Web-scale data. Today, providing keyword-based retrieval functionality is considered a minimal requirement for most data discovery systems. In database systems, keyword search provides a schema-independent retrieval interface, allowing users to access structured or semi-structured data without explicit knowledge of the database schema or query language.

3.2.1 Keyword Search in Documents and Tables

Keyword search is a well-studied concept in information retrieval (IR) (Manning et al. 2008). In the field of IR, *keyword search* refers to the process of locating

and ranking documents that are relevant to a user's query expressed as a set of keywords. It is one of the most fundamental and widely used techniques for accessing unstructured or semi-structured information and the main interface for Web search.

When a user submits a query consisting of one or more keywords, the IR system performs several key operations. During *query processing*, the system interprets and normalizes the query terms, possibly applying techniques such as stemming or synonym expansion. The system then accesses a previously created inverted index that maps terms to documents, retrieves all documents containing one or more of the query terms, and assigns each document a *relevance score* based on a ranking model, such as the vector space model, TF-IDF weighting, or probabilistic methods like BM25 (Manning et al. 2008). Finally, results are presented to the user in decreasing order of relevance, typically as a ranked list of document snippets or links.

When applying the concept of keyword search to databases, the objects of retrieval changes as databases typically are structured grids of information. Formally keyword search over tables can be defined as follows.

Definition 3.1 (Keyword Search Over Tables) Keyword search over tables is a query language for retrieving relevant tabular components, such as tables, rows, and columns, from a collection of tables D based on a set of user-defined input terms $Q = \{k_1, k_2, \ldots, k_n\}$, i.e., the keyword query. The keyword retrieval system identifies all tabular objects $d \in D$ that contain one or more of the keywords in Q and ranks them according to a predefined relevance function $f(Q, d)$, which measures the degree of match between the query and the data object.

Formally, the result set R is defined as

$$R = \{\, d \in D \mid \exists k_i \in Q,\ k_i \in d \,\}$$

with results ordered by decreasing $f(Q, d)$.

Figure 3.1 shows a conceptual architecture of a keyword-based search system for tabular datasets. The user submits a set of keywords that are matched against an internal index on top of the existing tables. Then the interface presents a ranked list of tables that contain the keywords at various locations of a table. One could rank based on proximity of keywords or their role: schema versus data, within a single cell or across many cells or tables.

3.2.2 Keyword Search Prototypes

Next, we will showcase individual systems that enable keyword-based discovery over tabular datasets. Early systems include DBxplorer (Agrawal et al. 2002), Discover (Hristidis and Papakonstantinou 2002), and Banks (Bhalotia et al. 2002). Also known commercial systems, such as Oracle 9i Text, IBM DB2 Text Information Extender, and Microsoft SQL Server 2000, enable keyword search within their

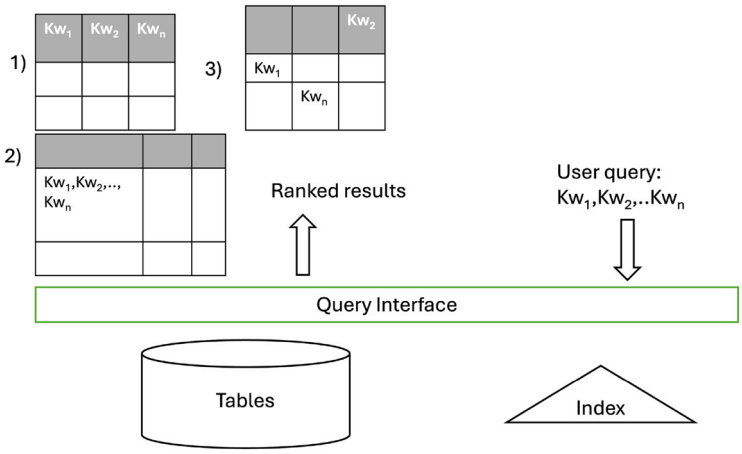


Fig. 3.1 A keyword query interface receives a set of keywords Kw_n and returns a set of relevant tables. Conceptually, the keywords can occur within the schema of the table or in the data. Some systems, such as DBxplorer, also support the retrieval of join paths that as a compound contain a set of queries

major database products by providing full-text indexes of tables (Hristidis and Papakonstantinou 2002).

3.2.2.1 DBxplorer

DBXplorer is one of the earliest keyword search systems. It enables keyword search over relational databases by building so-called symbol tables (Agrawal et al. 2002). These tables serve as inverted indexes that map keyword values to their locations in the database. During a preprocessing phase called publishing, the system scans all tables and attributes and records the coordinates of each keyword in terms of table, column, and row. Given a query consisting of a set of keywords, DBExplorer returns results in three steps.

1. Using the symbol tables that store meta information, it will identify relevant tables, columns, and rows.
2. Among the obtained relevant tables, it further explores the space of joinable tables to capture denormalized tables that together contain all terms of a keyword query.
3. For all discovered and joined tables, an SQL statement is constructed to obtain the rows with the corresponding keywords.

In essence, DBXplorer's indexing phase transforms relational data into a keyword-accessible structure, trading some preprocessing time and storage for fast, schema-independent keyword search. They also explore the trade-offs for non-exact matches inside table entries, which has implication on how the tables should be indexed.

3.2.2.2 Discover

Discover is yet another keyword search system for relational databases that functionally mimics DBXplorer (Hristidis and Papakonstantinou 2002). The main differences lie in the design of the retrieval process and the enumeration of potential join tree candidates, which are graphs of tuples across joinable tables that contain the defined user keywords. The authors claim that their enumeration technique ensures minimality with regard to the size of the retrieved networks and avoids redundant computation.

3.2.2.3 BANKS

BANKS stands for *b*rowsing *an*d *k*eyword *s*earching and approaches the problem of keyword search over relational data as a graph search problem (Bhalotia et al. 2002). Similar to DBXplorer, BANKS enables keyword search and identifies tables that contain keywords as part of their schema or data. In addition, BANKS provides an interface to browse data and generate hyperlinks for foreign keys and other links, on displayed results. This way, BANKS enables a hierarchical and graphical representation of discovered results. By modeling the information as a hyperlinked graph, BANKS is able to apply the Pagerank (Brin and Page 1998) algorithm for assessing the relevance of the tuples retrieved.

3.2.3 Further Impact

Keyword search essentially represents the most fundamental operation for finding relevant tabular objects. In essence, more advanced search operations, such as UNION search and finding JOINABLE tables, are constrained variations of keyword search. Aspects, such as indexing tables and ranking relevant results (Liu et al. 2006) require similar thought processes. Further, keyword search can be seen as the gateway toward natural-language search over databases (Saha et al. 2016), which we will also discuss in Sect. 3.6.

3.3 Union Search

Union search aims at finding tables or columns that are in the extension of a given user table or column. In the literature, one also finds the term schema-complement (Sarma et al. 2012). In that regard, the user query is not only a list of keywords but rather a grid structure of keywords. Use cases for union search typically require to find more instance of the same shape. This can be end goal on its own, e.g., the goal might be to find all tables on the same topic across many divisions

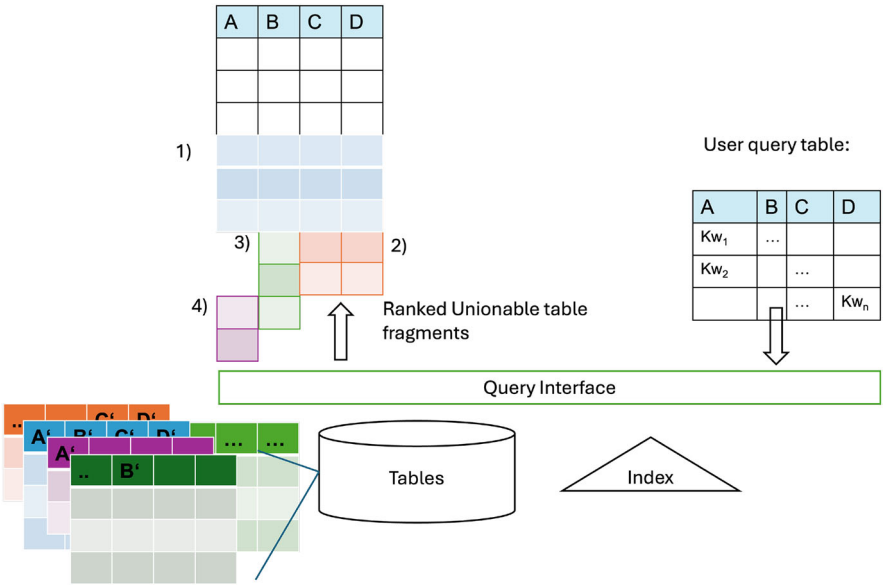


Fig. 3.2 Union discovery: The user query consists of a relational table or a set of columns. The system returns a set of table fragments that can be unioned with the input table

of an organization. The goal could also be to obtain evidence for a downstream model or for cross-checking a given table with tables of similar content for data cleaning.

As shown in Fig. 3.2, union search finds or synthesizes tables that can naturally extend the rows of the given user table. The figure illustrates that the result consists of either entire tables, as is the case for the blue table, column subsets, as in orange, or individual columns, as in purple and green. Each table fragment will be ranked based on a predefined unionability score. In this section, we will present existing heuristics for unionability scores. By doing so, we distinguish unionability on table and column level and how one influences the score of the other.

3.3.1 Semantics of Union Search

The UNION operator in SQL has very strict semantics. In order to be able to perform a union between two tables or projections, both resulting tables must project the same number of columns in an order so that two columns at the same position have the same data type. The UNION operator follows the set semantics, so that the resulting relation is stripped from all duplicate rows. In contrast, the UNION ALL operator follows the multi-set semantics.

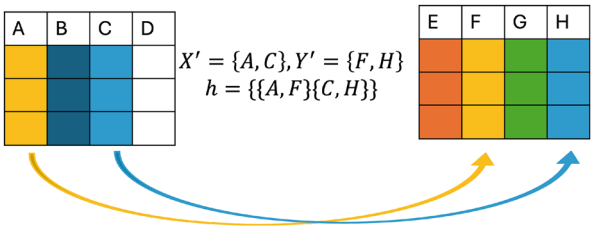


Fig. 3.3 Column alignment between two tables. Any two columns that display equal colors are unionable based on a chosen unionability score

In the context of data discovery, the search follows a maximally unionable principle, where the possibility that two tables are unionable receives a score. This score is calculated based on the number of *alignable* columns.

Definition 3.2 (Column Alignment) Given two sets of columns X and Y from two different tables, a column alignment is a one-to-one mapping: $h : X' \rightarrow Y'$ such that $X' \subseteq X$ and $Y' \subseteq Y$ and $|X'| = |Y'|$.

Definition 3.2 is applicable to any pair of tables, with most pairs resulting in a mapping where no column is aligned and the resulting unionability score, which we have not explicitly defined yet, will be zero. Figure 3.3 displays an example of a concrete alignment based on a given mapping. In this example, the attribute subsets $\{A, C\}$ from X and $\{F, H\}$ from Y constitute a maximally possible alignment where the bijective mapping function h maps $h(A) = F$ and $h(C) = H$. The choice of the mapping function and the search algorithm can be different depending on the use case and the type of datasets. In general, one can state that two tables are fully unionable if the mapping is defined as $h : X \rightarrow Y$.

Given the alignment function and the obtained score, unionability search translates in to a top-k unionability search algorithm, where for a given input table the top-k tables with the highest unionability score are returned to the user.

The unionability score of a table is implicitly defined by the unionability score of the individual columns. Here we refer to the definition provided by Nargesian et al. (2018):

Definition 3.3 The unionability score $U(h)$ of a column alignment $h : X \rightarrow Y$ is the joint probability of the unionability of attribute pairs, namely $U(A_i, h(A_i))$, in the alignment: $U(h) = \prod_{i \in \{1,\ldots,|X|\}} U(Ai, h(Ai))$

Note that the aggregation function here is a product. Yet, it is possible to choose alternative ways for aggregating the individual column scores.

3.3.2 Unionability of Columns

The unionability of a table directly follows from the unionability of the individual columns. Next, we discuss common heuristics for calculating the unionability of individual columns.

In standard SQL semantics, two columns with the same data type can be considered unionable. However, UNION discovery in data lakes is not aiming to find syntactically unionable columns but those that extend the content of a column within the extension of the original column domain. For example, the original dataset of a user might contain all movies from a certain studio. Here the union search approach aims at adding movies that have not been captured in the user dataset. As a result, the problem of finding unionable columns ultimately resembles a schema matching problem, which has been researched much earlier than union discovery (Bellahsene et al. 2011). Traditionally, one distinguishes label-based, instance-based, and structure-based matchers. Label-based matchers leverage features of column labels, such as plain edit-distnance or semantic similarity, to draw similarities across columns. Instance-based approaches leverage features of the actual values inside a column to define similarities. Such features could be as simple as the set overlap of values inside two columns or more abstract features based on value patterns. Finally, the locality of columns can be leveraged to identify similarities across columns of two tables or schemas (Melnik et al. 2002).

The main difference between union discovery and schema matching is that for union discovery not only the actual match between two columns matters but also to which extend one can expand the user-given dataset. One of the early approaches to define different unionability scores was by Nargesian et al. (2018). They distinguish three different similarities that inform on the unionablity of columns, set domain similarity, semantics domain similarity, and natural-language domain similarity.

3.3.2.1 Set Domain

Under set domain assumption, two columns are unionable if a significant portion of their corresponding values overlap. The ratio of the overlap should indicate that the values of both columns have been drawn from the same domain. This type of unionability heuristic works well in scenarios where redundancy and fragmentation is expected to happen. In general, it is possible to relax the strict value overlap to more approximate overlaps based on n-grams or patterns. This will generally increase the potential recall of a union search algorithm. Yet, the approach will fail to union columns that draw from the same domain but are based on disjoint characteristics of the population, e.g., columns with European countries vs. Asian countries.

3.3.2.2 Semantic Domain

An approach that is able to match columns without observing syntactical overlap of the values is to map column values to predefined external types or domains. In the simplest case, all values inside a column map to the same semantic type or category, which would simplify the union search problem to the discovery of columns with the same type. In reality, each column maps approximately to more

than one type. For example, one column might match to *Location*, *Birthplace*, and *Residency*, while the other maps to *Location* and *Headquaters*. The diversification of types shows here that although both columns contain information about cities they actually cover entirely different semantic subdomains, one about persons and the other about companies.

3.3.2.3 Natural-Language Domain

Finally, Nargesian et al. (2018) identify a class of columns that can come from the same domain but cannot be reasonably captured through ontologies or significant value overlaps. Those columns are nevertheless unionable if the relationship can be expressed informally. What they call natural-language association between domain values is suggested to be captured by language models. Naturally, capturing unionability in this setup requires to capture the latent similarity of column content in their occurrence of natural language. Early approaches used context-free embeddings, such as word2vec and fastText. Later approaches rely on more advanced foundation models that also capture additional context of column content (Fan et al. 2023).

The assumptions on column unionability likely depend on the type of dataset and lake and also the available metadata. Over the last few years, several methods have been proposed that excel at retrieving unionable columns based on approaches that try to cover multiple similarity aspects simultaneously or learning the union intent.

3.3.3 Union Search Prototypes

We now discuss three representative research prototypes that solve the unionability problem by either combining multiple scores or learning latent representations of columns.

3.3.3.1 Ensembling Unionability Scores

As previously mentioned, Nargesian et al. (2018) proposed a union discovery framework that compares the different domain assumptions for unionability. Further, they propose an ensemble method that leverages the scores that reflect unionability under each individual domain assumption to obtain an aggregated score for the unionability of a table.

Bogatu et al. (Bogatu et al. 2020) also propose D3L, an ensemble method that aggregates format similarity based on regular expressions derived for each column and the numerical distribution of values. D3L calculates the relatedness of two tables using the weighted average of the aforementioned similarity metrics. The weights are obtained based on the similarity distribution of the input columns and attributes in the data lake.

3.3.3.2 Semantic Graphs

SANTOS (Khatiwada et al. 2023) is a union search approach that relies on semantic annotations based on the semantic domain. SANTOS then uses a knowledge base to establish relationships between the identified types inside a table. This way, for each table, a semantic graph is generated that captures metadata not only on individual columns but also on how the columns relate to each other. To understand its benefit, consider an example of two tables: One table contains information about people, containing attributes on names and locations, etc. The other table contains information about artists and their residences and also columns on names and locations. If we only consider column similarities based on set overlap, semantic domain, or even natural-language domain, we might obtain a high confidence that the two tables are unionable. However, if one can establish through a relationship extraction routine that in the first table the location refers to the place of birth, while the location in the second table refers to the residence, one avoids this false association. SANTOS aims at exactly this by annotating tables with relationship information and then using an aggregated unionability score that also takes into account the correspondence of two tables with regard to the depicting column relationships.

3.3.3.3 Representation Learning

Starmie (Fan et al. 2023) aims at further improving over SANTOS through self-supervised learning so that it does not require predefined knowledge bases. In particular, it captures latent semantic types through contrastive representation learning (Chen et al. 2020), which is trained to learn how a column fits into a table. In other words, the representation of a column captures not only metadata about its own content and labels but also information about its co-occurrence with other columns of the same table. Through representation learning, it is possible to capture all this information in a latent space and represent each column as an individual vector, which has also implications on how tables may be stored.

In the end, the heuristics chosen for identifying useful unions can differ from dataset to dataset and data lake to data lake. In Chap. 6, we will discuss potential pitfalls in evaluating the usefulness and generalization of such approaches.

3.4 JOIN Search

While UNION search aims at discovering tabular fragments that vertically augment a given table, UNION search describes the task of identifying tables that horizontally, i.e., through Joins, augment a given table (Zhu et al. 2019; Fernandez et al. 2018). Analogous to schema complement for UNION search, there is the terminology of entity complement for JOIN search (Sarma et al. 2012). Typical use cases for JOIN

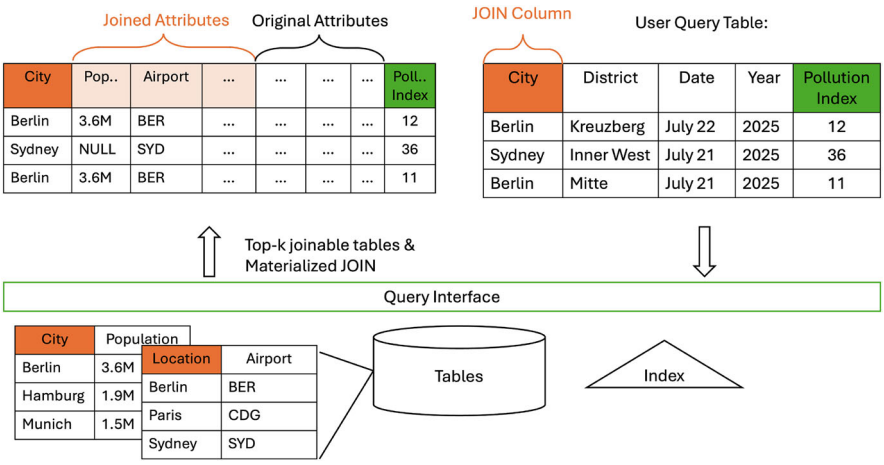


Fig. 3.4 Given a user table on Pollution index, JOIN discovery identifies tables with joinable columns. Columns of top-k joinable tables are appended through the materialized JOIN to the input table

discovery are entity augmentation (Yakout et al. 2012) and feature extraction (Santos et al. 2021). Entity augmentation aims at finding additional descriptive properties of given instances. Similarly, feature extraction aims at identifying properties that can serve as features for a downstream prediction task.

Figure 3.4 shows a simple example for JOIN discovery. A table fragment with information about cities and city districts associated with the corresponding particle dust pollution index is presented. The underlying use case is borrowed from the data science challenge of the BTW 2019 (Meyer et al. 2019; Esmailoghli et al. 2019) where the goal is to identify causes for high pollution. Through JOIN discovery, it is possible to associate the pollution index with additional aspects, such as colocated airports or population size. Note that the result will contain not only tables that JOIN on all JOIN column values but also tables that only partially cover them. For example, when joining the table on city populations, there is no joinable row for the JOIN key Sydney. Also note that some tables although properly joining with our input table might not provide useful columns to append.

Next, we will dive deep into different notions of joinability and how scores can be calculated. Then, we will discuss how the usefulness of the aligned columns, such as *population*, and *airport code*, is assessed in existing systems. Finally, we present a set of representative systems for JOIN discovery.

3.4.1 Joinability

In contrast to UNION discovery, it is not enough to merely find columns that fit with regard to the domain, because joinability aims at finding joinable rows. To assess how well a table joins with a given input table, the coverage of values in the chosen JOIN column or columns of the input table is decisive. The more values find a JOIN partner in a discovered candidate tables, the higher the joinability of that candidate table will be. The literature generally distinguishes exact joins and semantic joins. For both JOIN types, the variation of multicolumn joins exists.

3.4.1.1 Exact JOINs

The most basic joinability score $S_j(X, Y)$ between two columns X and Y can be calculated by measuring the set overlap between two columns. This translates into the well-known Jaccard similarity (Charikar 2002). Figure 3.5 shows three example columns *City*, A, and B, from an user input table, and two candidate tables T_1 and T_2. Using the Jaccard distance

$$d_{Jaccard}(X, Y) = \frac{|X \cap Y|}{|X \cup Y|},$$

we obtain the scores

$$d_{Jaccard}(City, A) = 0.5$$

and

$$d_{Jaccard}(City, B) = 0.42.$$

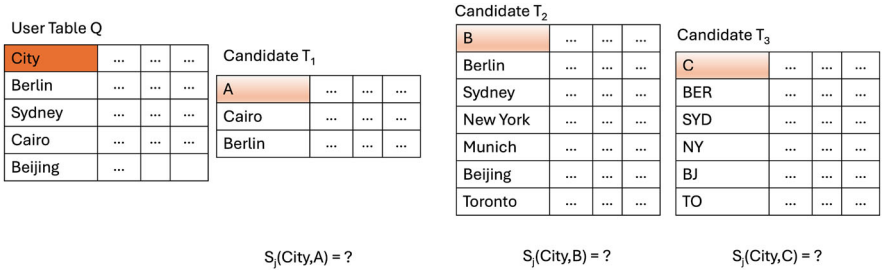


Fig. 3.5 The Jaccard similarity between City and A is higher than the similarity between City and B, but City finds more joins in B. Column C cannot be directly joined but requires first a transformation

Thus B would be a more suitable JOIN candidate. Now, note that Jaccard distance does not distinguish between input and candidate tables. Yet, as users, we are interested in tables that JOIN on as many values as possible from our input table. With this requirement, column B would actually be more suitable because it would JOIN with two more rows from the input table than A. To reflect that in the score, we need to adapt the joinability score slightly by changing the denominator (Zhu et al. 2016) from $|X \cup Y|$ to $|X|$ resulting in the set containment score

$$S_j(X, Y) = \frac{|X \cap Y|}{|X|}.$$

With this ratio, $S_j(City, A) = 0.5$ and $S_j(City, B) = 0.75$ clearly favoring table C_2 with JOIN column B.

3.4.1.2 Semantic or Approximate JOINs

Some lines of research argue that due to the prevalent heterogeneity in data lakes, the search for tables that satisfy the premise of exact joins will yield little recall, i.e., in most cases, not many joinable tables will be found (Santos et al. 2022). Without loss of generality, one can make this case certainly for a number of data lakes. In such scenarios, a relaxation of the exact value overlap premise is desired, which can be achieved by incorporating a corresponding value similarity score into the joinability score S_j. Another reason to focus on approximate JOIN discovery is to avoid the row-by-row materialization of the join, which can negatively affect the response time of a discovery query. Thus, instead each column is represented by a summary, and the similarity over the summary of two columns is used as a proxy for joinability. Such summaries are either hash-based methods (Fernandez et al. 2018; Santos et al. 2021, 2022) often based on local sensitive hashing variants or column embeddings as produced through neural architectures (Dong et al. 2023; Hu et al. 2025).

Consider again the example presented in Fig. 3.5. So far, we ignored Candidate T_3 because the set-based joinability scores would not capture any possible JOIN candidates on the only JOIN column C. Yet, with some domain knowledge, it is possible to virtually align the city names in *City* with the codes in C. While this opens up more opportunities for connecting columns, the approach would require approximate retrieval for identifying that the two columns can be semantically joined. Further, apart from identifying that the two columns may be joined, one has to also correctly align the values from each row, which is now less trivial because of the non-exact matches.

In some respects, the approximate joinability score is similar to the learned scores used for unionability. At a high level, both approximate joinability and unionability search reduce to identifying related columns. However, their representation design differs in two key aspects. First, joinability requires representations that approximate the feasibility of joining on individual column values, which is not relevant for

unionability. Second, for unionability, the broader column context tends to be more influential: A column is more likely to be unionable with a column from another table if its neighboring columns within the same table are also likely to be unionable with columns in the other table. The same consideration is not necessary for joinability. It might even be counterintuitive as it might lead to JOINs with tables that contain redundant information or self-JOINs.

3.4.1.3 Multicolumn JOINs

So far, the scores we discussed were applicable for situations where JOINs are only performed on single columns. There are situations, where JOIN discovery might aim at discovering tables that join on more than a single column of the input table. A typical scenario for multicolumn JOIN search is to look for event data that aligns with location and date. Here we can refer again to our example from Sect. 1.4, where to explain particle dust spikes in European cities, Esmailoghli et al. used longitude/latitutde and date information to find evidence from data lakes (Esmailoghli et al. 2019). As a result, it was possible to align spikes with well-known events and busy crossings. On a different note, common benchmarks, such as TPC-H, display many multicolumn primary key/foreign-key relationships (Esmailoghli et al. 2022).

To serve multicolumn JOIN discovery, there are several options to adapt existing techniques. A simple adaptation of the single-column score $S_j(X, Y)$ would be to first obtain the joinable table for each column independently and then intersect the result sets to obtain those tables that contain joinable columns for all given JOIN keys. This approach is straightforward for the exact JOIN premise, as the desired results by definition have to be inside the intersection. For approximate JOIN discovery, the intersection might be too restrictive as the nature of approximation might allow for including rows that only join on subsets of the JOIN column set from the input table. Here more holistic assessments might be of interest that assess the overall similarity of a value combination against another value combination.

3.4.2 Utility Augmented Joinability Scores

So far the discussed joinability scores merely describe how well two tables align on the selected JOIN columns. They are not capturing the information, as to which extend the resulting JOIN would be useful for a task. Admittedly, a JOIN with a very low joinability score is probably not useful, as the resulting table will be very small. The argument to be made here is that even a candidate with high joinability score might not be useful because it is not providing relevant information for the downstream task.

One of the well-researched tasks in the context of data discovery is dataset augmentation for machine learning (Cappuzzo et al. 2025). The idea is that a given table does not contain enough features, and through discovery of joinable tables,

one can obtain for information that can improve the accuracy of the downstream machine learning model (Esmailoghli et al. 2021; Santos et al. 2022; Chepurko et al. 2020).

To assess the usefulness of the discovered candidates, one needs to incorporate a corresponding score. There are different ways to enact such scores. A straightforward solution is to invoke the usefulness score after a relevant set of candidates has been discovered. Automatic Relational Data Augmentation (ARDA), for example, first joins all discovered joinable tables and then uses feature selection strategies to identify relevant features within those tables (Chepurko et al. 2020). The disadvantage of this approach is that the computationally heavy step of feature selection has to be performed on a potentially large set of features. A more computationally viable solution would require that the discovery process already prunes irrelevant candidates to relieve the feature selection process. Cocoa, for example, uses correlation calculation to identify and sort correlating columns in discovered tables (Esmailoghli et al. 2021). Also, for Cocoa, the two steps are sequential. Thus there is no trade-off consideration between the usefulness of a feature and the joinability of the corresponding table. Another approach based on sketching encodes the information about joinability and correlation within a single score (Santos et al. 2022, 2021).

3.4.2.1 "Thinking Twice About Joins"

In a paper with a title "To Join or Not to Join?: Thinking Twice about Joins before Feature Selection," Kumar et al. analyze a standard pipeline for learning over relational data in which a key–foreign-key (KFK) JOIN is used to denormalize a fact table with attributes from a referenced dimension table prior to feature selection (Kumar et al. 2016). They argue that many such joins are often unnecessary for predictive accuracy because the foreign key functionally determines the joined attributes, so the JOIN can be redundant from an information-theoretic point of view. However, they also characterize when joins can still help in practice: With finite training data, replacing a potentially low-dimensional set of informative foreign attributes with a high-cardinality foreign-key feature can increase model variance and overfitting risk. To assess the usefulness of a JOIN *a priori*, they propose conservative decision rules based on Vapnik–Chervonenkis-dimension generalization bounds, including a *Risk of Representation* (ROR) heuristic that upper-bounds the added generalization risk from using the foreign key as a proxy and a simpler *Tuple Ratio* (TR) rule that depends only on the ratio of training rows to the number of distinct foreign-key values. In practical terms, a JOIN is *not useful*, i.e., can be safely avoided, when these heuristics indicate low added variance risk. This is most notably the case when there are many training examples per foreign-key value (high TR), which preserves test accuracy while reducing feature-selection cost and overall pipeline complexity.

3.5 Operation Sequences and Compositions

In practice, users rarely rely on a single operation for their information needs. Consider the example depicted in Fig. 3.6. In this example, the user starts of the discovery process with two keywords to obtain a first table fragment that contains useful information. Then, the user wants to expand the dataset through UNION into a larger dataset. Meanwhile, being aware that the UNION operation might result in a sparse table, the user also wants to make sure that missing values are imputed with information from other tables inside the lake. Finally, the table is expanded horizontally with more columns through JOINs to serve for a downstream inference task. Catering this sequence of tasks not only requires the integration of multiple operators within a system but also opens questions regarding complex query languages and optimizations. In Chap. 4, we also address the associated problem of unified indexing. For now, we will discuss the challenges regarding the invocation and execution of operation sequences.

3.5.1 Operation Sequences and Grammars

While the majority of existing data discovery research has focused on building stand-alone solutions for individual operations, there are already prototypes that integrate several operations into a single unified discovery system (Esmailoghli et al. 2025; Fernandez et al. 2018). In such systems, a well-defined *collection of operators* is supported that are applied together as part of a single discovery workflow. We refer to such workflows as *operation sequences*, emphasizing that discovery capability is defined not by any one operator, but by the way multiple operators are composed within the system.

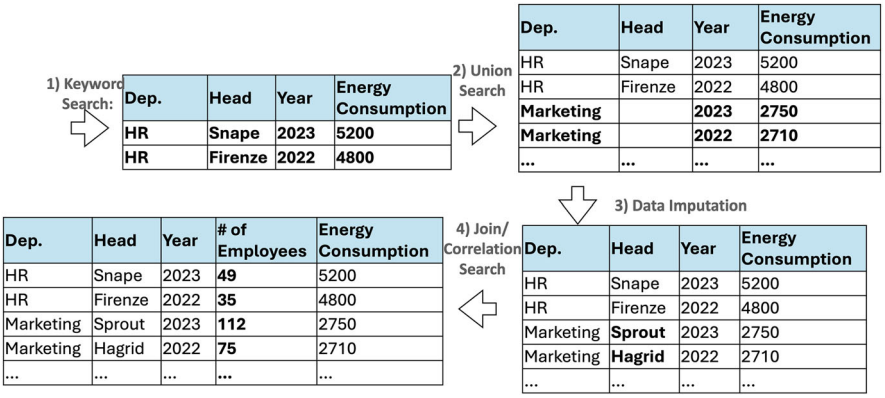


Fig. 3.6 The user performs a series of discovery operations to obtain the table they need

From this perspective, a discovery system is characterized by the *set of operator compositions it supports*. For example, a system may allow keyword search followed by schema-based filtering and join discovery, but not task-aware ranking over the results of join discovery. Another system may support keyword search combined with distribution-aware filtering, but only as a single composite operation rather than as independently invocable steps. These choices determine the expressive power of the discovery system as a whole. Operation sequences encode implicit assumptions about what intermediate information can be produced and consumed. When multiple operators are composed, the output of the composition must be representable in a form that is meaningful to all operators involved. This requirement constrains both operator design and system architecture. For instance, composing keyword search with distribution-aware predicates presupposes a shared representation that links textual metadata to distributional summaries. If such a representation does not exist, the composition itself becomes ill-defined, even if each operator is well-defined in isolation.

Compositions also determine how aggressively information is abstracted. Some operator collections are designed to be *lossy by construction*, collapsing datasets into coarse representations early in the workflow. Others preserve richer structure to support downstream reasoning. These design choices affect not only discovery accuracy but also which classes of queries the system can meaningfully support. In this sense, operation sequences act as a form of *capability boundary*: Queries that fall outside the supported compositions cannot be expressed, regardless of the underlying data.

3.5.1.1 Source Retrieval in AURUM

AURUM was one of the first systems to enable the usage of multiple discovery operations via a discovery language (Fernandez et al. 2018). Its discovery language, SRQL (Source Retrieval Query Language), interacts with the so-called Enterprise Knowledge Graph (EKG), which is a hypergraph that connects tables and columns based on their similarities.

The language is at column granularity, allowing users to view and retrieve results at columns and tables. Each such object is considered a discovery element (DE), which can be composed to more complex DE. The same idea of composability applies to the supported set of operations that are called discovery primitives (DP) and take as input a set of DEs and output another set of DEs.

Functionally, SRQL supports a range of discovery intents through primitives for schema/name similarity, content similarity, e.g., set-/range-/vector-based similarity, and joinability, including discovering join paths between datasets with bounded hops. These primitives can be combined using set-algebra operators, such as union/intersection/difference and "views," e.g., lifting column-level results to the table level, to express richer retrieval logic. Results are returned as a Discovery Result Set (DRS) that retains a provenance graph for explanation and validation

and supports diverse ranking strategies, such as certainty-based or coverage-based, without re-executing the query.

Consider the following example of a query in SRQL:

```
Q = table(
      joinPath(
        schemaSearch("customer"),
        contentSim(schemaSearch("email"))
      )
    )
```

In this example, schemaSearch("`customer`") and schemaSearch("`email`") are sets of *discoverable elements* (DEs), i.e., column-level nodes retrieved from the enterprise knowledge graph. The operators schemaSearch(·), contentSim(·), joinPath(·, ·), and table(·) are *discovery primitives* (DPs). The joinPath primitive yields a *discovery result set* (DRS) that includes both the discovered results and their provenance. Finally, table is a view that lifts the column-level results into table granularity, producing the final DRS returned by the query.

3.5.1.2 Search Plan Optimization in BLEND

BLEND is a more recent holistic discovery system that provides an API for the formulation of so-called *search plans*, which are compositions of low-level discovery operations (Esmailoghli et al. 2025). A *search plan* in BLEND is a declarative specification of a discovery task. The user constructs a `Plan` by adding *seekers*, which are atomic discovery operators such as keyword, single-column join, multicolumn join, and correlation seekers, and wiring their outputs through *combiners*, which are set operators such as intersection, UNION, difference, and counting. This way, a plan can represent anything from a single operator call to a multistage discovery pipeline, with each operator parameterized by its inputs and a top-k bound.

Operationally, BLEND compiles the plan into a directed acyclic graph (DAG) in which nodes are seekers/combiners and edges represent data flow and then derive an optimized execution plan. The optimizer exploits reorderable operator groups, e.g., around commutative intersections, ranks seekers using rule-based type priorities and learned cost estimates, and applies query rewriting that injects predicates (e.g., `TableId` $\in (\cdot)$ and `TableId` $\notin (\cdot)$), so intermediate results constrain the search space of subsequent operators, improving end-to-end runtime while preserving the plan semantics.

```
1  def build_search_plan(keywords, examples, queries):
2    plan = Plan()
3    # Keyword Search
4    plan.add('kw', Seekers.KW(keywords, k=10))
5    #\unionSearch
6    for clm in examples.columns:
7      plan.add(clm, Seekers.SC(examples[clm], k=100))
8    plan.add('counter', Combiners.Counter(k=10), examples.columns
       )
9    # Data Imputation
10   plan.add('examples', Seekers.MC(examples, k=10))
11   plan.add('query', Seekers.SC(queries, k=10))
12   plan.add('intersection', Combiners.Intersect(k=10), ['
       examples', 'query'])
13   # Correlation Search
14   plan.add('correlation', Seekers.Correlation(examples.joinkey,
        examples.target, k=10))
15   # Results Aggregation
16   plan.add('union', Combiners.Union(k=40), ['kw', 'counter', '
       intersection', 'correlation'])
17   return plan
```

Listing 3.1 Building search plan in BLEND.

Listing 3.1 depicts how a plan for the introductory example from Fig. 3.6 could look like. In essence, the user develops a `Plan()` object by adding the relevant seekers and combiners. The `Plan()` is initialized on Line 2. On Line 4, the keyword seeker is added to the plan. Next, a UNION search sub-plan is created by adding one single-column (SC) seeker per input column (Lines 6–7), followed by the *Counter* combiner (Line 8), which tracks the frequency of found columns to identify the most relevant tables. Next, the data imputation sub-plan is added by using a multicolumn (MC) seeker (Line 10), and for rows where only values of a single column exist, an SC seeker is used (Line 11). The results of both seeker types are combined on Line 12 with an intersection combiner. Similarly, the correlation seeker is used to identify correlating features (Line 14). Finally, the results of each sub-plan are aggregated via a UNION combiner (Line 16).

Figure 3.7 illustrates the DAG representation of this plan, which already highlights hyper-nodes that correspond to so-called execution groups (EGs). Reordering seekers within an EG does not change the output of the EG. The optimizer follows a topological order based on the dependency of the EGs and processes one EG after another. An EG is selected only after all its dependencies are executed. Once the next EG is selected for execution, BLEND optimizes the sub-plan of the EG before executing its seekers in the database.

While systems, such as AURUM and BLEND, already provide rich languages for formulating discovery queries, each has clear limitations. AURUM does not natively support optimization, and BLEND's optimization logic so far does not support trading-off approximate discovery results with runtime. Another clear path forward toward more usable ways of expressing data discovery operation sequences is to

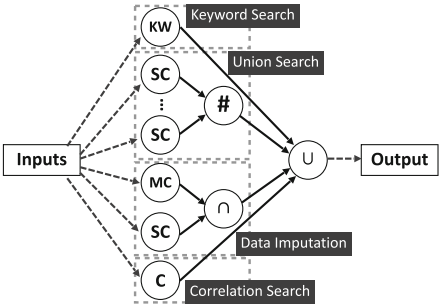


Fig. 3.7 DAG representation of the discovery plan for Listing 3.1

support natural-language discovery tasks and enable semantic parsing of discovery requests into actionable operators.

3.5.2 Domain-Specific Languages for Data Discovery

A fundamental limitation shared by the previously mentioned systems is that their query languages are *system-driven*: The syntax and semantics of each language are defined in terms of the internal data model of the underlying discovery system rather than in terms of the user's information needs. As a consequence, users must learn a new set of primitives whenever they switch systems, and the languages themselves are difficult to extend or formally reason about because they lack a semantic foundation.

Kang and Galhotra (2024) address this limitation through a *language-driven* approach to data discovery, taking inspiration from programming languages research. Their key observation is that most meaningful data discovery queries can be decomposed into two primitive classes: *transformation clauses*, which characterize how raw data is to be modified or enriched, and *constraint clauses*, which specify properties the returned datasets must satisfy. Extending philosophically from the Curry-Howard correspondence (Bellin et al. 2001), they identify that transformation and constraint clauses share behavior analogous to operations and types in a typical programming language, motivating a natural encoding of discovery queries into a typed language.

Semantic Foundations

The semantic backbone of the system is *Imperative Relational Algebra with Types* (ImpRAT), an algebraic language that extends standard relational algebra with a type-like constraint system. ImpRAT expressions consist of sequences of statements that evaluate either to a result or to a failure value, analogous to runtime errors in a typed programming language. Standard relational operators are augmented with a test operation that allows statements to branch or fail based on data properties. The semantics of an ImpRAT expression are defined inductively over its statement structure, providing a formal execution model against which discovery

query correctness can be reasoned about independently of any particular system implementation.

The TQL Language
Built on top of ImpRAT, TQL is a domain-specific discovery language whose queries are composed from two building blocks: *signatures*, which are predicates describing the search action to be applied to the repository, and *expressions*, which are formed from values, attributes, and operators over those values. A TQL query associates a type annotation encoding the downstream transformation context directly with the discovery request. This means that the analyst does not merely specify what schema properties a candidate table should have but also declares how the discovered table will be used, for instance, whether a particular column will serve as a join key, a grouping variable, or an outcome measure in a downstream task. The discovery system can use this downstream context to rerank candidates: A table that satisfies the schema constraint but whose relevant column has properties incompatible with the declared downstream transformation is penalized relative to the one whose column is well-suited to that use.

The formal semantics of TQL are defined via a translation from TQL surface queries into ImpRAT expressions, giving a full compositional account of query evaluation. This translation is defined by a BNF grammar and a set of typing rules.

3.6 Natural-Language Queries

Keyword search over a tabular data lake is, at its core, a way to *discover* data. You type a few terms, “revenue,” “Germany,” and “2024,” and the system responds by surfacing tables, columns, and perhaps a handful of representative rows that contain those strings or close variants. Natural-language question answering expands that discovery workflow into an end-to-end capability that can interpret a question, decide what data is needed, compute the result, and explain what it did. The shift is less about replacing keyword search and more about layering additional competencies on top of it. In a mature system, question answering still begins with retrieval, often the same catalog search and indexing machinery. In contrast to keyword search, it must be able to go beyond finding relevant tables and either provide tables that likely provide the answer to the question or even formulate a human readable answer. A question might be “what was Germany’s revenue last year?”, which will be interpreted with regard to explicit intent and relevant concepts. This semantic grounding is the first major expansion beyond keyword matching. The grounding is either indirectly through named entity recognition plus keyword retrieval or directly through latent embedding of the question into a shared embedding space of questions and tables.

Once the system has a plausible interpretation, it needs to plan the operations that actually produce an answer. Keyword search returns candidate tables, and a question answering agent then chooses the right tables, reconciles their shapes, and reasons

about relationships. In a data lake, the data that answers the question might not be in one place. Considering our example, "Revenue" might live in an invoices table, geography in a customer dimension, and the definition of "recognized revenue" in a finance model downstream of raw transactions. The system has to find join paths, detect keys, and avoid common pitfalls like grain mismatches, joining a monthly summary to daily transactions, or aggregating at the wrong level. This planning step turns "search" into "query construction," typically producing SQL or a logical plan that can be executed in the lake's compute engine (Pourreza and Rafiei 2023; Zhang et al. 2025; Wang et al. 2024).

Question answering introduces a different layer of complexity for a discovery system, namely accountability. A search engine can be useful even when it is imperfect, because the human can inspect the results. A question-answering system has to be correct enough to trust and transparent enough to debug. Practical systems need to include sanity checks, compare alternative interpretations when multiple columns could match "Revenue," and attach provenance, so the user can see exactly which datasets and fields were used to produce which answer.

In this sense, natural-language question answering is best understood as keyword search plus three additional layers: semantic understanding, structured reasoning, and verified execution. So far there are two ways that question answering and data discovery connect. The first approach is a direct continuation of keyword search, where a question is interpreted as a coherent question instead of being transformed into a set of keywords. The result is initially still a ranked list of tables. The second approach focuses on generating an answer from a given set of tables. We coin this approach multi-table question answering. The scale of tables is typically small and reduced to an already preselected set of tables.

3.6.1 Natural-Language Questions Over Large Repositories

Pneuma (Balaka et al. 2025; Balaka and Fernandez 2026) is an end-to-end *table discovery QA* system where a user's natural-language question Q is answered by producing a *ranked list of relevant tables*, i.e., "answer tables." Similar to previously discussed discovery systems, Pneuma is split into an offline representation/indexing phase and an online query processing pipeline. Offline, the *Data Register* ingests tables, e.g., from databases/CSVs, along with any available free text, i.e., *table context* that augments the downstream table representation with additional semantic context. The *Table Summarizer* then converts each table's content into a small set of textual content summaries that preserve schema information and row-level evidence. These summaries and contexts are treated as documents and passed to the index builder that embeds them into a vector space.

At query time, Pneuma produces the final ranked answer tables by combining outputs from three components. A *Lexical Retriever* based on BM25 (Robertson and Zaragoza 2009) outputs candidate documents with lexical relevance scores, while a *Semantic Retriever* outputs candidates via vector search over embeddings

of content summaries and contexts. *Hybrid Search* merges candidates and computes a combined score

$$s(d) = \alpha\, s_{\text{lexical}}(d) + (1 - \alpha)\, s_{\text{semantic}}(d),$$

for each document d. These scored (Q, d) pairs are then judged for their relevance. Pneuma uses these judgments to reorder results by pushing items judged irrelevant to the end while preserving the score-based ordering among those judged relevant. Finally, the system returns the tables associated with the top-ranked documents as the QA output. Disregarding the internal way of how tables are internally represented, the overall discovery workflow resembles the original approach for keyword search. The production of an answer can at this point be performed via other techniques that are designed for the purpose of answer generation and follow strategies, such as chain of tables (Wang et al. 2024; Zhang et al. 2025) and semantic parsing (Pourreza and Rafiei 2023).

3.6.2 Multi-table QA

While there is a large body of work on question answering on individual tables (Wang et al. 2024; Zhang et al. 2025; Pourreza and Rafiei 2023), the topic of producing answers based on large table repositories that require discovery has received little attention. Novel Retrieval-Augmented-Generation (RAG) paradigms (Yu et al. 2025) facilitate this approach connecting dataset search in large repositories with tabular question answering.

MultiTabQA (Pal et al. 2023) formulates *multi-table question answering* as a sequence-to-sequence generation problem: Given a natural-language question Q and a set of N input tables $T^N = \{t_1, \ldots, t_N\}$, the model must execute the implicit chain of relational operations, e.g., joins, set operations, aggregations, induced by Q over T^N, and generate the answer as a table T_{out}. A central design choice is that the system always emits a table, scalars become single-cell tables, list answers become single-row/column tables, and it also generates column headers to expose the semantics of the produced values.

Methodologically, MultiTabQA's QA behavior is enabled by a *linearized multi-table encoding* and a *structured table decoding* scheme. The encoder input concatenates the question with a serialized representation of each table, where every table is flattened in row-major order and bracketed by special keywords that preserve structure: `<table_name>` introduces the table name tokens (supporting disambiguation in multi-table settings), `col:` introduces the header sequence, and each row begins with `row i:` with columns separated by `|`. The decoder produces the answer using the same row-major flattening scheme (without a table name), emitting `col:` headers followed by rows with `row i:` and `|` delimiters, so the model learns not only which cells to output but also row/column alignment and header, value associations as part of the generated sequence. The QA model

is an autoregressive transformer encoder–decoder based on TAPEX (Liu et al. 2022) trained end to end to mimic a semantic parser/executor without emitting an intermediate logical form at inference time. Instead, the latent reasoning is learned implicitly through generation.

Such novel ecoder/decoder architectures do generally not scale to the large size of a data lake as all relevant tables need to be serialized and encoded during runtime. Thus building methods that interleave the potentials for table retrieval and question answering are currently a matter of active research.

3.7 Operations in Distributed Repositories

Modern data-driven systems increasingly operate in distributed and federated environments, where data is owned, governed, and stored by multiple independent providers. As introduced in Chap. 2, these settings impose structural constraints on visibility: Datasets may expose only metadata, statistical summaries, or curated samples rather than raw tuples. In such settings, the goal of data discovery to identify relevant datasets for analysis, learning, or decision-making must be rethought under constraints of privacy, autonomy, heterogeneity, and limited access. This section studies the operational foundations of data discovery in distributed settings, with a particular focus on federated data discovery.

3.7.1 Discovery Operations in Federated Environments

Federated settings raise fundamental questions about which discovery operations remain feasible and how their semantics must change when raw data is unavailable. The access function α introduced in Chap. 2 is the central determinant: operations that require $\alpha(T)$ = full degrade or become inapplicable when providers expose only metadata or summary statistics. We organize the discussion around how each major operation class adapts to these constraints. Figure 3.8 summarizes the feasibility and degradation behavior of all three operations across visibility levels, with the following subsections providing the technical details behind each cell.

Keyword Search Keyword search is the most naturally portable operator to federated settings, since it can operate entirely over dataset titles, descriptions, and schema annotations without requiring access to tuple content. Most federated discovery systems, including Google Dataset Search (Brickley et al. 2019), implement keyword search (Zhang and Balog 2018) as their primary discovery mechanism precisely because it is compatible with strict visibility constraints. However, as noted in Sect. 3.2, this makes keyword search inherently shallow in the federated case: It cannot capture distributional properties of column content, detect value-level overlaps, or assess suitability for downstream analytical tasks.

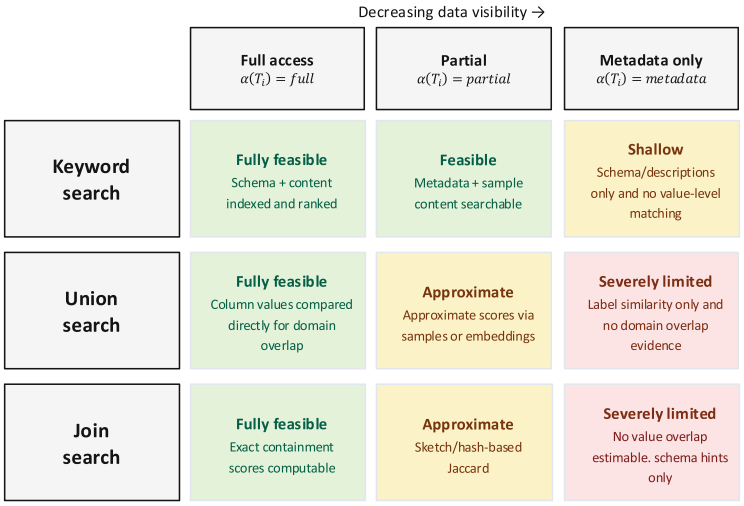


Fig. 3.8 Feasibility of keyword, union, and join search across decreasing data visibility levels

Two datasets that share a natural join key on county identifiers will not be surfaced as related unless their metadata descriptions use compatible terminology, a limitation identified directly by the designers of Google Dataset Search (Brickley et al. 2019).

Distribution-Aware Search Distribution-aware search generalizes metadata-based discovery by treating data distributions as first-class searchable objects. Rather than searching over textual descriptions alone, a distribution-aware system associates each dataset with summary representations, such as histograms, quantile sketches, or count-min sketches that approximate the underlying value distribution without disclosing individual records. Queries are then expressed as predicates over these summaries rather than over raw tuples.

The central query primitive in this paradigm is the *percentile predicate*, defined formally as follows.

Definition 3.4 (Percentile Predicate) Let T_i be a dataset with column c, and let $H(T_i[c])$ be a histogram synopsis of that column. A percentile predicate $\mathcal{P}_{c,\,f,\,\oplus,\,h}$ holds for T_i if and only if the fraction of values in $T_i[c]$ falling within $(-\infty, h)$ stands in the relation $\oplus \in \{<, \leq, >, \geq\}$ to the threshold $f \in (0, 1]$. Formally,

$$\mathcal{P}_{c,\,f,\,\oplus,\,h}(T_i) \;:=\; \frac{\left|\{v \in T_i[c] \mid v < h\}\right|}{|T_i[c]|} \;\oplus\; f.$$

In federated settings where $\alpha(T_i) < full$, the predicate is evaluated over $H(T_i[c])$ rather than raw values, introducing approximation error whose magnitude depends on histogram granularity and, where applicable, differential privacy noise.

For example, the predicate "at least 30% of patients are younger than 40" can be evaluated against a histogram without accessing any raw records, by summing the density of bins falling within the specified range. Composing multiple such predicates allows users to specify rich distributional requirements, for instance, simultaneously constraining both tails of an age distribution and making this query model substantially more expressive than keyword search alone.

A query Q in the distribution-aware setting is a Boolean combination of keyword predicates and percentile predicates. This allows analysts to narrow a repository first by topic, for instance, filtering for datasets whose profile contains the keyword "census" and then by distributional content, retaining only those that satisfy the required coverage thresholds. Evaluating such queries without raw data access requires providers to share dataset profiles containing column synopses. Because providers control the granularity of these synopses, the expressiveness of discovery is bounded by the coarsest summary any provider is willing to release. Finer synopses support more precise predicate evaluation but expose more information about the underlying distribution, creating a direct tension between discovery utility and privacy. Behme et al. (2024) and Esmailpour et al. (2025) study this setting in depth, the former proposing practical query evaluation algorithms over heterogeneous histogram collections and the latter establishing theoretical bounds on what is achievable under each visibility level.

Join Search Join search is the operation most severely constrained by federated settings. As discussed in Sect. 3.4, joinability fundamentally requires assessing value-level overlap between columns: The set containment score $S_j(X, Y) = |X \cap Y| / |X|$ is defined over actual column values, and its estimation requires either direct inspection of tuples or a summary that faithfully approximates value-level intersection.

In federated settings where $\alpha(T_i) \in \{partial,\ metadata\}$, neither condition holds. Schema-level metadata can indicate that two columns share a declared type or a natural-language label suggestive of a common domain, for instance, both described as "county identifier," but it cannot establish that their actual value sets overlap sufficiently to produce a meaningful join. Statistical summaries such as cardinality estimates and value range bounds provide weak signals: Two columns that cover the same numerical range may still be entirely disjoint at the value level.

A natural relaxation is to replace exact value overlap with an approximate joinability score derived from column summaries. If providers are willing to release column sketches, the Jaccard similarity between two columns can be estimated without exposing individual values, and join candidates can be ranked by estimated overlap. The accuracy of this estimate depends on the fidelity of the summary and the actual value distribution of the column. However, this approach still requires providers to disclose a form of column-level summary beyond bare metadata, which may itself be subject to governance constraints.

A more principled alternative for bilateral join discovery is secure multiparty computation (MPC) for private set intersection (Kamara et al. 2014), discussed in Sect. 2.3. MPC allows two organizations to compute the cardinality of their

column intersection without either party seeing the other's actual values, yielding an exact joinability estimate under full privacy. However, this approach is inherently bilateral, it requires both parties to agree to participate and execute a joint protocol, and its computational cost scales poorly to the bulk column comparisons required by open-ended join search over a large repository. As a result, MPC-based join discovery remains practical only for targeted scenarios where the candidate join partner is already known, not for discovery itself.

3.8 Additional Data Discovery Constraints

Beyond the choice between centralized, federated, and hybrid architectures, real-world discovery systems must operate under additional environmental constraints that fundamentally shape what discovery operation is possible. Among these, privacy requirements and data modality play a decisive role.

3.8.1 Privacy-Constrained Data Discovery

Example 3.1 Consider a consortium of hospitals collaborating to identify datasets relevant for predicting patient readmission risk. Each hospital holds sensitive patient records governed by strict regulations (e.g., HIPAA). While hospitals are willing to participate in discovery, they cannot expose raw data, detailed statistics, or even exact schema semantics.

In this setting, centralized discovery is infeasible. Even federated discovery over rich summaries may violate policy. Discovery must operate over privacy-preserving views of the data.

What distinguishes privacy-constrained discovery operations from traditional access control is that privacy is not binary. Instead of datasets being either visible or hidden, providers often allow partial visibility through carefully chosen representations. Metadata may be permissible, while value distributions are not. Coarse aggregates may be allowed, while fine-grained statistics are prohibited. As a result, discovery becomes a problem of reasoning over privacy-aware abstractions of data rather than data itself.

This shift has deep implications. Discovery operators that rely on inspecting values, such as join discovery or task-aware ranking, must either be redefined to operate over weaker signals or abandoned altogether. Even seemingly innocuous operations, such as ranking datasets by relevance, can leak information if repeated queries gradually reveal sensitive patterns. Privacy thus ceases to be a constraint layered on top of discovery; it becomes an integral part of the system's semantics.

Saibot One way to formalize privacy in discovery systems is to adopt differential privacy as a core design principle rather than an afterthought (Huang et al. 2023).

Instead of deciding ad hoc which statistics can be shared, data providers release representations that come with explicit, mathematically defined privacy guarantees. The discovery system then operates entirely within this protected space.

The platform Saibot exemplifies this approach. In Saibot, data providers choose privacy budgets and expose differentially private views of their data, enabling discovery and even downstream machine learning tasks without direct access to sensitive records. From the user's perspective, datasets remain searchable and usable, but every interaction consumes a portion of the privacy budget.

This fundamentally alters the nature of discovery. Queries are no longer deterministic; repeated searches may yield slightly different results due to injected noise. Rankings become probabilistic rather than absolute. Moreover, discovery is no longer free: Expressive queries are more expensive in terms of privacy loss than simple ones. The system must therefore balance discovery utility against long-term privacy sustainability.

From an operational standpoint, differential privacy transforms discovery into a budgeted process. The system must track privacy expenditure over time, coordinate queries across users, and design operators that degrade gracefully under noise. While this model imposes stricter limits on expressiveness, it provides clarity and composability—qualities that are essential in highly regulated domains.

3.8.2 Multimodal Data Discovery

The discovery operations discussed so far, keyword search, union search, join search, and their federated variants, share a common assumption: The data lake consists primarily of relational tables. Real-world enterprise data lakes, however, are considerably more heterogeneous. Modern organizations accumulate data across many formats simultaneously: structured tables holding factoid information, documents with rich contextual prose, graphs encoding symbolic knowledge, images, videos, audio streams, and spatiotemporal sensor signals. In such settings, answering a discovery query may require coordinating across modalities that no single relational operator can bridge.

This heterogeneity introduces two distinct challenges for discovery operations. The first is *cross-modal relevance*: identifying that a text document, a graph, and a table all bear on the same information need, even though they inhabit different representation spaces with no shared schema. The second is *cross-modal joinability*: establishing that records across modalities refer to the same real-world entities, so that information from one modality can be used to enrich or filter records in another. Both challenges compound the difficulties already present in the relational setting: incomplete metadata, value heterogeneity, and the open-world assumption because there is now no common data model against which to measure overlap or relevance.

These challenges are further sharpened in federated settings where raw data cannot be shared. For tabular data, discovery systems can often rely on schemas, attribute statistics, and histograms as approved summaries. For images or videos,

there is no obvious analog. Discovery typically relies on learned representations such as embeddings, which summarize semantic content in a compact form. However, these representations themselves may leak sensitive information, especially when shared across organizational boundaries. The core difficulty in distributed multimodal environments is therefore not simply heterogeneity of data types, but heterogeneity of what can be safely revealed. A provider may be willing to share textual descriptions of a video dataset but not visual embeddings. Another may allow spatiotemporal aggregates but prohibit fine-grained temporal resolution. As a result, multimodal discovery systems must reason over uneven and modality-specific abstractions, often combining incomparable signals within a single search process.

This raises a deeper question about the nature of discovery: Should systems attempt to unify all modalities into a shared representation space, or should they embrace modality-specific discovery operators with limited interoperability? In centralized settings, the former is often feasible. In federated settings, the latter is frequently the only practical option.

Wang et al. (2025) directly confront the centralized case by envisioning an end-to-end declarative system for querying and analyzing multimodal data lakes. Their central contribution is a set of *multimodal operators* that extend the classical relational algebra with primitives capable of expressing analytical workloads over heterogeneous sources. The core operators: Selection, Projection, Join, Aggregation, LookUp, Ranking, and Conjunction are defined at a level of abstraction that is independent of the underlying modality. A Join predicate, for instance, can specify either an exact relational condition or a natural-language similarity condition, with the physical implementation determined by the modality of the sources involved. The LookUp operator generalizes keyword search to encompass similarity search and prompt-based inference over language models, providing a single interface that subsumes retrieval across structured, semi-structured, and unstructured sources.

Discovery in this framework is not a stand-alone preprocessing step but an integral part of query execution. When a user submits a query without specifying which modality or data source the relevant data resides in, the system triggers a *discovery operator* implicitly as part of query planning. The discovery module maps the query's predicates to candidate data sources across modalities by computing relevance scores against intermediate representations of each source, returning those that exceed a threshold. This tight coupling of discovery and query planning is a significant departure from the pipeline model assumed in most prior work, where discovery and integration are treated as sequential and independent stages. A second line of work frames multimodal discovery as a planning and orchestration problem, where a language agent decomposes a natural-language query into modality-specific subtasks and coordinates their execution. Nooralahzadeh et al. (2025) propose M^2EX, a system that enables multimodal data exploration via language agents grounded in a real-world enterprise use case spanning databases, text documents, images, and videos.

Across these systems, several open challenges recur. The most fundamental is the absence of a unified data model: representing tables, graphs, documents, and

images within a single framework while preserving semantic relationships between them, particularly entity correspondences that cross-modality boundaries remains unsolved (Wang et al. 2025). Cross-modal data discovery itself requires training representation learning models that map records from different modalities into a shared embedding space, which demands high-quality cross-modal annotations that are expensive to obtain (Xiong et al. 2025). Taken together, these observations suggest that multimodal discovery is best understood not as a simple extension of the relational operations covered in Sects. 3.2–3.4, but as a qualitatively different problem that requires new operator semantics, new relevance models, and new planning frameworks. The relational operations remain relevant as special cases, and they handle the structured component of a multimodal query, but they must be composed with non-relational primitives in a principled way. In centralized settings, unifying all modalities into a shared representation space is a tractable goal (Wang et al. 2025). In federated settings, where visibility constraints impose modality-specific limits on what each provider can safely disclose, modality-specific operators with limited interoperability may be the only practical design choice. How to define, optimize, and evaluate such compositions across both settings is an active research direction.

3.9 Conclusion

This chapter gave a comprehensive overview of existing discovery operations. Keyword search, union search, and join search remain the core primitives because they capture the dominant ways analysts interact with external data: locating relevant tables, expanding coverage vertically, and enriching context horizontally. The practical value of these operations depends on how downstream usefulness is assessed and how uncertainty, provenance, and integration effects are communicated to the user.

A central takeaway is that real discovery workflows are iterative and compositional: Users move from keyword search to union, from union to imputation, and from join discovery to task-aware ranking and often reformulate their requests along the way. This shifts the design problem from building isolated retrieval mechanisms to building systems that support operation sequences with clear semantics, efficient execution, and meaningful intermediate representations. Query languages and plan-based discovery systems provide the framework for expressing sequences and enable optimizing them.

The chapter also highlights that discovery is becoming increasingly semantic. Natural-language interfaces extend classical keyword retrieval by grounding user intent, planning multistep discovery pipelines, and, in some cases, generating direct answers. Similarly, learned representations and hybrid retrieval methods broaden the notion of compatibility beyond exact overlap toward semantic relatedness, approximate joinability, and contextual usefulness. Yet these advances do not eliminate the need for structured operators. Rather, they reinforce the need for discovery systems

that integrate lexical, structural, and semantic signals in a controlled and explainable manner.

Finally, the chapter makes clear that discovery operations cannot be studied in isolation from their deployment environment. In distributed, federated, privacy-aware, and increasingly multimodal repositories, the semantics of discovery change because systems must often reason over summaries, abstractions, or protected views rather than raw data. As a result, future discovery systems must balance expressiveness, efficiency, privacy, and robustness while remaining transparent about approximation and access constraints. Taken together, these observations suggest a clear research direction: Holistic data discovery systems should combine rich operator support, principled composition, semantic interfaces, and environment-aware execution. The next chapter builds on this view by turning from discovery operations to the indexing and systems mechanisms required to support them at scale.

References

Agrawal S, Chaudhuri S, Das G (2002) Dbxplorer: a system for keyword-based search over relational databases. In: Proceedings of the International Conference on Data Engineering (ICDE), pp 5–16, DOI https://doi.org/10.1109/ICDE.2002.994693

Balaka MIL, Fernandez RC (2026) The pneuma project: Reifying information needs as relational schemas to automate discovery, guide preparation, and align data with intent. In: Proceedings of the Conference on Innovative Data Systems Research (CIDR), www.cidrdb.org, URL https://www.cidrdb.org/papers/2026/p31-balaka.pdf

Balaka MIL, Alexander D, Wang Q, Gong Y, Krisnadhi A, Fernandez RC (2025) Pneuma: Leveraging llms for tabular data representation and retrieval in an end-to-end system. Proceedings of the ACM on Managanagement of Data (PACMMOD) 3(3):200:1–200:28, URL https://doi.org/10.1145/3725337

Behme L, Galhotra S, Beedkar K, Markl V (2024) Fainder: A fast and accurate index for distribution-aware dataset search. Proc VLDB Endow 17(11):3269–3282, DOI https://doi.org/10.14778/3681954.3681999, URL https://www.vldb.org/pvldb/vol17/p3269-behme.pdf

Bellahsene Z, Bonifati A, Rahm E (eds) (2011) Schema Matching and Mapping. Data-Centric Systems and Applications, Springer, URL https://doi.org/10.1007/978-3-642-16518-4

Bellin G, De Paiva V, Ritter E, et al. (2001) Extended curry-howard correspondence for a basic constructive modal logic. In: Proceedings of methods for modalities, vol 2

Bhalotia G, Hulgeri A, Nakhe C, Chakrabarti S, Sudarshan S (2002) Keyword searching and browsing in databases using banks. In: Proceedings of the International Conference on Data Engineering (ICDE), pp 431–440, DOI https://doi.org/10.1109/ICDE.2002.994756

Bogatu A, Fernandes AAA, Paton NW, Konstantinou N (2020) Dataset discovery in data lakes. In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE, pp 709–720, URL https://doi.org/10.1109/ICDE48307.2020.00067

Brickley D, Burgess M, Noy NF (2019) Google dataset search: Building a search engine for datasets in an open web ecosystem. In: Liu L, White RW, Mantrach A, Silvestri F, McAuley JJ, Baeza-Yates R, Zia L (eds) Proceedings of the International World Wide Web Conference (WWW), ACM, pp 1365–1375, URL https://doi.org/10.1145/3308558.3313685

Brin S, Page L (1998) The anatomy of a large-scale hypertextual web search engine. In: Proceedings of the International World Wide Web Conference (WWW), Elsevier Science Publishers B. V., NLD, WWW7, p 107–117

Cappuzzo R, Coelho A, Lefebvre F, Papotti P, Varoquaux G (2025) Retrieve, merge, predict: Augmenting tables with data lakes. Trans Mach Learn Res 2025, URL https://openreview.net/forum?id=4uPJN6yfY1

Charikar MS (2002) Similarity estimation techniques from rounding algorithms. In: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, pp 380–388

Chaudhuri S, Das G (2009) Keyword querying and ranking in databases. Proceedings of the VLDB Endowment (PVLDB) 2(2):1658–1659, URL https://doi.org/10.14778/1687553.1687622

Chen T, Kornblith S, Norouzi M, Hinton G (2020) A simple framework for contrastive learning of visual representations. In: Proceedings of the International Conference on Machine Learning (ICML), JMLR.org, ICML'20

Chepurko N, Marcus R, Zgraggen E, Fernandez RC, Kraska T, Karger DR (2020) ARDA: automatic relational data augmentation for machine learning. Proceedings of the VLDB Endowment (PVLDB) 13(9):1373–1387, DOI https://doi.org/10.14778/3397230.3397235, URL http://www.vldb.org/pvldb/vol13/p1373-chepurko.pdf

Dong Y, Xiao C, Nozawa T, Enomoto M, Oyamada M (2023) Deepjoin: Joinable table discovery with pre-trained language models. Proceedings of the VLDB Endowment (PVLDB) 16(10):2458–2470, DOI https://doi.org/10.14778/3603581.3603587, URL https://www.vldb.org/pvldb/vol16/p2458-dong.pdf

Esmailoghli M, Redyuk S, Martinez R, Abedjan Z, Rabl T, Markl V (2019) Explanation of air pollution using external data sources. In: BTW

Esmailoghli M, Quiané-Ruiz J, Abedjan Z (2021) COCOA: correlation coefficient-aware data augmentation. In: Velegrakis Y, Zeinalipour-Yazti D, Chrysanthis PK, Guerra F (eds) Proceedings of the International Conference on Extending Database Technology (EDBT), OpenProceedings.org, pp 331–336, URL https://doi.org/10.5441/002/edbt.2021.30

Esmailoghli M, Quiané-Ruiz J, Abedjan Z (2022) MATE: multi-attribute table extraction. Proceedings of the VLDB Endowment (PVLDB) 15(8):1684–1696, URL https://www.vldb.org/pvldb/vol15/p1684-esmailoghli.pdf

Esmailoghli M, Schnell C, Miller RJ, Abedjan Z (2025) BLEND: A Unified Data Discovery System . In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE Computer Society, Los Alamitos, CA, USA, pp 737–750, DOI https://doi.org/10.1109/ICDE65448.2025.00061, URL https://doi.ieeecomputersociety.org/10.1109/ICDE65448.2025.00061

Esmailpour A, Galhotra S, Raychaudhury R, Sintos S (2025) A theoretical framework for distribution-aware dataset search. Proc ACM Manag Data 3(2):90:1–90:26, URL https://doi.org/10.1145/3725227

Fan G, Wang J, Li Y, Zhang D, Miller RJ (2023) Semantics-aware dataset discovery from data lakes with contextualized column-based representation learning. Proceedings of the VLDB Endowment (PVLDB) 16(7):1726–1739, URL https://www.vldb.org/pvldb/vol16/p1726-fan.pdf

Fernandez RC, Abedjan Z, Koko F, Yuan G, Madden S, Stonebraker M (2018) Aurum: A data discovery system. In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE Computer Society, pp 1001–1012, URL https://doi.org/10.1109/ICDE.2018.00094

Hristidis V, Papakonstantinou Y (2002) Discover: keyword search in relational databases. In: Proceedings of the VLDB Endowment (PVLDB), VLDB '02, p 670–681

Hu X, Lei C, Qin X, Katsifodimos A, Faloutsos C, Rangwala H (2025) Polyjoin: Semantic multi-key joinable table search in data lakes. In: Chiruzzo L, Ritter A, Wang L (eds) Findings of the Association for Computational Linguistics: (NAACL), Association for Computational Linguistics, pp 384–395, URL https://doi.org/10.18653/v1/2025.findings-naacl.23

Huang Z, Liu J, Alabi D, Fernandez RC, Wu E (2023) Saibot: A differentially private data search platform. Proc VLDB Endow 16(11):3057–3070, DOI https://doi.org/10.14778/3611479.3611508, URL https://www.vldb.org/pvldb/vol16/p3057-huang.pdf

Idreos S, Papaemmanouil O, Chaudhuri S (2015) Overview of data exploration techniques. In: Sellis TK, Davidson SB, Ives ZG (eds) Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015, ACM, pp 277–281, URL https://doi.org/10.1145/2723372.2731084

Kamara S, Mohassel P, Raykova M, Sadeghian S (2014) Scaling private set intersection to billion-element sets. In: International conference on financial cryptography and data security, Springer, pp 195–215

Kang AY, Galhotra S (2024) TQL: towards type-driven data discovery. In: Ding W, Lu C, Wang F, Di L, Wu K, Huan J, Nambiar R, Li J, Ilievski F, Baeza-Yates R, Hu X (eds) IEEE International Conference on Big Data, BigData 2024, Washington, DC, USA, December 15-18, 2024, IEEE, pp 7338–7343, URL https://doi.org/10.1109/BigData62323.2024.10825227

Khatiwada A, Fan G, Shraga R, Chen Z, Gatterbauer W, Miller RJ, Riedewald M (2023) SANTOS: relationship-based semantic table union search. Proceedings of the ACM on Managanagement of Data (PACMMOD) 1(1):9:1–9:25, URL https://doi.org/10.1145/3588689

Kumar A, Naughton J, Patel JM, Zhu X (2016) To join or not to join?: Thinking twice about joins before feature selection. In: Proceedings of the International Conference on Management of Data (SIGMOD), pp 19–34

Liu F, Yu C, Meng W, Chowdhury A (2006) Effective keyword search in relational databases. In: Proceedings of the International Conference on Management of Data (SIGMOD), Association for Computing Machinery, New York, NY, USA, p 563–574, URL https://doi.org/10.1145/1142473.1142536

Liu Q, Chen B, Guo J, Ziyadi M, Lin Z, Chen W, Lou J (2022) TAPEX: table pre-training via learning a neural SQL executor. In: The International Conference on Learning Representations (ICLR), OpenReview.net, URL https://openreview.net/forum?id=O50443AsCP

Manning CD, Raghavan P, Schütze H (2008) Introduction to Information Retrieval. Cambridge University Press, USA

Melnik S, Garcia-Molina H, Rahm E (2002) Similarity flooding: a versatile graph matching algorithm and its application to schema matching. In: Proceedings of the International Conference on Data Engineering (ICDE), pp 117–128, DOI https://doi.org/10.1109/ICDE.2002.994702

Meyer HJ, Grunert H, Waizenegger T, Woltmann L, Hartmann C, Lehner W, Esmailoghli M, Redyuk S, Martinez R, Abedjan Z, et al. (2019) Particulate matter matters-the data science challenge@ btw 2019. Datenbank-Spektrum pp 1–18

Nargesian F, Zhu E, Pu KQ, Miller RJ (2018) Table union search on open data. Proceedings of the VLDB Endowment (PVLDB) 11(7):813–825, DOI https://doi.org/10.14778/3192965.3192973, URL http://www.vldb.org/pvldb/vol11/p813-nargesian.pdf

Nooralahzadeh F, Zhang Y, Fürst J, Stockinger K (2025) Multi-modal data exploration via language agents. In: Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, pp 795–813

Pal V, Yates A, Kanoulas E, de Rijke M (2023) MultiTabQA: Generating tabular answers for multi-table question answering. In: Rogers A, Boyd-Graber J, Okazaki N (eds) Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL), Association for Computational Linguistics, Toronto, Canada, pp 6322–6334, DOI https://doi.org/10.18653/v1/2023.acl-long.348, URL https://aclanthology.org/2023.acl-long.348/

Pourreza M, Rafiei D (2023) DIN-SQL: decomposed in-context learning of text-to-SQL with self-correction. In: Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS), Curran Associates Inc., Red Hook, NY, USA, NIPS '23

Robertson S, Zaragoza H (2009) The probabilistic relevance framework: BM25 and beyond. Found Trends Inf Retr 3(4):333–389, URL https://doi.org/10.1561/1500000019

Saha D, Floratou A, Sankaranarayanan K, Minhas UF, Mittal AR, Özcan F (2016) Athena: an ontology-driven system for natural language querying over relational data stores. Proceedings of the VLDB Endowment (PVLDB) 9(12):1209–1220, URL https://doi.org/10.14778/2994509.2994536

Santos ASR, Bessa A, Chirigati F, Musco C, Freire J (2021) Correlation sketches for approximate join-correlation queries. In: Li G, Li Z, Idreos S, Srivastava D (eds) Proceedings of the International Conference on Management of Data (SIGMOD), ACM, pp 1531–1544, URL https://doi.org/10.1145/3448016.3458456

Santos ASR, Bessa A, Musco C, Freire J (2022) A sketch-based index for correlated dataset search. In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE, pp 2928–2941, URL https://doi.org/10.1109/ICDE53745.2022.00264

Sarma AD, Fang L, Gupta N, Halevy AY, Lee H, Wu F, Xin R, Yu C (2012) Finding related tables. In: Candan KS, Chen Y, Snodgrass RT, Gravano L, Fuxman A (eds) Proceedings of the International Conference on Management of Data (SIGMOD), ACM, pp 817–828, URL https://doi.org/10.1145/2213836.2213962

Wang J, Feng Y, Shen C, Rahman S, Kandogan E (2025) Towards operationalizing heterogeneous data discovery. URL https://arxiv.org/abs/2504.02059

Wang Z, Zhang H, Li CL, Eisenschlos JM, Perot V, Wang Z, Miculicich L, Fujii Y, Shang J, Lee CY, Pfister T (2024) Chain-of-table: Evolving tables in the reasoning chain for table understanding. In: The International Conference on Learning Representations (ICLR), URL https://openreview.net/forum?id=4L0xnS4GQM

Xiong R, Zhao S, Chen C, Xu Z (2025) Optimizing multimodal data queries in data lakes. Tsinghua Science and Technology 30(6):2625–2637

Yakout M, Ganjam K, Chakrabarti K, Chaudhuri S (2012) Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In: Candan KS, Chen Y, Snodgrass RT, Gravano L, Fuxman A (eds) Proceedings of the International Conference on Management of Data (SIGMOD), ACM, pp 97–108, URL https://doi.org/10.1145/2213836.2213848

Yu X, Jian P, Chen C (2025) TableRAG: A retrieval augmented generation framework for heterogeneous document reasoning. In: Christodoulopoulos C, Chakraborty T, Rose C, Peng V (eds) Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, Suzhou, China, pp 14063–14082, DOI https://doi.org/10.18653/v1/2025.emnlp-main.710, URL https://aclanthology.org/2025.emnlp-main.710/

Zhang H, Ma Y, Yang H (2025) ALTER: Augmentation for large-table-based reasoning. In: Chiruzzo L, Ritter A, Wang L (eds) Proceedings of the Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics (NAACL), Association for Computational Linguistics, Albuquerque, New Mexico, pp 179–198, DOI https://doi.org/10.18653/v1/2025.naacl-long.9, URL https://aclanthology.org/2025.naacl-long.9/

Zhang S, Balog K (2018) Ad hoc table retrieval using semantic similarity. In: Proceedings of the 2018 world wide web conference, pp 1553–1562

Zhu E, Nargesian F, Pu KQ, Miller RJ (2016) LSH ensemble: Internet-scale domain search. Proceedings of the VLDB Endowment (PVLDB) 9(12):1185–1196, DOI https://doi.org/10.14778/2994509.2994534, URL http://www.vldb.org/pvldb/vol9/p1185-zhu.pdf

Zhu E, Deng D, Nargesian F, Miller RJ (2019) JOSIE: overlap set similarity search for finding joinable tables in data lakes. In: Boncz PA, Manegold S, Ailamaki A, Deshpande A, Kraska T (eds) Proceedings of the International Conference on Management of Data (SIGMOD), ACM, pp 847–864, URL https://doi.org/10.1145/3299869.3300065

Chapter 4
Indexing Strategies for Data Discovery



Abstract Finding relevant tables in a data lake is only half the problem; doing so at the scale of hundreds of millions of tables that cannot fit in main memory requires efficient index structures tailored to the specific demands of data discovery. This chapter examines the indexes that make discovery practical. We cover five broad families: value-based indexes built around the classical inverted index, column-based indexes that use MinHash locality-sensitive hashing (LSH) and its variants to approximate set similarity without exhaustive pairwise comparison, row-based indexes that represent composite-key membership, table-level indexes that operate over embeddings or metadata summaries, and graph-based indexes that capture relationships between datasets to support exploratory search. We also discuss indexes designed specifically to discover correlating columns, addressing the mismatch between join quality and the downstream task. We close the chapter by examining how pretrained language models and foundation models are reshaping the indexing landscape, replacing or augmenting handcrafted structures with dense vector retrieval and learned scoring functions. The trade-off between exactness, scalability, and semantic awareness runs throughout discovery pipelines, and the right balance depends heavily on the operator, the lake, and what the user actually needs.

Keywords Indexing data lakes · Inverted index · Locality-sensitive hashing · MinHash · Sketches · Embedding-based indexing

4.1 Introduction

The previous chapter discussed data discovery operators and their individual goals in answering the question of *WHAT* to find and *WHAT* constitutes relevant tables or columns. Efficient execution of these operators on large data lakes of hundreds of millions of tables requires answering the question of *HOW* to perform these operations. As modern lakes cannot reside in main memory due to their size, efficient data structures and filtering techniques become relevant. As a result, a large

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027

Z. Abedjan et al., *Data Discovery in Data Lakes*,
https://doi.org/10.1007/978-3-032-30822-1_4

Table 4.1 Mapping between indexes and data discovery studies

Index		Papers
Value (cell)	Value → Location	Xiao et al. (2011), Esmailoghli et al. (2025), Zhu et al. (2019), Xiao et al. (2009), Bharadwaj et al. (2021), Sarma et al. (2012), Abedjan et al. (2016), Yakout et al. (2012)
	Location → Value	Xiao et al. (2011), Zhu et al. (2019), Abedjan et al. (2016)
	Value → Class (Domain)	Sarma et al. (2012)
	Value → Embedding	Bogatu et al. (2020)
Column	LSH Variations	Venetis et al. (2012), Fernandez et al. (2019), Zhu et al. (2016), Amsterdamer and Cohen (2021), Nargesian et al. (2018b), Bogatu et al. (2020), Fernandez et al. (2018), Christensen et al. (2025)
	Column → Metadata	Abedjan et al. (2016), Xiao et al. (2011), Zhu et al. (2019), Fernandez et al. (2019), Sarma et al. (2012)
	Column Names → Tables	Yakout et al. (2012), Amsterdamer and Cohen (2021)
	Column → Domain	Zhang and Ives (2020)
	Column → Rank	Santos et al. (2021), Santos et al. (2022), Esmailoghli et al. (2025)
	Column → Embedding	Suri et al. (2021a), Bogatu et al. (2020), Fan et al. (2023)
Row	Row → Table	Yakout et al. (2012), Abedjan et al. (2016), Esmailoghli et al. (2025)
Table	Table → Embedding	Balaka et al. (2025), Liu et al. (2024), Santana and Ribeiro (2023), An et al. (2025), Eltabakh et al. (2024)
	Table → Metadata	Abedjan et al. (2016)
Graph	V: Columns, E: Similarities	Fernandez et al. (2018), Zhang and Ives (2020), Zhang and Ives (2019), Yakout et al. (2012), Bharadwaj et al. (2021), Nargesian et al. (2020), Ouellette et al. (2021), Nargesian et al. (2018a), Nargesian et al. (2023), Singh et al. (2016)
No Index		Hu et al. (2025), Walker et al. (2023), Majumder et al. (2024), Kayali et al. (2024), Hu et al. (2023), Dong et al. (2023); Suri et al. (2021b)

number of indexes have been developed to address this issue. In this chapter, we will take a closer look at these indexes and how they speed up data discovery operators.

Table 4.1 summarizes indexes for data discovery. It categorizes indexes into six groups: five index-based families grouped by the granularity at which they are queried, value-based indexes, column-based indexes, row-based indexes, table-based indexes, and graph-based indexes, and a sixth "No Index" category covering recent approaches that bypass conventional index structures and rely directly on large language models (LLMs) for discovery. Each of these index types serves a

distinct purpose. Value-, column-, row-, and table-based indexes enable querying data lakes at different levels of granularity, from individual cell values to entire tables. Graph-based indexes facilitate exploratory search by capturing relationships between datasets, allowing users to navigate table connections.

In this chapter, we discuss the use cases and construction techniques for each indexing category. We begin with value-based and column-based indexes, which form the backbone of most discovery systems, followed by a dedicated section on correlation-oriented indexes. We then briefly cover row-based, table-based, and graph-based indexes, which address discovery at coarser granularity and through relational structures. We then survey embedding-based and foundation model approaches that move beyond syntactic content to capture semantic meaning. Finally, we discuss the implications for indexing in distributed lakes.

4.2 Value-Based Indexes

Value-based indexes focus on cell values so that the granularity of mappings refers to individual cell values and their locations. They allow querying the data lake by a value or its location identifier, resembling conventional inverted and forward indexes, respectively.

The *Value* $\rightarrow$ *Location(s)* index, conventionally known as the *inverted index*, maps a cell value to a list of locations in the data lake, where the value occurs. A location can be a unique cell identifier, such as a combination of table, column, and row identifiers (Abedjan et al. 2016; Esmailoghli et al. 2025), a unique column identifier (Xiao et al. 2011; Zhu et al. 2019; Xiao et al. 2009; Bharadwaj et al. 2021; Sarma et al. 2012; Yakout et al. 2012), or a more coarse-grained location, where simply the table identifier is stored. The list of hits for a value in the data lake is accordingly its corresponding posting list. For instance, *Paris* $\rightarrow$ $[T_1, T_{12}, \dots]$, where "Paris" is a cell value and $[T_1, T_{12}, \dots]$ is the posting, indicates that the value "Paris" exists in tables T_1, T_{12}, ... The *inverted index* is the most fundamental structure for data discovery. It is extensively used in keyword search, JOIN search, as well as more complicated operators, such as correlation search.

The *Location* $\rightarrow$ *Value(s)* index, also known as forward index, maps a location to one or multiple values. If the location represents a column, the index maps the column identifier to a collection of its cell values, and if the location represents a cell, including a table, column, and row identifier, the query result would be a single cell value.

Value $\rightarrow$ *Class* and *Value* $\rightarrow$ *embedding* indexes are used in semantic similarity search. A class can be types obtained from a knowledge graph (Sarma et al. 2012), e.g., "Paris $\rightarrow$ City", or an embedding, which is a high-dimensional vector representing the value (Bogatu et al. 2020).

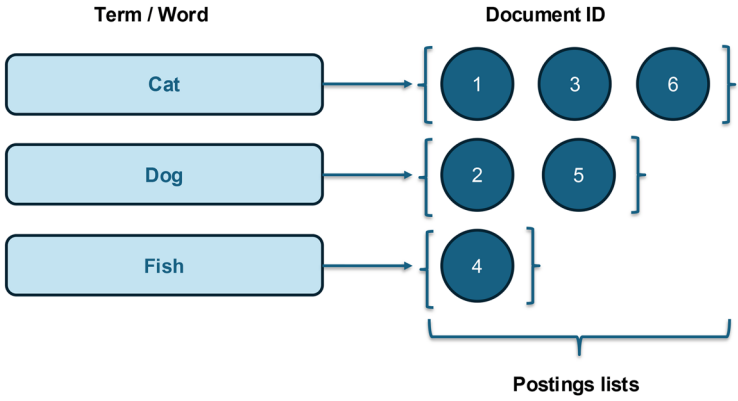


Fig. 4.1 The structure of an inverted index. Each term, word, or token is mapped to a list of entity identifiers, e.g., a document containing those words or columns containing cell values. This list of identifiers is known as the posting list

4.2.1 Inverted Index

Inverted indexes have been the fundamental index structure in information retrieval, where terms are mapped to their containing documents, i.e., websites; Manning et al. (2008). Figure 4.1 demonstrates an example of a traditional inverted index mapping words to the documents containing those words. From a data discovery perspective, an inverted index can serve a similar purpose by enabling the retrieval of tables via keywords and facilitating the calculation of joinability scores for exact JOINs. Tables or columns replace the role of the documents in Fig. 4.1 and cell values replace words and terms.

An early example of an inverted index in data lakes was used for transformation discovery (Abedjan et al. 2016, 2015). Given an input column or a set of input columns with values in a specific representation and some example values in a target representation, the discovery system, DataXFormer, searches tables that contain rows aligning the input and output examples. Those tables are used to harvest potential outputs for the remaining values in the input columns. The index in this system maps values to row, column, and table identifiers and vice versa. DataXFormer uses this index to align candidate values according to the table rows in which they occur. This is essential for transformation discovery, where the system must identify how input values in one column relate to output values in another, requiring row-level co-occurrence information that column-level indexes alone cannot provide.

Such an inverted index can serve both JOIN and UNION discovery by facilitating a score based on value overlap (see also Sect. 3.3–3.4). Figure 4.2 illustrates a simple approach to use an inverted index in finding candidate tables for a JOIN discovery operator. The figure shows an inverted index, mapping values to their corresponding table identifiers, where the posting lists are sorted based on a global order. This order

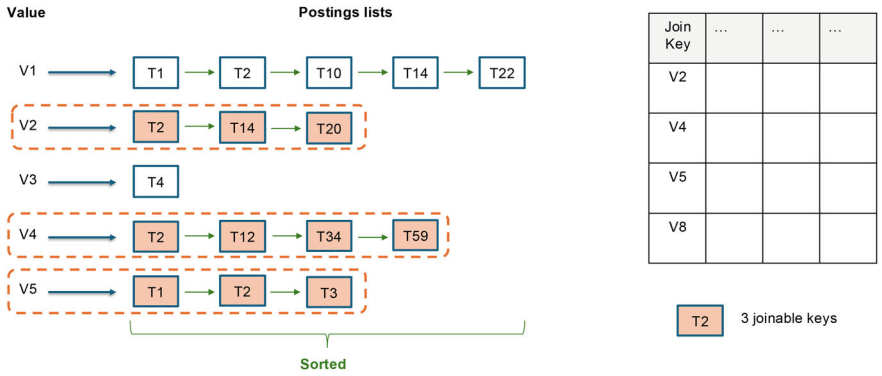


Fig. 4.2 The illustration of utilizing an inverted index in discovering joinable columns to a given query column

enables further optimization of the discovery process. On the right hand side of the figure, there is the user-provided table with a chosen JOIN column. The goal is to find tables from the data lake that overlap the most with the JOIN column.

To this end, an exact JOIN discovery algorithm fetches for each value in the JOIN column the corresponding posting lists and counts the occurrence of each table identifier across all posting lists. In our example, table T2 contains three out of four JOIN keys given in the input table.

Note that in our example, we indexed the table identifiers, resulting in tables as the query results. However, there is no guarantee that all the JOIN keys from the input table appear in the same column of T2. Therefore, a more practical approach is to also consider columns as sets and map cell values to column identifiers. A common column identifier can be a unique id for columns across the data lake or a compound ID, where table identifier and column identifier, e.g., index in the table, are concatenated.

The simple example illustrated in Fig. 4.2 requires one to evaluate all candidate tables sequentially. However, in modern data lakes with large numbers of tables, this approach is computationally prohibitive (Galhotra et al. 2023). Therefore, to enable applications that rely on exact data discovery, a long line of approaches aims at optimizing inverted index-based discovery. Some of such approaches are SSJoin (Chaudhuri et al. 2006), PPJoin (Xiao et al. 2011), MPJoin (Ribeiro and Härder 2011), GroupJoin (Bouros et al. 2012), AdaptJoin (Wang et al. 2012), Top-k Join (Xiao et al. 2009), and Josie (Zhu et al. 2019). These optimizations are divided into two categories of approaches: threshold-based (Fernandez et al. 2018; Wang et al. 2019, 2017; Bayardo et al. 2007; Wang et al. 2012; Mann and Augsten 2014; Mann et al. 2016; Deng et al. 2015) and top-k-based filtering (Zhu et al. 2019; Xiao et al. 2009). These two categories differ in what constitutes joinable. The former assumes a table is joinable if it contains a column that has an overlap with a given query column greater than a predefined threshold, and the latter category discovers joinable tables and only reports the top-k ones based on their overlap with the join

key. This difference in the definition of a joinable table results in different ways of pruning non-joinable candidates.

4.2.1.1 Threshold-Based Filtering

Threshold-based optimization prunes all candidates that will certainly yield smaller overlaps than a predefined threshold to prevent unwanted calculations. A common approach is prefix filtering.

The key idea is to sort values within columns by their global rarity, placing the rarest values first. Intuitively, for a column C, achieving an overlap ratio $\geq t$ requires at least $\lceil t \cdot |C| \rceil$ shared values. So, if the last $\lceil t \cdot |C| \rceil - 1$ values in the sorted order, i.e., the *suffix*, all overlap with a given column, then the *prefix* must have at least one shared value with the query column. The prefix is all values except the suffix: $p = |C| - \lceil t \cdot |C| \rceil + 1$ values. Any candidate pair whose prefixes share no value can be discarded without a full pairwise comparison because they certainly are not going to pass the threshold.

Example (Prefix Filtering) Consider three columns (C_1, C_2, C_3), where their values are sorted by global value order (rarest first) and the overlap threshold $t = 0.5$:

$$C_1 = [v_1,\ v_2,\ v_3,\ v_4,\ v_5] \qquad |C_1| = 5$$
$$C_2 = [v_2,\ v_3,\ v_6,\ v_7] \qquad |C_2| = 4$$
$$C_3 = [v_6,\ v_7,\ v_8] \qquad |C_3| = 3.$$

The prefix length for each column is

- C_1: $p_1 = 5 - \lceil 0.5 \times 5 \rceil + 1 = 3 \Rightarrow \text{prefix} = \{v_1, v_2, v_3\}$
- C_2: $p_2 = 4 - \lceil 0.5 \times 4 \rceil + 1 = 3 \Rightarrow \text{prefix} = \{v_2, v_3, v_6\}$
- C_3: $p_3 = 3 - \lceil 0.5 \times 3 \rceil + 1 = 2 \Rightarrow \text{prefix} = \{v_6, v_7\}$.

The prefix index maps each prefix value to the columns whose prefix contains it. When querying with C_1, we look up its prefix values $\{v_1, v_2, v_3\}$. Since C_2's prefix $\{v_2, v_3, v_6\}$ shares the values v_2 and v_3 with C_1's prefix, the pair (C_1, C_2) becomes a candidate and must be fully verified. In contrast, C_3's prefix $\{v_6, v_7\}$ has no overlap with C_1's prefix, so the pair (C_1, C_3) is *pruned entirely* without performing a full pairwise comparison. The comparison is unnecessary because the length of the prefix and the lack of overlap suggest that the threshold of $t = 0.5$ cannot be met. In fact, the exact Jaccard $J(C_1, C_3) = 0$ confirms that the pruning is correct.

Prefix filtering is not very selective, meaning that many nonoverlapping columns can pass the filter. Bayardo et al. (2007) propose two new filtering strategies, namely positional and suffix filtering. *Positional filtering* tightens the bound mid-scan: As the algorithm reads through a candidate column, it uses the current position in the sorted list to recompute a tighter upper bound on the remaining possible overlap,

pruning the pair as soon as that bound drops below t. *Suffix filtering* checks whether the unread tail of the candidate column is long enough to contribute to the remaining overlap still needed; if not, the pair is discarded immediately without reading further.

To further improve the efficiency of the filtering, parallelization (Vernica et al. 2010; Deng et al. 2014; Sarma et al. 2014), dynamic indexing (Wang et al. 2019), and dependency-based pruning techniques are proposed (Wang et al. 2017).

4.2.1.2 Top-*k* Filtering

Besides the threshold-based approaches, top-k algorithms have been introduced that aim to find the top-k most joinable pairs rather than any pair scoring higher than a threshold (Zhu et al. 2019; Xiao et al. 2009). Zhu et al. (2019) propose Josie, which is an adaptation of the previous top-k JOIN discovery approach (Xiao et al. 2009). Josie is the most efficient approach that leverages inverted indexes for finding exact top-k JOINs. In particular, Josie leverages two index structures, an inverted index and a forward index. These structures are illustrated in Fig. 4.3. The figure represents columns, considered as sets, e.g., S_1, and column values in lowercase letter v, e.g., v_{200}. Again, columns and column values follow a global order. The forward index, depicted on the top part of the figure, maps columns to their containing values. Josie uses an augmented inverted index (see the bottom of Fig. 4.3), which maps each value to its corresponding column identifier as well as the location of the value in the column. Although columns are considered as sets, their values follow a global

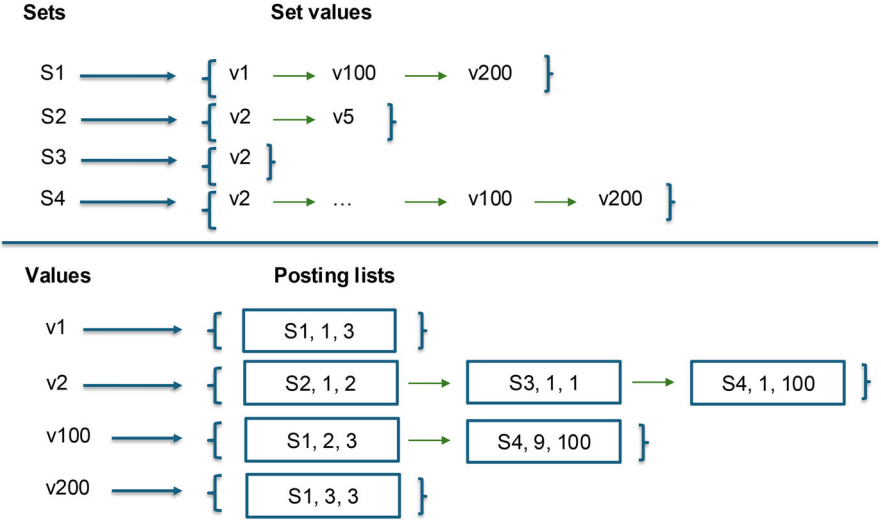


Fig. 4.3 The two index structures used in Josie (Zhu et al. 2019). Top: forward index. Bottom: inverted index

order; therefore, their position in a column is fixed. Furthermore, the index stores the size of the column, i.e., the number of values in the column.

Josie uses these index structures to optimize the JOIN discovery algorithm explained in Fig. 4.2. It does not necessarily read all posting lists per value in the given query column; rather, it leverages a sophisticated optimization approach, where it decides whether it is beneficial to read the posting list or skip it and read the next column values from the forward index. Josie leverages a combination of prefix and positional filtering strategies to address the trade-off between reading a column and reading a posting list of a value.

Another version of the inverted index that is used for keyword search rather than JOIN discovery is the inverted index with skip pointers. In this structure, similar to the previous indexes, the posting list is sorted based on a global order of values. In addition to the normal pointers from the current value to the next, allowing one to sequentially traverse the posting lists, periodic pointers allow *skipping* some location identifiers while reading the posting lists. Figure 4.4 shows this structure.

To illustrate the benefit of the skip pointers in the inverted index, consider a keyword search query where the user queries for tables containing V_1 and V_4. A keyword search algorithm can traverse both of the posting lists at the same time by reading table identifiers in order. Once the algorithm reaches T_{16} for value V_1 and T_8 for value V_4, the next read would be reading T_{41} for V_4. This read provides the algorithm with the information that there is no table identifier between T_8 and T_{41} for V_4. Since the query clearly indicates that the user expects tables that contain both values, the algorithm can safely use the skip pointer in the posting list corresponding to V_1 and jump to T_{28}, skipping two elements in the index. This is illustrated in Fig. 4.5.

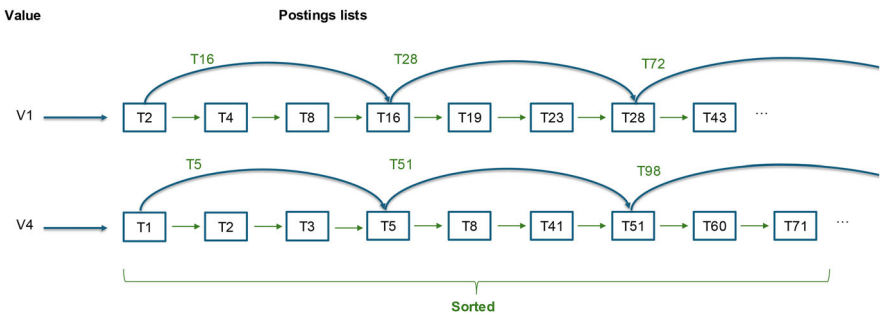


Fig. 4.4 The illustration of an inverted index with skip pointers

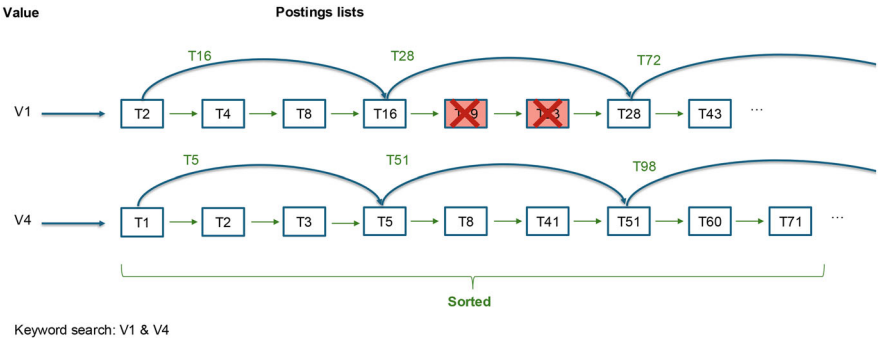


Fig. 4.5 The illustration of the inverted index with skip pointers with a keyword query and the skipped posting list elements

4.2.2 Inverted Index for Multicolumn Retrieval

The inverted index has also been employed for more complex discovery approaches. In particular, the inverted index has been shown to be useful for multicolumn JOIN discovery algorithms (Esmailoghli et al. 2022). Multicolumn JOIN search, also known as composite key JOIN search, is the JOIN discovery search problem where instead of a single JOIN query key, the user provides the JOIN key that contains more than a single column. This is a common use case, where the data at hand contains a composite key, such as a combination of location and datetime (Meyer et al. 2019). Using a single-value inverted index, one has to first retrieve the JOIN candidates for each column individually and then identify the intersection of both result sets and then filter for the tables based on either thresholds or top-k approaches. This approach can lead to very large intermediate result sets and impose additional overhead for intersection and post-processing. For instance, in the case of the composite key ⟨`datetime,location`⟩, the approach might obtain two large sets of intermediate results with tables that join on location but are temporally irrelevant and those that join on datetime but refer to a different locality.

In theory, one might build a multicolumn inverted index where the search values refer to all possible value combinations observed in tables of a lake. However, this is not feasible for even relatively large data lakes due to its factorial storage complexity.

MATE (Esmailoghli et al. 2022) addresses this problem by using a hashing function that works as a bloom filter Bloom (1970). As depicted in Fig. 4.6, MATE hashes each row in the data lake into a so-called *Super Key* that represents all possible JOIN keys in the table row. Then, in the discovery phase, MATE hashes the given composite key and uses bloom filter-like pruning to remove irrelevant JOIN keys. To this end, MATE fetches candidate tables based on querying one of the JOIN keys in the query columns using a single-value inverted index and then applies the bloom filter to remove false positives. To generate the super keys, MATE

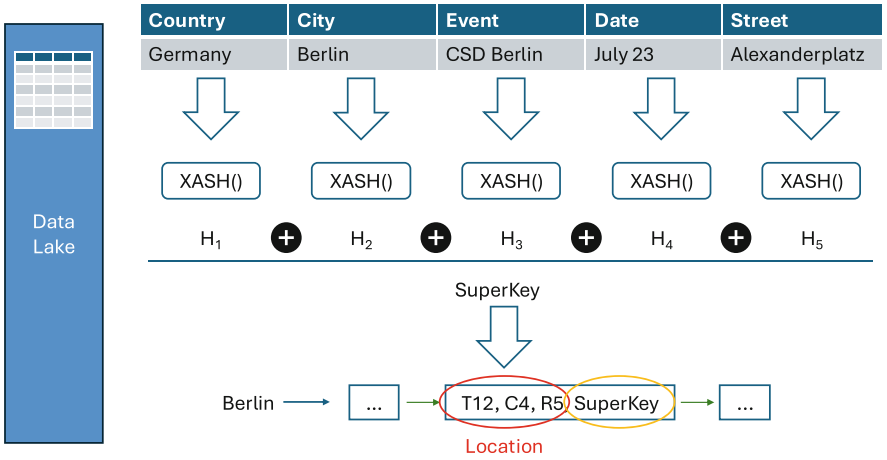


Fig. 4.6 Multicolumn inverted index for composite key JOIN discovery

leverages a tailored hash function, called XASH, designed to capture syntactic features of JOIN keys while maintaining low collision between hashes. As shown, XASH hashes each key value independently and aggregates them using a bitwise OR operation to generate the super key, which is then stored as an additional element in the traditional inverted index, allowing for multicolumn JOIN search using the traditional single-value inverted index.

Example Consider a composite JOIN key consisting of two columns, `City` and `Year`. For simplicity, we use an 8-bit vector for hash space and assume XASH maps each distinct value to one bit position:

Value	XASH bit vector
"Berlin"	`00000100` (bit 2)
"Paris"	`10000000` (bit 7)
"2020"	`00100000` (bit 5)
"2021"	`00000001` (bit 0)

The super key for each row is the bitwise OR of the hashes of its values:

- Query row (Berlin, 2020): `00000100` | `00100000` = `00100100`
- Lake row A (Berlin, 2020): `00100100` (same values, same super key)
- Lake row B (Paris, 2020): `10000000` | `00100000` = `10100000`
- Lake row C (Berlin, 2021): `00000100` | `00000001` = `00000101`

MATE filters candidates by checking whether every bit set in the query super key is also set in the lake row's super key, i.e., $(SK_{query} \,\&\, SK_{lake}) = SK_{query}$:

- Row A: `00100100` & `00100100` = `00100100` ✓ **Candidate**

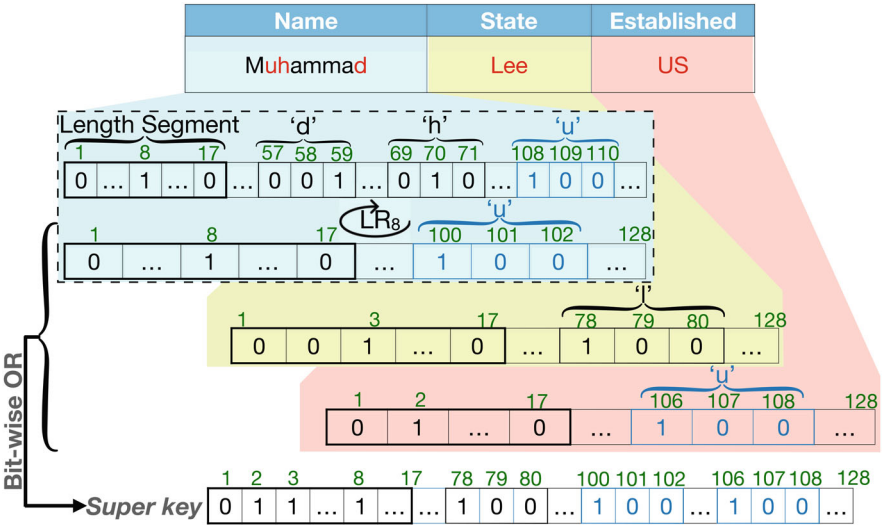


Fig. 4.7 Example of how XASH values are generated and the results are aggregated into a super key (Esmailoghli et al. 2022)

- Row B: `00100100` & `10100000` = `00100000` ≠ `00100100` → **Pruned** (bit 2 for "Berlin" is absent)
- Row C: `00100100` & `00000101` = `00000100` ≠ `00100100` → **Pruned** (bit 5 for "2020" is absent)

With a single bitwise AND per row, MATE eliminates rows B and C without any string comparison, demonstrating how the super key compresses composite-key membership into an efficient bloom-filter-like check.

The example above demonstrates how the pruning proposed by MATE works. Figure 4.7 illustrates how the XASH values are generated. Consider a table row containing three cell values: "muhammad" (C1), "lee" (C2), and "us" (C3). For each value, XASH first identifies the least frequent characters, since rare characters are less likely to appear across different words; they serve as better discriminators. It then encodes each selected character by recording its relative position within the string: the string is divided into β equal regions from left to right, and a single bit is set in the corresponding position within that character's dedicated segment of the hash array. Additionally, a separate length segment records which bit corresponds to the length of the value, allowing the system to quickly discard nonmatching rows without inspecting the character segments at all. As a final step, the entire character portion of the hash is rotated left by the length of the value, so that the same character appearing in two values of different lengths will map to different bit positions, reducing intercolumn collisions.

Once each cell value has its own XASH result, the super key for the row is computed by applying a bitwise OR across all three individual hashes, as shown at

the bottom of Fig. 4.7. This aggregation ensures that the super key encodes evidence of every value present in the row. When querying, the system computes the same XASH over each composite key value combination from the query table and checks whether the resulting bit pattern is fully masked by the candidate row's super key. If even a single bit of the query's hash is not set in the super key, the row is immediately pruned without any further value comparison. As a result, this operation can never produce false negatives.

4.3 Column-Based Indexes

Column-based index structures index tables on the column level by using column representations, such as identifiers, headers, or other summaries. The goal is to efficiently discover relevant and in particular similar columns for a given column. A naive approach of representing columns in a data lake for discovery is in the form of a Boolean matrix, where the X-axis comprises the dimension of columns and the y-axis the dimension of all possible terms in the data lake. A cell (i, j) in the matrix equals 1 if and only if the corresponding column j contains the cell value represented by row i. Figure 4.8 shows such a matrix.

To query this data structure, the query column must be represented as a binary vector over the same dimension as y, i.e., with one entry per possible value in the data lake, so that similarity can be computed directly against any stored column. Figure 4.9 depicts such a query column and how to calculate the Jaccard similarity of

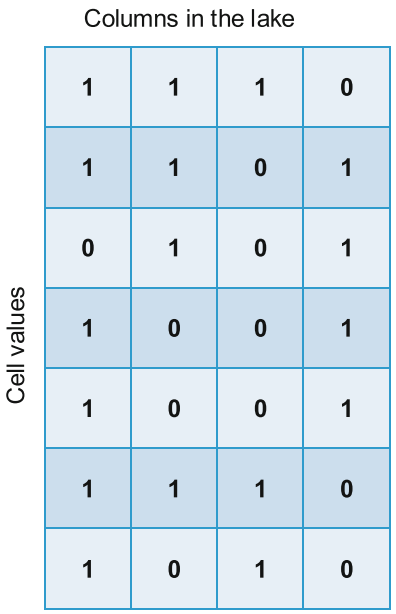


Fig. 4.8 A Boolean matrix representing columns in the data lake (matrix columns) and the values they contain (rows of the matrix)

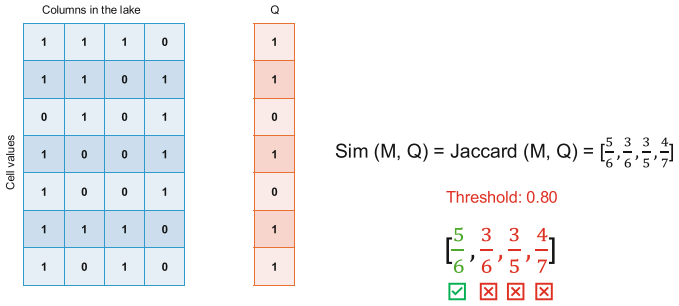


Fig. 4.9 Given a matrix that represents the columns inside a lake and a query column vector, one can calculate the exact Jaccard similarity through simple vector calculus

the column Q with each column represented in the matrix. As the Jaccard similarity is defined as the size of the intersection of two sets divided by their union, we can simply use the representing vectors u and v and calculate $\frac{\sum_i (u_i v_i)}{\sum_i (u_i + v_i)}$, which is the component sum of the dot product $u \cdot v$ divided by the component sum of $u + v$. Once the similarities are calculated, one can find the most similar columns by setting a threshold, in our example, the threshold is 80%.

Calculating similarity between the query and every possible column in the data lake can be very expensive. For instance, the Dresden Web table corpus contains above $700M$ columns in the data lake. Calculating the similarity for all these columns at query time is not practical. Therefore, approximate solutions have been introduced to solve this problem. As shown in Table 4.1, we categorize column-based indexes into the following classes: locality-sensitive hashing (LSH) indexes (Venetis et al. 2012; Fernandez et al. 2019; Zhu et al. 2016; Amsterdamer and Cohen 2021; Nargesian et al. 2018b; Bogatu et al. 2020; Fernandez et al. 2018), column metadata indexes (Xiao et al. 2011; Zhu et al. 2019; Fernandez et al. 2019; Sarma et al. 2012), column to table indexes (Yakout et al. 2012), column to domain indexes (Abedjan et al. 2016; Zhang and Ives 2020), column to value rank indexes (Santos et al. 2021, 2022), and column to embedding indexes (Suri et al. 2021a; Bogatu et al. 2020). In this section, we mainly cover the techniques related to LSH indexes. The indexing techniques that map columns to metadata, table identifiers, and domains are skipped as the approaches for such mappings are either straightforward or derivatives of inverted indexes. We will discuss the subtypes of column to value rank and column to embedding in follow-up sections.

4.3.1 MinHash and Locality-Sensitive Hashing for Column Similarity

The most prominent among approximate solutions in existing data discovery literature is *MinHash locality-sensitive hashing (MinHash LSH)* (Broder 1997a), which provides a probabilistic approach to efficiently identify similar columns without exhaustive pairwise comparisons. Here, we first introduce the concept of MinHash as a technique for estimating set similarity, then explain locality-sensitive hashing as a framework for approximate nearest neighbor search, and finally show how these two concepts are combined to create an efficient index structure for column similarity discovery, e.g., join discovery.

4.3.1.1 MinHash: The Core Idea

MinHash (minimum hashing) is a technique introduced by Broder (1997b) that allows us to estimate the Jaccard similarity between two sets using compact signatures, without explicitly computing the intersection and union.

Before diving into MinHash, let us recall the similarity measure we aim to approximate. Given two sets A and B, the *Jaccard similarity* is defined as

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}. \tag{4.1}$$

The Jaccard similarity ranges from 0 (completely disjoint sets) to 1 (identical sets). In the context of data discovery, sets A and B represent the distinct values contained in two columns. Two columns with high Jaccard similarity share a large proportion of their values, making them good candidates for join operations.

Consider two columns C_1 and C_2 containing city names:

$$C_1 = \{\text{Berlin, Paris, London, Rome, Madrid}\}$$

$$C_2 = \{\text{Berlin, Paris, Vienna, Amsterdam}\}.$$

The intersection contains the values appearing in both columns: $C_1 \cap C_2 = \{\text{Berlin, Paris}\}$, giving us $|C_1 \cap C_2| = 2$. The union contains all distinct values: $|C_1 \cup C_2| = 7$. Therefore, the Jaccard similarity is $J(C_1, C_2) = \frac{2}{7} \approx 0.286$.

The naive approach to computing Jaccard similarity requires materializing both sets and computing their intersection and union, which has time complexity $O(|A| + |B|)$ for each pair comparison.

We now want to represent the columns in a way that the cost of the comparison remains constant and independent of the size of the two columns. MinHash does exactly that. Imagine we assign a random number to every possible cell value that could appear in any column. Now, for any column, we examine the random numbers assigned to its values and select the minimum one. This minimum value is

referred to as the MinHash of that column. The MinHash has a remarkable property: *The probability that two columns have the same MinHash equals their Jaccard similarity*. The reason is that when we pick the minimum random number from the union of two sets, that minimum element is equally likely to be any element in the union. The two sets will have the same MinHash only if this minimum element happens to be in their intersection. Since each element in the union is equally likely to be the minimum, the probability is simply the fraction of elements that are in the intersection, which is exactly the Jaccard similarity.

In practice, we cannot store random numbers for every possible value. Instead, we use hash functions that effectively simulate this randomness. A hash function h maps any value to a number. The MinHash value of a column C under hash function h is

$$\text{MinHash}_h(C) = \min_{x \in C} h(x). \tag{4.2}$$

In other words, we hash every value in the column and keep the smallest hash value.

A single MinHash value gives us a very rough binary estimate, because with a single hash, the values either match or they do not. To get reliable estimates, we use multiple independent hash functions $h_1, h_2, \ldots, h_k$ and create a *signature* for each column:

$$\text{Sig}(C) = \langle \text{MinHash}_{h_1}(C), \text{MinHash}_{h_2}(C), \ldots, \text{MinHash}_{h_k}(C) \rangle. \tag{4.3}$$

The Jaccard similarity can then be estimated by computing the fraction of positions where the signatures match:

$$\hat{J}(A, B) = \frac{\text{number of positions where Sig}(A) = \text{Sig}(B)}{k} \tag{4.4}$$

Let us compute MinHash signatures for three small columns. We will use $k = 4$ hash functions defined as

$$h_1(x) = (x + 1) \mod 5$$
$$h_2(x) = (3x + 1) \mod 5$$
$$h_3(x) = (2x + 4) \mod 5$$
$$h_4(x) = (x + 3) \mod 5.$$

Assume we have encoded our column values as integers, giving us three columns:

$$C_1 = \{0, 3\}$$
$$C_2 = \{0, 1, 3\}$$
$$C_3 = \{2, 4\}.$$

Step 1: Compute Hash Values for All Elements We apply each hash function to each possible element value:

Element x	$h_1(x)$	$h_2(x)$	$h_3(x)$	$h_4(x)$
0	1	1	4	3
1	2	4	1	4
2	3	2	3	0
3	4	0	0	1
4	0	3	2	2

Step 2: Compute MinHash Signatures For each column, we find the minimum hash value across its elements for each hash function:

For $C_1 = \{0, 3\}$:

- h_1: $\min(h_1(0), h_1(3)) = \min(1, 4) = 1$.
- h_2: $\min(h_2(0), h_2(3)) = \min(1, 0) = 0$.
- h_3: $\min(h_3(0), h_3(3)) = \min(4, 0) = 0$.
- h_4: $\min(h_4(0), h_4(3)) = \min(3, 1) = 1$.

So $\text{Sig}(C_1) = \langle 1, 0, 0, 1 \rangle$.

Similarly, we compute:

- $\text{Sig}(C_2) = \langle 1, 0, 0, 1 \rangle$ (for elements 0, 1, 3).
- $\text{Sig}(C_3) = \langle 0, 2, 2, 0 \rangle$ (for elements 2, 4).

Step 3: Estimate Jaccard Similarities We count matching positions:

- $\hat{J}(C_1, C_2)$: Signatures match in all 4 positions $\rightarrow \hat{J} = 4/4 = 1.0$.
- $\hat{J}(C_1, C_3)$: Signatures match in 0 positions $\rightarrow \hat{J} = 0/4 = 0.0$.
- $\hat{J}(C_2, C_3)$: Signatures match in 0 positions $\rightarrow \hat{J} = 0/4 = 0.0$.

The exact Jaccard similarities are $J(C_1, C_2) = 2/3 \approx 0.67$, $J(C_1, C_3) = 0$, and $J(C_2, C_3) = 0$. Notice that the estimate $\hat{J}(C_1, C_2) = 1.0$ overestimates the true value. This is because we used only $k = 4$ hash functions and small sets. With more hash functions, the estimates converge to the true values.

The accuracy of MinHash estimation improves with more hash functions. Specifically, to achieve a standard error of ϵ, we need approximately $k = \frac{1}{4\epsilon^2}$ hash functions. For example, to get estimates within ± 0.05 of the true similarity, we need about $k = 100$ hash functions. Note that for every column in the lake, we need to perform the k hashes once, and we can then reuse the calculated hashes as often as we want for future comparisons.

4.3.1.2 Locality-Sensitive Hashing (LSH)

MinHash gives us compact signatures for estimating Jaccard similarity, but for a given column, we still need to evaluate its hash against all columns in a lake to find

similar columns. In situations where we need to obtain the relationships across all columns of a lake to create a graph, the problem becomes even more problematic because we have to compare all pairs of signatures requiring $O(n^2)$ comparisons for n columns. *Locality-sensitive hashing (LSH)* solves this by building signatures so that similar columns are likely to end up in the same "bucket," while dissimilar columns are unlikely to collide.

The key idea is that we want a hashing scheme where the probability of two items hashing to the same bucket increases with their similarity. For MinHash signatures, this is achieved through the *banding technique*.

4.3.1.3 The Banding Technique

Given a MinHash signature of length k, we divide it into b *bands* of r *rows* each, so $k = b \times r$. Two columns become candidates if they match in *all rows of at least one band*.

Figure 4.10 illustrates this concept. Each signature is split into bands, and each band is hashed to a bucket. If two signatures land in the same bucket for any band, they become a candidate pair.

Let s be the true Jaccard similarity between two columns. Since each MinHash component matches with probability s:

- Probability that all r rows in one band match: s^r
- Probability that a band does NOT match: $1 - s^r$
- Probability that NO band matches (across all b bands): $(1 - s^r)^b$
- Probability of becoming candidates (at least one band matches): $1 - (1 - s^r)^b$

This gives us the S-curve probability function:

$$P(\text{candidate} \mid \text{similarity } s) = 1 - (1 - s^r)^b. \tag{4.5}$$

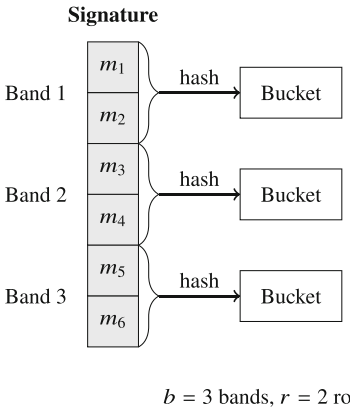


Fig. 4.10 The banding technique: a signature of length $k = 6$ is divided into $b = 3$ bands of $r = 2$ rows each. Each band is hashed separately to a bucket

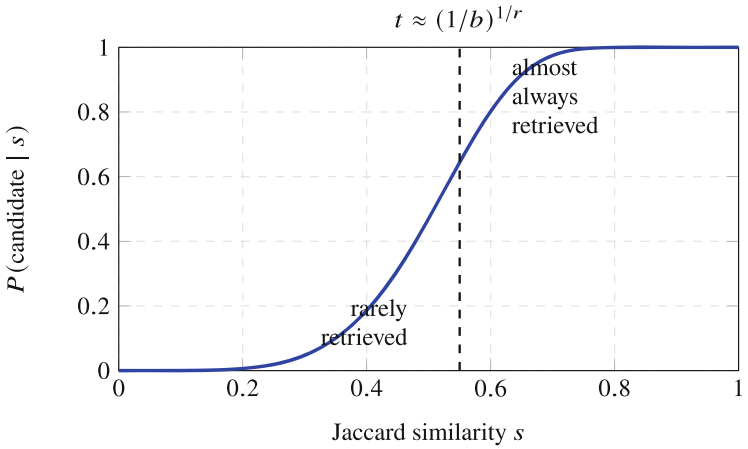


Fig. 4.11 The S-curve of LSH (here $b = 20$, $r = 5$). The probability of two columns becoming candidates rises sharply around the threshold t, acting as a soft filter

Table 4.2 Probability of becoming candidates for different banding parameters ($k = 100$)

b	r	Threshold	$P(s = 0.2)$	$P(s = 0.5)$	$P(s = 0.8)$
20	5	≈0.55	0.006	0.470	0.9996
10	10	≈0.79	<0.001	0.001	0.893
50	2	≈0.14	0.866	0.999	≈1
25	4	≈0.45	0.040	0.762	0.9999

The banding technique creates a sharp threshold effect, illustrated in Fig. 4.11. The curve rises steeply around the threshold $t \approx (1/b)^{1/r}$: pairs with similarity below t are rarely retrieved, i.e., low false-positive rate, while pairs above t are almost always retrieved, i.e., high recall. By choosing b and r, practitioners control where this cutoff falls.

Table 4.2 shows how different parameter choices affect the probability of becoming candidates at various similarity levels, assuming $k = 100$ hash functions.

With $b = 20$ bands and $r = 5$ rows, pairs with 80% similarity are almost always identified as candidates (99.96%), while pairs with only 20% similarity rarely are (0.6%). This is exactly what we want for efficient similarity search.

Figure 4.12 plots these S-curves for the same parameter settings. Each curve shows the sharp threshold effect: pairs above the threshold are almost certain to become candidates, while pairs below it are largely filtered out.

The choice of b and r depends on the application:

- **More rows per band** (higher r): Raises the threshold, requiring higher similarity. Fewer false positives, but might miss some true matches
- **More bands (higher b)**: Lowers the threshold. Better recall, but more false positives to filter

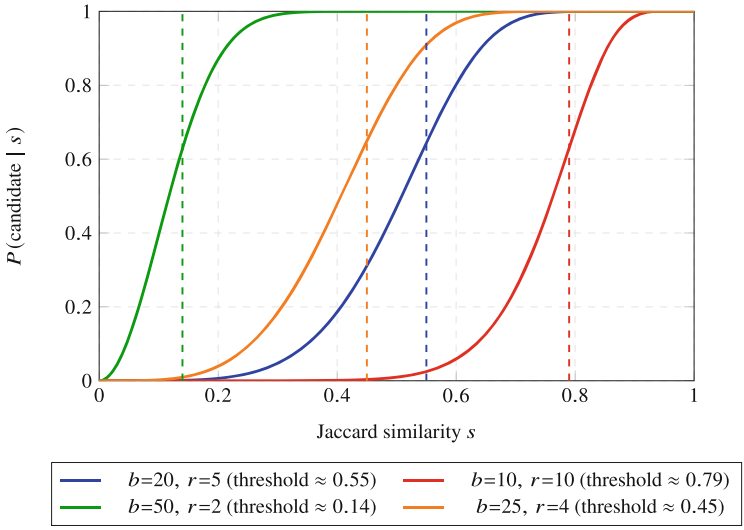


Fig. 4.12 S-curves $P(\text{candidate} \mid s) = 1 - (1 - s^r)^b$ for different banding parameters ($k = 100$). Increasing r shifts the threshold to the right (higher similarity required); increasing b lowers the threshold and steepens the transition

4.3.1.4 Putting It All Together: The MinHash LSH Algorithm

Now we describe the complete MinHash LSH approach for column similarity search. In the index construction phase, i.e., offline phase, for each column C in the data lake:

1. Compute the MinHash signature using k hash functions.
2. Divide the signature into b bands of r rows.
3. For each band, hash it and insert the column ID into the corresponding bucket.

In the query processing phase, i.e., online phase, given a query column Q:

1. Compute the MinHash signature of Q. When column values are strings, first apply a standard hash function, such as MurmurHash, to convert them to integers, then apply MinHash.
2. For each band, hash it and retrieve all column IDs in that bucket.
3. Union all retrieved IDs. These are the candidate similar columns.
4. Verify candidates by computing the exact similarity.

The index has a space complexity of $O(n \cdot k)$ for signatures and $O(n \cdot b)$ for bucket entries. Index construction has a time complexity of $O(n \cdot m)$ where m is the average column size. Query processing has a time complexity of $O(m + b + |C|)$, where $|C|$ is the number of candidates retrieved. It is important to note that the query time complexity is independent of the total number of columns n.

Example Let us walk through an example with five columns, using $k = 6$ hash functions, $b = 3$ bands, and $r = 2$ rows per band.

Assume that the hash for each value is calculated (Step 1 in Sect. 4.3.1.1) and the min hash signatures are obtained (Step 2 in Sect. 4.3.1.1) as follows:

$$\text{Sig}(C_1) = \langle 12, 45, 78, 23, 56, 89 \rangle$$
$$\text{Sig}(C_2) = \langle 12, 45, 34, 67, 56, 89 \rangle$$
$$\text{Sig}(C_3) = \langle 90, 11, 78, 23, 44, 55 \rangle$$
$$\text{Sig}(C_4) = \langle 12, 45, 78, 23, 56, 89 \rangle$$
$$\text{Sig}(C_5) = \langle 33, 22, 11, 99, 88, 77 \rangle.$$

Notice that C_1 and C_4 have identical signatures.

Dividing into Bands:

Column	Band 1 (rows 1–2)	Band 2 (rows 3–4)	Band 3 (rows 5–6)
C_1	$\langle 12, 45 \rangle$	$\langle 78, 23 \rangle$	$\langle 56, 89 \rangle$
C_2	$\langle 12, 45 \rangle$	$\langle 34, 67 \rangle$	$\langle 56, 89 \rangle$
C_3	$\langle 90, 11 \rangle$	$\langle 78, 23 \rangle$	$\langle 44, 55 \rangle$
C_4	$\langle 12, 45 \rangle$	$\langle 78, 23 \rangle$	$\langle 56, 89 \rangle$
C_5	$\langle 33, 22 \rangle$	$\langle 11, 99 \rangle$	$\langle 88, 77 \rangle$

Bucket Assignments:

- Band 1 buckets: $\{C_1, C_2, C_4\}$ share bucket "12-45"; C_3 and C_5 are alone
- Band 2 buckets: $\{C_1, C_3, C_4\}$ share bucket "78-23"; C_2 and C_5 are alone
- Band 3 buckets: $\{C_1, C_2, C_4\}$ share bucket "56-89"; C_3 and C_5 are alone

Candidate pairs (share at least one bucket):

- (C_1, C_2): Yes (Bands 1 and 3)
- (C_1, C_3): Yes (Band 2)
- (C_1, C_4): Yes (all bands)
- (C_2, C_4): Yes (Bands 1 and 3)
- (C_3, C_4): Yes (Band 2)
- C_5: No shared buckets

Column C_5 is correctly identified as dissimilar to all others without any pairwise comparison. This example illustrates the core trade-off of MinHash LSH. Highly similar pairs like (C_1, C_4) collide in every band and are always retrieved. Moderately similar pairs like (C_1, C_2) match in some bands and are still surfaced. Pairs like (C_1, C_3) match in only one band and may be false positives, requiring exact verification. Meanwhile, C_5, dissimilar to all, is filtered out without any pairwise

comparison. The banding parameters b and r control where this trade-off between recall and precision falls.

LSH is commonly used for join discovery, where columns with overlapping hash buckets are considered join candidates (Gionis et al. 1999). The hashing algorithm used to generate the LSH index directly depends on the similarity measure required for the discovery approach. For instance, MinHash LSH (Broder 1997b) is used to calculate Jaccard similarity, while SimHash LSH (Charikar 2002) is employed for calculating Cosine similarity. Note that LSH yields an approximate but fast solution for nearest-neighbor lookups. Hence, when a certain margin of error is allowed, LSH can increase the efficiency of the discovery pipeline.

4.3.2 Beyond Minhash LSH

While MinHash LSH is highly effective, it has limitations that have motivated several extensions:

1. **Fixed threshold:** Standard LSH requires setting the threshold at index time. Aurum and LSH Forests address this issue.
2. **Bias toward smaller sets:** Jaccard similarity is defined as $J(Q, C) = |Q \cap C|/|Q \cup C|$. Because the union grows with the size of C, a smaller candidate column yields a smaller denominator and thus a higher score, even when the overlap is identical. This makes Jaccard biased toward smaller columns. The remedy is *containment similarity*, $\text{Cont}(Q, C) = |Q \cap C|/|Q|$, which fixes the denominator to the query size so that all candidates covering the same query rows receive equal scores. LSH Ensemble (Zhu et al. 2016) builds on standard LSH for fast approximate computation, but estimates containment rather than Jaccard.
3. **Index construction time:** Building signatures for millions of columns is slow. LAZO (Fernandez et al. 2019) uses one permutation hashing to speed this up significantly. Multi-probe LSH is another solution to this problem.

Aurum (Fernandez et al. 2018) tackles the first challenge of fixed thresholds. The objective of Aurum is to efficiently create a graph of joinable columns in the data lake with weighted edges, where weights represent the joinability score between columns, in this case the Jaccard similarity. To do this, Aurum builds multiple LSH indexes, each primed for different thresholds. It uses this collection of indexes to build the join graph by assigning the edge label as the maximum LSH threshold, in which the columns are similar. Similar to the standard LSH approach, Aurum still requires the user to commit to a similarity threshold before queries arrive by building multiple indexes for a discrete set of thresholds.

LSH Forest (Bawa et al. 2005) relaxes this requirement by adapting the effective threshold at query time. It addresses the self-tuning challenge with an index that requires no threshold to be specified at build time. The index consists of a forest of trees, where each tree stores the column signatures as prefixes in a sorted array. Given a query column, LSH Forest first descends each tree top-down to the deepest

prefix that matches the query signature and then expands the search upward until a set of candidate columns have been collected. This top-down descent followed by bottom-up expansion lets the effective similarity threshold adapt to the local data density automatically. As a result, the same index can answer k-nearest-neighbor queries for any k without requiring a rebuild or threshold re-specification.

The second challenge regarding bias toward small columns can be addressed through LSH Ensembles (Zhu et al. 2016). The MinHash LSH approximates the Jaccard Similarity as the similarity score. Zhu et al. (Zhu et al. 2016) show that, although fast, the Jaccard similarity is biased toward smaller tables. If two candidate columns present the same overlap with a given input column, the one with a smaller number of rows will result in higher Jaccard similarity. This is because the Jaccard similarity has the union of the join key values in the denominator, and a smaller denominator leads to higher Jaccard similarity.

Example Consider a query column Q containing 100 distinct city names, and two candidate columns from the data lake:

- C_A: contains 50 values from Q and 50 additional values not in Q (total size: 100)
- C_B: contains 50 values from Q and 5 additional values not in Q (total size: 55)

Both candidates share exactly 50 values with Q, meaning a join with either column would augment the same 50 query rows. Yet Jaccard scores them differently:

$$J(Q,\ C_A) = \frac{50}{|Q \cup C_A|} = \frac{50}{100 + 100 - 50} = \frac{50}{150} \approx 0.33$$

$$J(Q,\ C_B) = \frac{50}{|Q \cup C_B|} = \frac{50}{100 + 55 - 50} = \frac{50}{105} \approx 0.48.$$

C_B scores 45% higher than C_A despite offering the same practical utility.

This bias of Jaccard similarity is undesirable in data discovery scenarios since the final integrated results, e.g., joined tables, are the final product the user demands. To address this issue, the authors in the LSH Ensemble paper propose the set containment similarity as the similarity score. It substitutes the denominator of the Jaccard similarity with the size of the query table: $\text{Cont}(Q, C_A) = \text{Cont}(Q, C_B) = 50/100 = 0.5$, correctly ranking both candidates in our example as equally useful. However, as the MinHash LSH index (Broder 1997b) can only estimate the Jaccard similarity, Zhu et al. (Zhu et al. 2016) propose a new approach that leverages the LSH index to estimate the containment similarity. The LSH Ensemble index leverages a partitioning technique that creates disjoint groups of columns according to their sizes. Then, it separates the LSH indexes generated for each partition. At query time, they also adapt the user-provided threshold to obtain the containment similarity rather than the Jaccard similarity.

Finally, to tackle the problem of time-consuming index creation, LAZO (Fernandez et al. 2019) uses one permutation hashing (OPH) sketches (Shrivastava 2017) to

reduce the number of hashes required before creating the LSH index. They propose a solution to estimate the Jaccard and containment similarity on demand.

Multi-probe LSH (Lv et al. 2007) takes a complementary approach. Standard LSH requires the construction of many hash tables to achieve good recall across a range of thresholds, inflating both memory usage and runtime. Multi-probe LSH instead uses a small number of hash tables but, for each query band, probes not only the exact bucket where the query lands but also a sequence of *nearby* buckets. These are the buckets obtained by modifying one or more band hash values by a small amount. The method constructs a perturbation sequence that enumerates neighboring buckets in decreasing order of their likelihood of containing true matches, so a small number of additional probes recovers most of the recall that would otherwise require far more hash tables. In practice, Multi-probe LSH can reduce the required number of hash tables by an order of magnitude while maintaining comparable recall, making it especially attractive when memory is constrained.

4.4 Indexes for Discovering Correlating Columns

The column-based indexes discussed so far only capture the similarity between two columns based on value overlaps. Therefore, these structures are widely used to discover joinable tables for a given table at hand. As we discussed in Sect. 3.4, the goal of JOIN discovery is usually to obtain additional informative dimensional information or useful features for downstream machine learning tasks. Yet the heuristic used does not in any way assess the usefulness of the obtained attributes for such purposes. Thus, the materialized JOINs must be vetted once the discovery is done resulting in irrelevant tables and additional computational overhead. The objective misalignment is due to the fact that join discovery evaluates the candidate tables merely based on their overlap with the query column and ignores the usefulness of the resulting joined table. This is problematic since not all joinable tables contain informative columns for the user's task at hand. Even with a post-processing step to evaluate the informativeness of such tables, there is no guarantee that these useful tables are among the top joinable tables, i.e., the result set. Ultimately, this is prone to false negatives, as well as false positives. Nevertheless, one has to evaluate each newly joined column and its informativeness for the task at hand.

A series of indexes attempt to solve the objective misalignment via two entirely different strategies. Quadrant Count Ratio (QCR) (Santos et al. 2022) solves the task by creating sketches that capture joinability and feature relevance. COCOA (Esmailoghli et al. 2021; Becktepe et al. 2023) proposes an extension of the inverted index with additional correlation-based pruning. Both approaches assume that the discovery serves a downstream machine learning task, so that the input table represents an incomplete training dataset and contains not only a JOIN column but also a *target column*, which contains the obtained predictions for each row of

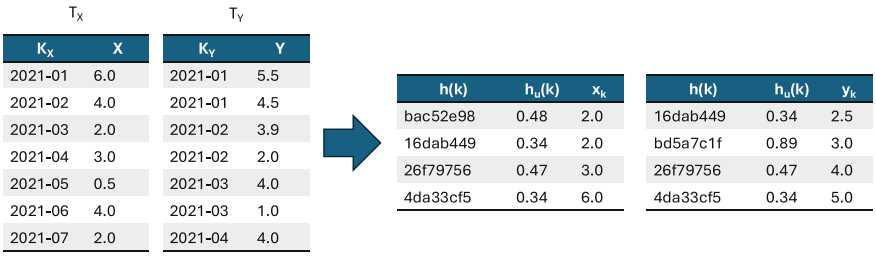


Fig. 4.13 The data sketch proposed by Santos et al. (2021) for correlation discovery

the table. The goal is now to find tables that join on the JOIN column and correlate with the *target column*.

4.4.1 Sketch-Based Correlation Indexes

Santos et al. (2021) introduce a new index structure that maps pairs of categorical and numerical columns to smaller sketches. The goal is to calculate correlation more efficiently on smaller samples. The reason this approach uses pairs of categorical and numerical columns is due to their assumption that the join keys are mostly string columns and the informative columns are numerical ones.

Figure 4.13 illustrates this structure. T_X and T_Y are two column pairs drawn from different tables in the data lake. In each pair, K_X (resp. K_Y) is the nonnumerical join key column and X (resp. Y) is the numerical candidate column that its correlation with the target needs to be calculated.

To build the sketch for each pair, the key values are first mapped to hash strings via a hash function $h(\cdot)$, producing the $h(k)$ column shown in the right-hand tables. A uniform hash function $h_u(\cdot)$ then maps each hash string to a value in $[0, 1]$, yielding the $h_u(k)$ column. The numerical feature values are carried over unchanged as x_k (resp. y_k). The two tables on the right of the arrow are the resulting sketches for T_X and T_Y respectively. Applying the K-minimum values (KMV) strategy, only the K rows with the smallest $h_u(k)$ values are kept, which reduces each full table to a compact fixed-size sample.

The joinability between the two sketches is established through $h(k)$ because both tables are hashed with the same function $h(\cdot)$. Two rows that share the same original key value will produce identical $h(k)$ and $h_u(k)$ entries. Consequently, rows with a matching $h(k)$ across the left and right sketches represent originally joinable rows. This allows join cardinality and correlation to be estimated directly on the small sketches rather than materializing the full join.

Note that using this proposed sketch in finding correlating columns yields approximate results because joinable rows are unknown at the time of indexing, and the generated sample may not perfectly represent joinable rows.

The above sketch-based index enables efficient join and correlation estimation in a two-step pipeline: first, joinable tables are discovered based on the hashed key values, and then correlation is calculated only on the retrieved tables. However, this does not necessarily address the objective misalignment issue, since the most informative, i.e., correlating, column may not be among the top-k joinable tables.

To answer this question, Santos et al. (Santos et al. 2022) propose an index that allows for simultaneous discovery of joinable and correlating tables. This new index leverages a quadrant count ratio (QCR) statistic to efficiently estimate the correlation between candidate columns and the target. QCR leverages the position of the values in each column to their corresponding averages. In particular, if most of the corresponding value pairs have the same position with regard to their average, i.e., either both are greater than the averages or both are smaller than the averages, columns are considered to be positively correlated. If the value pairs are mostly located in opposite positions in comparison to their corresponding averages, they are negatively correlated.

Figure 4.14 illustrates the quadrant count ratio (QCR), a nonparametric measure of association between two variables. The method partitions the bivariate space into four quadrants using the average of each variable ($\tilde{x}$ and $\tilde{y}$) as reference lines. Points falling in quadrants I (high X, high Y) and III (low X, low Y) represent *concordant* pairs, i.e., observations where both variables deviate from their mean in the same direction, while points in quadrants II (low X, high Y) and IV (high X,

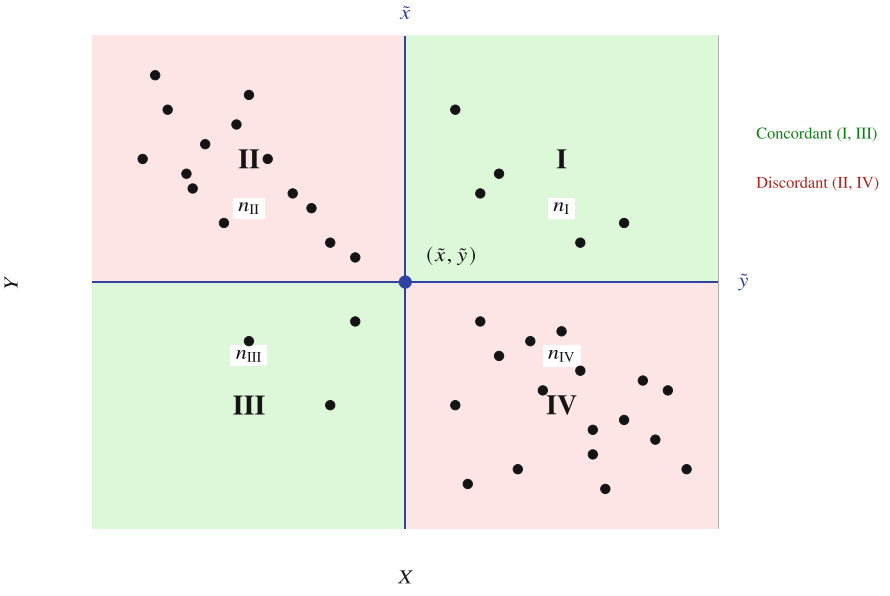


Fig. 4.14 Quadrant count ratio (QCR): data points are divided into four quadrants by the means $\tilde{x}$ and $\tilde{y}$. Concordant quadrants (I and III, shaded green) and discordant quadrants (II and IV, shaded red) determine the sign and magnitude of the QCR

low Y) represent *discordant* pairs with opposite deviations. The QCR is computed as the difference between concordant and discordant counts, normalized by the total number of observations:

$$\text{QCR} = \frac{(n_{\text{I}} + n_{\text{III}}) - (n_{\text{II}} + n_{\text{IV}})}{n}.$$

The measure ranges from -1 (perfect negative association) to $+1$ (perfect positive association), with values near zero indicating no monotonic relationship.

Example Consider ten observations of two variables X and Y, with means $\tilde{x} = \tilde{y} = 5.5$:

Row	X	Y	Quadrant
1	2	3	III (concordant: both below mean)
2	8	7	I (concordant: both above mean)
3	3	6	II (discordant: X below, Y above)
4	9	9	I (concordant)
5	1	2	III (concordant)
6	7	8	I (concordant)
7	4	4	III (concordant)
8	6	5	IV (discordant: X above, Y below)
9	5	2	III (concordant)
10	10	9	I (concordant)

Counting: $n_{\text{I}} = 4$, $n_{\text{II}} = 1$, $n_{\text{III}} = 4$, $n_{\text{IV}} = 1$, giving:

$$\text{QCR} = \frac{(4+4) - (1+1)}{10} = \frac{6}{10} = 0.6.$$

The score of 0.6 indicates a moderately strong positive association. Eight of the ten points agree in direction with the overall trend; the two discordant observations (row 3, where X is low but Y is high relative to their respective means, and row 8, where X is high but Y is low) reduce the score from a perfect 1.0.

Santos et al. (2022) follow the indexing approach depicted in Fig. 4.13 with one major difference. In this approach, the second hash function that maps the key values into numerical values between zero and one is updated as follows:

$$t_k = \begin{cases} h_u(h(k) \oplus +1) & \text{if } c_k - \mu_c > 0 \\ h_u(h(k) \oplus -1) & \text{if } c_k - \mu_c < 0. \end{cases}$$

In this formula, each hash incorporates not only the key value but also the position of the numeric value corresponding to the key value with respect to its average. This incorporates the correlation calculation into the join discovery process by concatenating the relative position of the numeric values for each pair. As a result,

tables retrieved by querying this new hash key are not only joinable on the key column, but their corresponding numerical values are also guaranteed to lie in the same quadrant, relative to their mean, as the numerical values of the query column.

4.4.2 *Exact Correlation Discovery*

Both of the previous index structures for correlation discovery are approximate solutions. Both approaches build their indexes by drawing a fixed-size sample of rows from each column pair offline, without knowledge of the query table that will arrive at the online phase. As a result, these approaches can lead to false positives and false negatives with regard to discovered features.

An example for an indexing approach that captures exact correlations is COCOA (Esmailoghli et al. 2021). COCOA focuses on nonlinear correlations, in particular rank-based correlations, such as Spearman's correlation and rank-biserial correlation coefficient (RBC), which enables capturing more complex associations between candidate features and the user target column. By doing so, COCOA captures the correlation of a target column not only with numerical features but also with categorical ones.

The core data structure in COCOA is the order index, which stores the relative position of values in each column. With the help of the order index, COCOA can reconstruct value orders, which are relevant for calculating rank-based scores, for each JOIN induced subset of the column without the need for sorting. This way, COCOA calculates rank-based correlation scores in linear time.

Figure 4.15 illustrates an example column and the order index built on top of it. On the bottom of the figure, the column values are shown accompanied by their row identifiers. The blue-colored value indicates the smallest value in the column and the red value shows the largest value of the column. To build the order index, the values must be sorted and ranks, i.e., the position of the value in the sorted

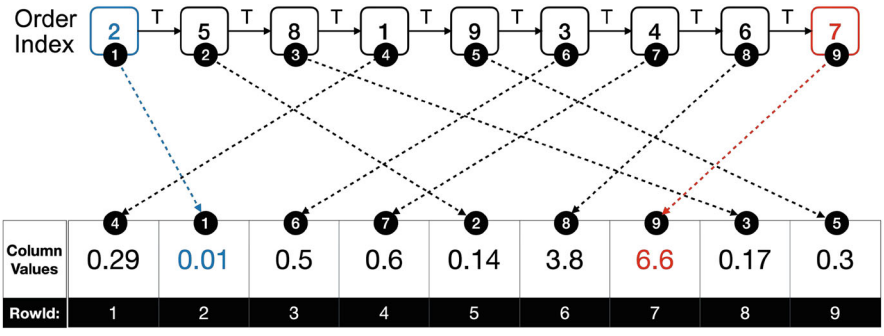


Fig. 4.15 The order index used in COCOA (Esmailoghli et al. 2021). Column values that are not in the linked list can be skipped

column must be obtained. These ranks are shown using small numbered circles, where the number is the rank of the value. The order index is the linked list of row identifiers where the values are sorted, meaning that the first element of the order index represents the row identifier of the smallest value and the last element refers to the row identifier of the largest value in the column. The linked lists are connected by a labeled edge, where the label is a binary flag indicating whether the next value is larger than the current value or they are identical. Note that this index can be generated for both numerical and nonnumerical columns. For categorical columns, COCOA creates one-hot encoding representations that are compatible with the order index.

COCOA operates in two stages: In the first stage, a *Table Finder* component uses an inverted index to identify the top-k joinable tables from the data lake on the user-specified query column q. The JOIN discovery stage can be replaced by any other JOIN discovery approach discussed in the earlier chapters. However, rather than materializing the full JOIN, a lightweight *Join Mapper* component constructs a *JoinMap* data structure for each candidate external table. The JoinMap records for each row in the external column the corresponding row from q. If a value in the external table has no match in q, the JoinMap entry is left empty ($\emptyset$). In the second stage, the *data augmenter* uses the order index together with the JoinMap to compute the correlation between the candidate column and the target column t, handling numerical and categorical columns differently.

Having the index in place and having obtained a *JoinMap* for a query table, COCOA calculates correlations as follows: **Numerical Columns.** To compute the Spearman's correlation coefficient (SCC) between a numerical external candidate column and the user-provided target column, COCOA traverses the order index from the minimum value to the maximum, following the linked list in the index. At each step, it consults the JoinMap to determine whether the current value participates in the join. If a mapping exists, the current rank, maintained incrementally during the traversal, is assigned to the corresponding row in the input dataset. Values without a joinable key are skipped, and the pointer advances along the outgoing link. This single pass assigns correct, consecutive ranks only to joinable rows.

Figure 4.16 illustrates this process for a numerical column of an external table. Starting from the minimum value (0.01), the algorithm checks the JoinMap and finds a match in row 3 of the query column, so rank 1 is assigned to that row. The pointer then advances to the next value (0.14), which has no JoinMap entry and is skipped. The following value (0.17) maps to row 4, which receives rank 2. This continues until the pointer reaches the maximum value (6.6) and no outgoing links remain. All operations of reading the JoinMap, updating the current rank, and advancing the pointer are performed in constant time. Furthermore, since each outgoing link refers to a unique row identifier, making the traversal cycle-free, the full SCC computation is achieved in $O(m)$.

Categorical Columns For categorical columns, COCOA computes the rank-biserial correlation coefficient (RBC) between each one-hot encoded feature of the candidate column and the target. A naive approach would explicitly generate all

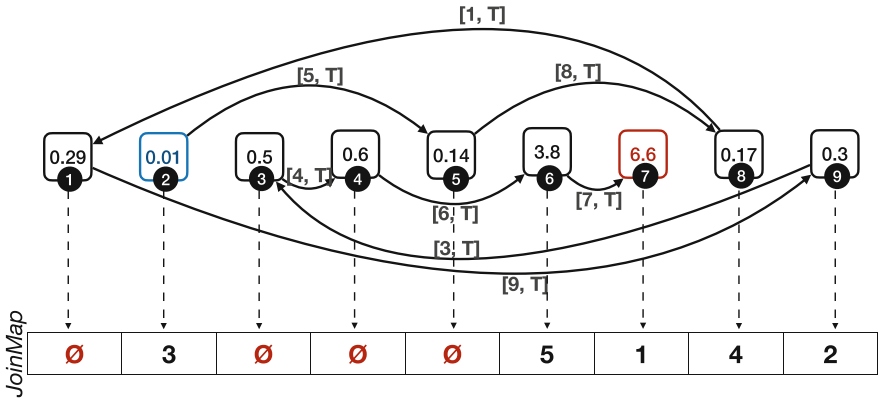


Fig. 4.16 Visual representation of the order index for the column `Area` together with the JoinMap, from the running example in COCOA (Esmailoghli et al. 2021). The pointer traverses the linked list from the minimum to the maximum value, assigning ranks only to rows that have a match in the JoinMap

one-hot features and compute the RBC for each, resulting in $O(m^2)$ complexity for a column with up to m unique values. COCOA avoids this by leveraging the order index to iterate over categorical values in sorted order, reading all occurrences of the same category consecutively. The binary edge labels in the order index signal when the traversal moves to a new category: a `True` label on an edge indicates that the next value is strictly greater, i.e., a different category, marking the boundary between two one-hot features.

Figure 4.17 illustrates this for a categorical column containing continent names. Red arrows depict that the next value is a different category. COCOA begins at the alphabetically minimum category and accumulates, for each group of equal values, the sum of corresponding target ranks s_1 and the count n_1. Once a `True`-labeled edge is encountered, the current one-hot feature is complete and its RBC is computed as

$$\text{RBC} = \frac{m \cdot s_1 - n_1 \cdot S}{std \cdot m \cdot \sqrt{n_1(m - n_1)}},$$

where S is the sum of all target ranks, std is their standard deviation, and m is the total number of rows. Since S, std, and m are computed once upfront, only s_1 and n_1 need to be updated per category. The maximum RBC across all one-hot features is reported as the final correlation score for the categorical column. As the algorithm reads the column only once, the complexity is $O(m)$.

Together, the order index and the JoinMap allow COCOA to compute exact rank-based correlations for both numerical and categorical columns in linear time, without materializing joins or explicitly generating one-hot encodings.

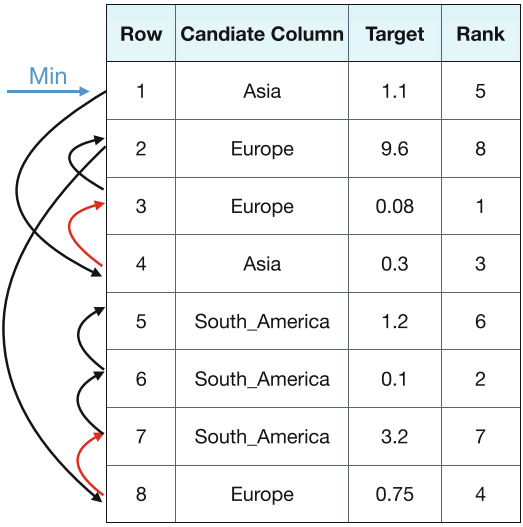


Fig. 4.17 Categorical candidate column example from COCOA (Esmailoghli et al. 2021). The order index enables iterating over values of the same category consecutively, so the RBC for all one-hot features is computed in a single pass

4.4.3 Discussion

Correlation-based indexes are the first attempt to move from related table discovery toward useful table discovery. Existing techniques can either capture certain types of proxy metrics, such as correlations, which limits the scope, or they require posterior verification of the obtained features after retrieval through feature selection Chepurko et al. (2020), which is more time-consuming.

4.5 Row-Based Indexes

The indexes discussed so far operate at the level of individual values or entire columns. Row-based indexes, on the other hand, map table rows or their compact representations to the tables that contain them. This level of abstraction is particularly useful for discovery methods that require simultaneous evaluation of entire rows or multiple row elements, such as multicolumn join discovery or transformation-based search.

As shown in Table 4.1, one of the primary row-based index structure is the *Row → Table* mapping. Two representative systems that leverage row-based indexes are Infogather (Yakout et al. 2012) and MATE (Esmailoghli et al. 2022).

Infogather (Yakout et al. 2012) indexes pairs of values within each row, mapping them to the table identifier in which they co-occur. The discovery approach proposed in Infogather leverages this row-based index to locate key-value pairs stored in table rows, enabling the system to find tables that contain specific attribute-value associations. For example, given a query requesting the population of a city, the

index can locate rows where the city name and population value co-occur, even if the schema of the candidate table differs from that of the query.

A more recent row-based indexing approach is the super key structure introduced by MATE (Esmailoghli et al. 2022, 2025). While strictly speaking, MATE's general approach has a column-based view, the hashing component can be considered a row-based index. As discussed in Sect. 4.2, MATE hashes the values within each row into a compact bit vector, called the super key, using the XASH hash function. The super key aggregates all column values in a row via bitwise OR, producing a bloom filter-like representation that captures composite-key membership at the row level. At query time, MATE computes the super key for the query composite key and checks candidate rows using a single bitwise AND operation, pruning rows whose super keys do not subsume the query.

4.6 Table-Based Indexes

Table-based indexes operate at the coarsest granularity. They map entire tables to their representations, such as embeddings or metadata summaries, and allow querying the data lake using a table identifier, an embedding, or a table metadata signature as the key.

Embedding tables into a latent space (Balaka et al. 2025; Liu et al. 2024; Santana and Ribeiro 2023; An et al. 2025; Eltabakh et al. 2024) is a table-level indexing approach used to discover semantically similar tables. Rather than comparing individual columns or values, the embedding captures the overall content and structure of a table in a dense vector. At query time, the system computes the embedding of the query table and retrieves the nearest neighbors from the index, typically using approximate nearest-neighbor structures such as hierarchical navigable small world (HNSW). These approaches are discussed in more detail in Sect. 4.8, where we cover the role of embeddings and foundation models in data discovery.

Table embeddings are particularly useful when the discovery objective is holistic similarity, i.e., when the user seeks tables that are about the same topic or describe the same entity type, rather than tables that share specific and exact column values. This makes table-level indexes a natural fit for tasks such as dataset recommendation, table retrieval from natural-language queries, and semantic deduplication.

The *Table* → *Metadata* index (Abedjan et al. 2016) maps table identifiers to available metadata describing the table, such as its source, schema information, column names, data types, and descriptive annotations. This index supports search for the available metadata of a given table without needing to scan its actual content. In profiling-based discovery scenarios, metadata indexes allow the system to quickly filter candidate tables by schema properties, data types, or annotations before performing more expensive content-based comparisons.

4.7 Graph-Based Indexes

Graph-based indexes generally use any of the aforementioned indexing strategies but apply them to capture the relationships between all elements inside a lake. In these structures, nodes typically represent columns or tables, and edges encode some form of similarity or relatedness between them. This structure enables exploration-based discovery, where a user navigates the graph to find relevant datasets rather than issuing a direct query. The general structure of graph indexes (Fernandez et al. 2018; Zhang and Ives 2020, 2019; Yakout et al. 2012; Bharadwaj et al. 2021; Nargesian et al. 2020; Ouellette et al. 2021; Nargesian et al. 2018a, 2023; Singh et al. 2016) is a graph where vertices represent columns and edges represent similarities between them. However, the specific construction and intended use of the graph vary across approaches.

4.7.1 Similarity-Based Navigation

Aurum (Fernandez et al. 2018) generates a graph index based on column similarities. In this graph, nodes represent columns and edges represent their Jaccard similarity scores. To construct the graph efficiently, Aurum builds multiple LSH indexes at different thresholds and assigns the weight of each edge as the highest threshold at which the two columns are similar. This graph enables similarity-based navigation: given a column of interest, the user can explore related columns by traversing edges, progressively discovering tables that share content with the starting point. This form of exploration is particularly valuable when the user does not have a precise query in mind but instead wants to understand what related data exists in the lake.

Infogather (Yakout et al. 2012) also uses a graph-based structure, where it builds an entity complement graph that encodes which tables can augment each other with additional attributes. The nodes in this graph represent tables, and edges connect tables that share entities but provide complementary information. This structure supports a different kind of navigation, where rather than finding similar tables, the user discovers tables that can enrich their current dataset.

4.7.2 Topic-Driven Navigation

Nargesian et al. (Nargesian et al. 2020; Ouellette et al. 2021; Nargesian et al. 2023) propose a graph index for topic-driven navigation. They generate a tree structure based on the semantic domain of the columns in the data lake. In this hierarchical graph, non-leaf nodes represent topics or semantic categories, while leaf nodes represent individual columns. The user can navigate through the topics, moving

down from broad categories to specific columns of interest. This is conceptually different from similarity-based navigation. Instead of following content overlap, the user follows semantic categories to locate data. This approach is particularly effective when the data lake contains tables from diverse domains and the user needs to first identify the relevant domain before searching for specific columns.

4.7.3 Knowledge Graph-Enhanced Discovery

Beyond similarity and topic-based graphs, several approaches leverage external knowledge graphs to enhance data discovery. Zhang and Ives (Zhang and Ives 2020, 2019) construct a discovery graph where nodes represent columns and edges capture relationships derived from both content overlap and external knowledge sources. By incorporating domain knowledge into the graph structure, the system can identify relationships between columns that share no syntactic overlap but are semantically related through an ontology or knowledge base. Similarly, Bharadwaj et al. (2021) built a graph structure that connects datasets through semantic relationships, enabling users to discover datasets that are related not just by content but by meaning.

4.7.4 Discussion

The graph-based index is the only index category that explicitly models inter-dataset relationships rather than properties of individual datasets. This makes it uniquely suited for exploratory data discovery, where the user's information need is not fully specified upfront. However, graph indexes come with their own trade-offs. Constructing the graph requires computing pairwise similarities or relationships, which can be expensive for large data lakes. Maintaining the graph as new tables are added or existing ones are modified requires efficient update mechanisms. Despite these challenges, graph-based indexes remain an important component of the indexing, as they support a mode of discovery, i.e., navigation and exploration, which the other index categories cannot provide on their own.

4.8 Embedding-Based and Foundation Model Approaches

The index structures discussed in this chapter, e.g., LSH, prefix filtering, correlation sketches, and order indexes, are designed around explicit representations of column content, e.g., sets of values, sorted ranks, or hash signatures. A common limitation of this family of approaches is that they merely operate on the syntactic aspect of data, with little or no understanding of its semantics. Two columns containing the

words “automobile” and “car” will appear disjoint to a set-overlap index, yet they are semantically interchangeable. Similarly, a join-key column whose values are country codes rather than country names will not match a query column with full country names, even though the columns are functionally joinable.

The emergence of pretrained language models and large language models (LLMs) has opened new possibilities for data discovery. Rather than indexing the literal content of columns, one can index their *meaning*. Dense vector embeddings map column values, schema elements, or entire tables into a high-dimensional space where semantic similarity corresponds to geometric proximity. This shift enables new index structures, i.e., approximate nearest-neighbor (ANN) indexes over embedding vectors, and also motivates hybrid architectures that pair embedding-based retrieval with traditional search. At the far end of the spectrum, foundation models are applied directly as reasoning engines, bypassing dedicated index structures altogether and relying on the parametric knowledge encoded in the model weights.

We organize the approaches surveyed in this section into five clusters that reflect the degree to which they rely on a dedicated index structure versus pure model inference.

4.8.1 ANN Indexes over Dense Column Embeddings

The most direct adaptation of traditional index design to the embedding era replaces stored values with dense vectors and substitutes exact lookup with approximate nearest-neighbor (ANN) search. Addressing the question of *“how to generate the column embedding”* admits several answers, each optimized for a different discovery objective, yet all converge on the same index structure at query time.

The simplest strategy encodes raw column values using a pretrained model and uses the resulting vectors directly. This is sufficient when joinability is defined by semantic proximity rather than exact value overlap. A similarity join in this setting retrieves all embedding vectors from one table that fall within a distance threshold ϵ of a given query vector from the other table (Santana and Ribeiro 2023). Existing set-similarity join algorithms are designed for sparse representations and cannot handle this case. This problem motivates a dedicated ANN index over the embedding space. The hierarchical navigable small world (HNSW) graph addresses this problem (Bogatu et al. 2020). HNSW constructs a layered proximity graph during indexing. Each node is connected to its nearest neighbors at multiple granularity levels. Greedy traversal at query time locates approximate nearest neighbors in sublinear time. Its combination of low construction overhead, fast query performance, and tolerance for high-dimensional vectors makes it the preferred ANN backbone for data lake discovery.

A more powerful strategy learns the embedding rather than adopting a generic encoder. Contrastive learning has proven particularly effective for this problem. The idea behind contrastive learning is to treat relevant entities, i.e., columns

and tables, as positive pairs and unrelated ones as negatives. Starmie (Fan et al. 2023) trains a RoBERTa-based encoder to bring positive pairs close and push negatives apart for the problem of union discovery. The resulting representations capture the distributional semantics of a column's content in a way that is robust to syntactic variations. All column embeddings are inserted into an HNSW index at indexing time, and the embedding of a query column probes the index at retrieval time, recovering union candidates even when overlapping values are synonyms or paraphrases.

A third variant changes neither the encoder architecture nor its training objective, but instead enriches what gets embedded. Rather than feeding raw column values to the model, an LLM first reads several features of context about the column, e.g., its source, its position in the table, its neighboring columns, its value distribution, and any available description, and writes a short natural-language narration that summarizes all of them (An et al. 2025). That narration, rather than the raw values, is what gets encoded into a dense vector and stored in an HNSW-backed vector database. The intuition is that a well-written description conveys meaning more accurately than a list of raw cell values, so the resulting embedding is a better proxy for what the column actually represents. The same index can also be used for catalog construction, i.e., clustering semantically related columns together and asking the LLM to label each cluster with the goal of producing a hierarchical summary of the data lake without any manual curation.

4.8.2 Hybrid Retrieval: Combining Full-Text and Vector Indexes

No single retrieval modality covers the full range of queries a discovery system must handle. Exact search is useful when the query shares exact vocabulary with the indexed content, but breaks down under terminology mismatch or paraphrase. Embedding-based search handles synonym and semantic variation well, but can falter on highly domain-specific queries that fall outside the training distribution of the embedding model. One natural response is to maintain both indexes in parallel and merge their results, leading to a hybrid architecture employing both modalities.

One application of hybrid retrieval is data cleaning (Eltabakh et al. 2024). The discovery task here is to find tuples within the data lake that are similar to a dirty tuple and can serve as clean reference examples. Those discovered tuples are then presented to an LLM, which uses them as in-context examples to generate the corrected value in a retrieval-augmented generation (RAG) scenario. Because the dirty tuple may match its reference either by shared keywords or by semantic similarity, a single index is not enough. An inverted index handles keyword matches and a Faiss vector index (Douze et al. 2025) over sentence embeddings handles semantic ones. Both are queried, their result lists are merged, and the combined set of discovered tuples is handed to the LLM as context.

Another application of hybrid retrieval is natural-language table search (Balaka et al. 2025). The discovery task here is to locate, within a lake, the tables that best match a user's plain-English description. The difficulty is that tables are indexed by structured content while the query is not. To make the two sides comparable, each table is first described in natural language, where an LLM writes a short narration of each column, expands abbreviations, and appends a few sample rows. That text is then indexed in both an inverted index and a vector index. At query time, the user's question is issued against both, and the merged result list is passed to the LLM not to rank all candidates from scratch but only to filter out the ones that are clearly irrelevant. This keeps inference costs low while adding the semantic judgment that the indexes alone cannot provide.

4.8.3 LLM-Augmented Discovery Pipelines with Lightweight Indexing

Not every LLM-based discovery system needs a primary embedding index. Some tasks have a natural division of labor: A traditional index handles the parts that can be expressed as similarity queries, and the LLM handles the parts that cannot, such as causal reasoning, domain knowledge, and contextual judgment.

Causal dataset discovery is a good example of such an approach (Liu et al. 2024). The goal is to find columns in a data lake that are causally related to a given target variable, not just correlated with it. No index can make that distinction on its own, since causality depends on domain knowledge rather than column content. First, a hash join over a shared key column generates an initial set of joinable tables. Second, a Faiss ANN index over FastText embeddings removes near-duplicate column pairs, keeping the candidate set clean. Third, mutual information between each candidate column and the target is estimated on the join result, and weak candidates are discarded. Finally, the non-filtered pairs are presented to an LLM with a prompt asking it to classify each relationship as causal, reverse-causal, or spurious. The index contributes efficiency and cheap deduplication at scale, while the LLM contributes the one thing the index cannot, i.e., an understanding of what a causal relationship actually means.

4.8.4 Self-Supervised Representation Learning for Table Relationships

The approaches discussed so far all rely on some form of index that is queried at retrieval time. A different philosophy eliminates the index entirely. Instead, a neural encoder is trained offline so that its output at inference time is already a relevance score for a candidate column pair. There is no separate lookup step. The model

is both the index and the retrieval function. Training relies on self-supervised or multitask objectives, so large manually labeled datasets are not required.

When the discovery task is column relatedness, i.e., deciding whether two columns are related enough to support a join or union, one approach trains a model that scores column pairs directly, without building any index (Hu et al. 2025). The key insight is that this scoring task improves when the model is simultaneously trained on related discovery tasks, such as schema classification, entity matching, and schema prediction. All of them require the same structural and semantic understanding, so training on all of them together produces a stronger model than training on any one alone. A self-adaptive loss function balances their contributions during training, and the auxiliary training data is generated by the LLM itself, avoiding manual annotation. At inference time, the system scores a column pair without involving any conventional index.

For union search, the index-free approach addresses a specific discovery failure: two columns can be perfectly unionable yet have individual representations that are different (Hu et al. 2023). One might use full country names while another uses ISO codes; one might be clean while the other is noisy or abbreviated. Scoring each column in isolation and then comparing vectors misses these matches. The solution is to encode the pair jointly. In this approach, AutoTUS, a BERT-based encoder, takes both columns as input at once and produces a single vector representing the relationship between them, not the content of either column in isolation. The encoder is trained without labels. The initial clustering assigns pseudo-classes to similar pairs, and subsequent rounds refine the representations. This approach adds synthetic noise to prepare the model for messy tables found in real data lakes. At retrieval time, the encoder's output is the discovery decision; similarly, no external index is needed.

4.8.5 Foundation Models as Semantic Reasoners

At the far end of the spectrum are approaches that build no index at all. The idea is that a foundation model trained on a large corpus has already absorbed a great deal of knowledge about datasets, schemas, domain terminology, and real-world relationships. If the queried data is well known enough to appear in that training corpus, the model may be able to extract it from its weights alone, without looking up any index.

This paradigm has been applied in three distinct ways. The first is conversational search (Walker et al. 2023). Instead of typing keywords into a data portal, a user describes what they need in plain language and has a dialogue with the model. The model can suggest relevant datasets, explain why each might be useful, describe its provenance, and flag potential quality issues. The limitation is hallucination and limited knowledge. The model may confidently suggest datasets that do not exist, and it only knows what was in its training data. If the relevant dataset is proprietary or was published recently, the model will not find it.

The second application treats the model not as a search interface but as a discovery agent (Majumder et al. 2024). Given a collection of datasets, the system searches for scientific hypotheses on its own, without the user specifying what to look for. It conducts this through a team of specialized agents. One plans the investigation, one writes statistical analysis code, one interprets the results, and one critiques and revises the conclusions. Together, they run regressions and statistical tests on the provided data to confirm or reject candidate hypotheses. The datasets are loaded directly into the model's context or into a code execution environment without employing any external index. Discovery here is entirely a reasoning problem, and the LLM plays every role in the process.

A third application is the discovery of the semantic identity of tables and columns in a data lake, addressing the questions of what real-world entity type a table represents, what semantic type each column holds according to a standard ontology, and which column pairs should serve as join keys (Kayali et al. 2024). All three are discovery problems. The approach poses each of them as a prompting problem. Table content and metadata are serialized into the prompt, and the model produces its answer in natural language. One practical challenge is that the model's answer does not always name a valid ontology entry. An *anchoring* step handles this by substituting the nearest valid entry using embedding similarity when the output fails a validity check, i.e., guardrailing.

Summary

Table 4.3 summarizes the index structures and discovery objectives of the approaches surveyed in this section. Moving from top to bottom in the table, the role of the dedicated index diminishes while the role of the foundation model expands. This progression reflects a fundamental trade-off: dedicated index structures (HNSW, BM25, Faiss) offer sublinear query time and predictable latency guarantees, but require offline construction and capture only the features built into their structure. Foundation models offer flexible, semantics-aware reasoning that generalizes across tasks, but at substantially higher inference cost and with no formal guarantees on output quality. Hybrid architectures, such as Pneuma and RetClean, suggest that the most practical systems for large-scale data lakes will combine fast index-based candidate generation with LLM-based semantic refinement.

4.9 Coresets for Discovery Indexes

The indexing strategies surveyed in this chapter share a common structural assumption that the index is built from the complete content of each dataset in the lake, or from sketches of that content such as MinHash signatures or LSH buckets that are computed over all tuples. In very large lakes or in federated settings where datasets reside across multiple providers, this assumption becomes problematic.

Table 4.3 Summary of embedding-based and foundation model approaches for data discovery, organized by primary index structure

System	Index structure	LLM role	Task
Santana and Ribeiro (2023)	HNSW (embedding ANN)	None	Similarity join
Starmie (Fan et al. 2023)	HNSW (contrastive embedding)	None (training only)	Join and union
LEDD (An et al. 2025)	Milvus/HNSW vector store	Narration + clustering	Semantic search
RetClean (Eltabakh et al. 2024)	BM25 + Faiss/HNSW	Imputation (RAG)	Data cleaning
Pneuma (Balaka et al. 2025)	BM25 + vector index	Narration + reranking	Table retrieval
Liu et al. (2024)	Faiss (FastText) + hash join	Causal classification	Causal discovery
DiscoverGPT (Hu et al. 2025)	None (fine-tuned encoder)	Multitask scoring	Column relatedness
AutoTUS (Hu et al. 2023)	None (pair encoder)	Contextual pair encoding	Table union
Walker et al. (2023)	None	Dataset recommendation	Dataset discovery
DataVoyager (Majumder et al. 2024)	None	Hypothesis search	Scientific discovery
Chorus (Kayali et al. 2024)	None (anchoring fallback)	Zero/few-shot annotation	Table/column tasks

Transmitting and processing entire datasets to build or refresh an index is expensive, and in the federated case, it may be prohibited by governance constraints. A natural question is therefore: is it possible to build an accurate discovery index from a principled compressed representation of each dataset, one that is much smaller than the full dataset but probably preserves the properties that discovery queries depend on?

This question is addressed by the framework of *coresets*. In the geometric algorithms literature, a coreset for a dataset P with respect to a query class Q is a weighted subset $S \subseteq P$ such that for every query $q \in Q$, evaluating q on S (using the weights) approximates the result of evaluating q on P within a controlled additive or multiplicative error (Har-Peled 2011). The key property is that the size of S depends on the desired approximation error and the complexity of the query class, not on the size of P. This makes coreset-based indexes attractive precisely in the settings where conventional full-data indexes are infeasible.

4.9.1 Distribution-Aware Discovery and the Coreset Construction Problem

Esmailpour et al. (2025) introduced the first theoretically grounded framework that applies coreset ideas directly to data discovery, specifically to the problem of *distribution-aware dataset search*. Rather than asking which datasets are joinable or unionable with a query table, this discovery operator asks which datasets in the repository satisfy distributional predicates specified by the user. Two classes of predicates are studied.

The *percentile indexing problem* (Ptile) asks the discovery system to identify all datasets P_j in the repository for which the fraction of points falling within a user-specified region R of the attribute space lies within a specified interval θ. Intuitively, this corresponds to queries such as find all datasets in which at least 30% of records have income values between $40,000 and $80,000. The *preference indexing problem* (Pref) generalizes this to directional preferences: it asks for datasets in which the k-th largest projection of the data along a user-specified direction vector lies within an interval. Preference queries capture richer distributional structure than percentile queries, enabling discovery criteria such as find datasets in which the top decile of values along a composite socioeconomic dimension exceeds a specified threshold.

Both query classes are formally defined over a repository $\mathcal{P} = \{P_1, \ldots, P_N\}$ of N datasets, where each $P_i \subset \mathbb{R}^d$ for constant dimension d. The goal is to build a data structure over $\mathcal{P}$ that, given a query predicate, reports all datasets satisfying it efficiently.

Hardness and Index Design A key theoretical contribution of the framework is a hardness result establishing that *near-linear space data structures with polylogarithmic query time cannot be achieved in the centralized setting* for these query classes, under standard computational geometry assumptions. This result is significant for

discovery system designers: it shows that the performance regime achievable for simpler discovery operators such as exact join search cannot be replicated for distributional discovery without accepting either larger index size or longer query time.

Given this lower bound, the authors design *approximate* data structures that operate within an $\tilde{O}(N)$ space budget and answer both Ptile and Pref queries in $\tilde{O}(1 + \text{OUT})$ time, where OUT is the output size. The approximation guarantee is controlled by two parameters: an approximation error $\varepsilon \in (0, 1)$ and a synopsis approximation error δ. Concretely, the index guarantees two properties simultaneously. First, every dataset that truly satisfies the query predicate is returned (no false negatives). Second, every dataset returned by the index satisfies the predicate up to an additive error of $\varepsilon + 2\delta$ (controlled false positives). This asymmetric guarantee of exact recall with bounded precision error is well-suited to discovery contexts where missing a relevant dataset is more costly than retrieving a marginally ineligible one.

Federated Setting and Synopsis-Based Indexing The framework is explicitly designed to operate in both the centralized and federated discovery settings introduced in Chap. 2. In the centralized setting, the index is constructed from direct access to each dataset's tuples. In the federated setting, only a *synopsis* of each dataset is available, a compact summary released by the data provider that captures the distributional properties needed for query evaluation without exposing individual records.

This is where the coreset connection becomes concrete. The synopsis plays the role of the coreset: it is a compressed representation of P_i that must preserve, for every query in the supported class, an answer within the error budget δ. The construction of a suitable synopsis is therefore a coreset construction problem for the Ptile and Pref query classes. The authors show that ε-approximations of the underlying point set—weighted subsets that approximate the empirical measure of P_i within ε in the relevant metric—serve as valid synopses, and that such approximations can be constructed with size independent of $|P_i|$, depending only on d and the desired error.

In the federated setting, the end-to-end index therefore consists of a collection of small synopses, one per dataset provider, that are transmitted to the discovery system in place of the raw data. The discovery system builds its query data structure over these synopses rather than the original datasets. The total space and query time guarantees are preserved, with the additive 2δ term in the precision error reflecting the information loss introduced by synopsis compression.

4.10 Conclusion

This chapter surveyed the landscape of index structures that make data discovery feasible and efficient at the scale of modern data lakes. The progression from simpler to more sophisticated indexes mirrors a broadening notion of what it means for two tables or columns to be related.

Value-based indexes, i.e., the inverted index, provide the exact set-level overlap that underpins join and keyword search operators. Refinements such as prefix and positional filtering, top-k pruning (Dong et al. 2021), and super-key hashing (Esmailoghli et al. 2022) push the boundaries of exact discovery, scaling from single-column to composite-key scenarios while maintaining storage efficiency.

Column-based LSH indexes trade exactness for speed: MinHash LSH approximates Jaccard similarity in sublinear time, LSH Ensemble corrects the inherent bias toward smaller columns by switching to containment similarity, and LAZO (Fernandez et al. 2019) reduces construction overhead through one-permutation sketching. Together, the value-based and column-based index families efficiently address the *what is joinable* question across hundreds of millions of columns.

Correlation-oriented indexes shift the question from *what can be joined* to *what is worth joining*. KMV sketches and QCR-based hashing allow correlation to be estimated jointly with joinability, avoiding the expensive post-join evaluation step (Santos et al. 2021, 2022). COCOA's order index goes further (Esmailoghli et al. 2021), enabling exact rank-based correlation for both numerical and categorical columns in linear time by interleaving join mapping with index traversal.

Row-based indexes operate at an intermediate granularity, mapping rows or their compact representations to the tables that contain them. MATE's super-key structure enables multicolumn join discovery with bloom filter-like efficiency, while Infogather (Yakout et al. 2012; Zhang and Chakrabarti 2013) and DataXFormer (Abedjan et al. 2015, 2016) exploit row-level co-occurrence for entity lookup and transformation discovery. Table-based indexes operate at the coarsest level, mapping entire tables to dense embeddings or metadata summaries for holistic similarity search and schema-level filtering. Graph-based indexes take a fundamentally different approach: rather than indexing properties of individual datasets, they capture relationships between datasets, supporting exploration-based discovery through similarity navigation (Fernandez et al. 2018), topic-driven hierarchies (Nargesian et al. 2018b), and knowledge graph-enhanced search.

Embedding-based and foundation model approaches represent the most recent research frontier, moving beyond syntactic content to capture semantic meaning. ANN indexes over dense vectors, such as HNSW, handle complex semantics including synonym matching, whether through generic pretrained encoders or task-specific contrastive learning (Fan et al. 2023). Hybrid architectures pair inverted indexes with vector stores to handle both vocabulary-dependent and vocabulary-independent queries (Eltabakh et al. 2024; Balaka et al. 2025). LLM-augmented pipelines reserve model inference for reasoning steps that no index can perform, such as causal classification. Self-supervised encoders such as DiscoverGPT (Hu et al. 2025) and AutoTUS (Hu et al. 2023) eliminate the index entirely, replacing lookup with a learned scoring function that operates directly on column pairs. At the far end of the spectrum, foundation models serve as zero-shot reasoning engines, answering discovery questions from their parametric knowledge alone.

The central trade-off across this entire spectrum is between predictable, sublinear query latency and flexible semantic judgment. Dedicated index structures guarantee the former; foundation models provide the latter at substantially higher inference

cost. The most practical systems for large-scale data lakes will likely combine both, using fast index-based candidate generation followed by LLM-based semantic refinement, a pattern already visible in RetClean (Eltabakh et al. 2024) and Pneuma (Balaka et al. 2025). Choosing the right index, or combination of indexes, depends on the discovery operator to be accelerated, the scale of the lake, the acceptable error budget, and whether the user's intent is best captured syntactically or semantically.

References

Abedjan Z, Morcos J, Gubanov MN, Ilyas IF, Stonebraker M, Papotti P, Ouzzani M (2015) Dataxformer: Leveraging the web for semantic transformations. In: Proceedings of the Conference on Innovative Data Systems Research (CIDR)

Abedjan Z, Morcos J, Ilyas IF, Ouzzani M, Papotti P, Stonebraker M (2016) Dataxformer: A robust transformation discovery system. In: Proceedings of the International Conference on Data Engineering (ICDE), pp 1134–1145

Amsterdamer Y, Cohen M (2021) Automated selection of multiple datasets for extension by integration. In: Demartini G, Zuccon G, Culpepper JS, Huang Z, Tong H (eds) Proceedings of the International Conference on Information and Knowledge Management (CIKM), ACM, pp 27–36, URL https://doi.org/10.1145/3459637.3482322

An Q, Ying C, Zhu Y, Xu Y, Zhang M, Wang J (2025) LEDD: large language model-empowered data discovery in data lakes. CoRR abs/2502.15182, URL https://doi.org/10.48550/arXiv.2502.15182,

Balaka MIL, Alexander D, Wang Q, Gong Y, Krisnadhi A, Fernandez RC (2025) Pneuma: Leveraging llms for tabular data representation and retrieval in an end-to-end system. Proceedings of the ACM on Management of Data (PACMMOD) 3(3):200:1–200:28, URL https://doi.org/10.1145/3725337

Bawa M, Condie T, Ganesan P (2005) LSH forest: self-tuning indexes for similarity search. In: Ellis A, Hagino T (eds) Proceedings of the International World Wide Web Conference (WWW), ACM, pp 651–660, URL https://doi.org/10.1145/1060745.1060840

Bayardo RJ, Ma Y, Srikant R (2007) Scaling up all pairs similarity search. In: Williamson CL, Zurko ME, Patel-Schneider PF, Shenoy PJ (eds) Proceedings of the International World Wide Web Conference (WWW), ACM, pp 131–140, URL https://doi.org/10.1145/1242572.1242591

Becktepe J, Esmailoghli M, Koch M, Abedjan Z (2023) Demonstrating MATE and COCOA for data discovery. In: Proceedings of the International Conference on Management of Data (SIGMOD), ACM, pp 119–122, URL https://doi.org/10.1145/3555041.3589716

Bharadwaj S, Gupta P, Bhagwan R, Guha S (2021) Discovering related data at scale. Proceedings of the VLDB Endowment (PVLDB) 14(8):1392–1400, URL http://www.vldb.org/pvldb/vol14/p1392-bharadwaj.pdf

Bloom BH (1970) Space/time trade-offs in hash coding with allowable errors. Communications of the ACM 13(7):422–426

Bogatu A, Fernandes AAA, Paton NW, Konstantinou N (2020) Dataset discovery in data lakes. In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE, pp 709–720, URL https://doi.org/10.1109/ICDE48307.2020.00067

Bouros P, Ge S, Mamoulis N (2012) Spatio-textual similarity joins. Proceedings of the VLDB Endowment (PVLDB) 6(1):1–12, URL https://doi.org/10.14778/2428536.2428537

Broder AZ (1997a) On the resemblance and containment of documents. In: Proceedings of the Compression and Complexity of Sequences, pp 21–29, DOI https://doi.org/10.1109/SEQUEN.1997.666900

Broder AZ (1997b) On the resemblance and containment of documents. In: Carpentieri B, Santis AD, Vaccaro U, Storer JA (eds) Compression and Complexity of SEQUENCES 1997, Positano,

Amalfitan Coast, Salerno, Italy, June 11-13, 1997, Proceedings, IEEE, pp 21–29, URL https://doi.org/10.1109/SEQUEN.1997.666900

Charikar M (2002) Similarity estimation techniques from rounding algorithms. In: Symposium on Theory of Computing, ACM, pp 380–388

Chaudhuri S, Ganti V, Kaushik R (2006) A primitive operator for similarity joins in data cleaning. In: 22nd International Conference on Data Engineering (ICDE'06), IEEE, pp 5–5

Chepurko N, Marcus R, Zgraggen E, Fernandez RC, Kraska T, Karger DR (2020) ARDA: automatic relational data augmentation for machine learning. Proceedings of the VLDB Endowment (PVLDB) 13(9):1373–1387, DOI https://doi.org/10.14778/3397230.3397235, URL http://www.vldb.org/pvldb/vol13/p1373-chepurko.pdf

Christensen MP, Leventidis A, Lissandrini M, Rocco LD, Miller RJ, Hose K (2025) Fantastic tables and where to find them: Table search in semantic data lakes. In: Simitsis A, Kemme B, Queralt A, Romero O, Jovanovic P (eds) Proceedings of the International Conference on Extending Database Technology (EDBT), OpenProceedings.org, pp 397–410, URL https://doi.org/10.48786/edbt.2025.32

Deng D, Li G, Hao S, Wang J, Feng J (2014) Massjoin: A mapreduce-based method for scalable string similarity joins. In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE Computer Society, pp 340–351

Deng D, Li G, Wen H, Feng J (2015) An efficient partition based method for exact set similarity joins. Proceedings of the VLDB Endowment (PVLDB) 9(4):360–371

Dong Y, Takeoka K, Xiao C, Oyamada M (2021) Efficient joinable table discovery in data lakes: A high-dimensional similarity-based approach. In: 37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021, IEEE, pp 456–467, URL https://doi.org/10.1109/ICDE51399.2021.00046

Dong Y, Xiao C, Nozawa T, Enomoto M, Oyamada M (2023) Deepjoin: Joinable table discovery with pre-trained language models. Proceedings of the VLDB Endowment (PVLDB) 16(10):2458–2470, DOI https://doi.org/10.14778/3603581.3603587, URL https://www.vldb.org/pvldb/vol16/p2458-dong.pdf

Douze M, Guzhva A, Deng C, Johnson J, Szilvasy G, Mazaré PE, Lomeli M, Hosseini L, Jégou H (2025) The faiss library. URL https://arxiv.org/abs/2401.08281,

Eltabakh MY, Naeem ZA, Ahmad MS, Ouzzani M, Tang N (2024) Retclean: Retrieval-based tabular data cleaning using llms and data lakes. Proceedings of the VLDB Endowment (PVLDB) 17(12):4421–4424, DOI https://doi.org/10.14778/3685800.3685890, URL https://www.vldb.org/pvldb/vol17/p4421-eltabakh.pdf

Esmailoghli M, Quiané-Ruiz J, Abedjan Z (2021) COCOA: correlation coefficient-aware data augmentation. In: Velegrakis Y, Zeinalipour-Yazti D, Chrysanthis PK, Guerra F (eds) Proceedings of the International Conference on Extending Database Technology (EDBT), OpenProceedings.org, pp 331–336, URL https://doi.org/10.5441/002/edbt.2021.30

Esmailoghli M, Quiané-Ruiz J, Abedjan Z (2022) MATE: multi-attribute table extraction. Proceedings of the VLDB Endowment (PVLDB) 15(8):1684–1696, URL https://www.vldb.org/pvldb/vol15/p1684-esmailoghli.pdf

Esmailoghli M, Schnell C, Miller RJ, Abedjan Z (2025) BLEND: A Unified Data Discovery System . In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE Computer Society, Los Alamitos, CA, USA, pp 737–750, DOI https://doi.org/10.1109/ICDE65448.2025.00061, URL https://doi.ieeecomputersociety.org/10.1109/ICDE65448.2025.00061

Esmailpour A, Galhotra S, Raychaudhury R, Sintos S (2025) A theoretical framework for distribution-aware dataset search. Proc ACM Manag Data 3(2):90:1–90:26, URL https://doi.org/10.1145/3725227

Fan G, Wang J, Li Y, Zhang D, Miller RJ (2023) Semantics-aware dataset discovery from data lakes with contextualized column-based representation learning. Proceedings of the VLDB Endowment (PVLDB) 16(7):1726–1739, URL https://www.vldb.org/pvldb/vol16/p1726-fan.pdf

Fernandez RC, Abedjan Z, Koko F, Yuan G, Madden S, Stonebraker M (2018) Aurum: A data discovery system. In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE Computer Society, pp 1001–1012, URL https://doi.org/10.1109/ICDE.2018.00094

Fernandez RC, Min J, Nava D, Madden S (2019) Lazo: A cardinality-based method for coupled estimation of jaccard similarity and containment. In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE, pp 1190–1201, URL https://doi.org/10.1109/ICDE.2019.00109

Galhotra S, Gong Y, Fernandez RC (2023) Metam: Goal-oriented data discovery. In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE, pp 2780–2793, URL https://doi.org/10.1109/ICDE55515.2023.00213

Gionis A, Indyk P, Motwani R, et al. (1999) Similarity search in high dimensions via hashing. Proceedings of the VLDB Endowment (PVLDB) 99(6):518–529

Har-Peled S (2011) Geometric approximation algorithms. 173, American Mathematical Soc.

Hu X, Wang S, Qin X, Lei C, Shen Z, Faloutsos C, Katsifodimos A, Karypis G, Wen L, Yu PS (2023) Automatic table union search with tabular representation learning. In: Rogers A, Boyd-Graber JL, Okazaki N (eds) Findings of the Association for Computational Linguistics: (ACL), Association for Computational Linguistics, pp 3786–3800, URL https://doi.org/10.18653/v1/2023.findings-acl.233

Hu X, Qin X, Lei C, Katsifodimos A, Shen Z, Srinivasan B, Rangwala H (2025) Discovergpt: Multitask fine-tuning large language model for related table discovery. In: Chiruzzo L, Ritter A, Wang L (eds) Findings of the Association for Computational Linguistics: NAACL 2025, Albuquerque, New Mexico, USA, April 29 - May 4, 2025, Association for Computational Linguistics, pp 358–373, URL https://doi.org/10.18653/v1/2025.findings-naacl.21

Kayali M, Lykov A, Fountalis I, Vasiloglou N, Olteanu D, Suciu D (2024) CHORUS: foundation models for unified data discovery and exploration. Proceedings of the VLDB Endowment (PVLDB) 17(8):2104–2114, DOI https://doi.org/10.14778/3659437.3659461, URL https://www.vldb.org/pvldb/vol17/p2104-kayali.pdf

Liu J, Sun S, Nargesian F (2024) Causal dataset discovery with large language models. In: Fekete J, Omidvar-Tehrani B, Rong K, Shraga R (eds) Proceedings of the Workshop on Human-In-the-Loop Data Analytics, HILDA 24, Santiago, Chile, 14 June 2024, ACM, pp 1–8, URL https://doi.org/10.1145/3665939.3665968

Lv Q, Josephson W, Wang Z, Charikar M, Li K (2007) Multi-probe LSH: efficient indexing for high-dimensional similarity search. In: Proceedings of the International Conference on Very Large Databases (VLDB), ACM, pp 950–961, URL http://www.vldb.org/conf/2007/papers/research/p950-lv.pdf

Majumder BP, Surana H, Agarwal D, Hazra S, Sabharwal A, Clark P (2024) Position: Data-driven discovery with large generative models. In: International Conference on Machine Learning (ICML), OpenReview.net, URL https://openreview.net/forum?id=5SpjhZNXtt

Mann W, Augsten N (2014) PEL: position-enhanced length filter for set similarity joins. In: GI-Workshop Grundlagen von Datenbanken, CEUR-WS.org, CEUR Workshop Proceedings, vol 1313, pp 89–94

Mann W, Augsten N, Bouros P (2016) An empirical evaluation of set similarity join techniques. Proceedings of the VLDB Endowment (PVLDB) 9(9):636–647

Manning CD, Raghavan P, Schütze H (2008) Introduction to Information Retrieval. Cambridge University Press, USA

Meyer HJ, Grunert H, Waizenegger T, Woltmann L, Hartmann C, Lehner W, Esmailoghli M, Redyuk S, Martinez R, Abedjan Z, et al. (2019) Particulate matter matters-the data science challenge@ btw 2019. Datenbank-Spektrum pp 1–18

Nargesian F, Pu KQ, Zhu E, Bashardoost BG, Miller RJ (2018a) Optimizing organizations for navigating data lakes. CoRR abs/1812.07024, 1812.07024

Nargesian F, Zhu E, Pu KQ, Miller RJ (2018b) Table union search on open data. Proceedings of the VLDB Endowment (PVLDB) 11(7):813–825, DOI https://doi.org/10.14778/3192965.3192973, URL http://www.vldb.org/pvldb/vol11/p813-nargesian.pdf

Nargesian F, Pu KQ, Zhu E, Bashardoost BG, Miller RJ (2020) Organizing data lakes for navigation. In: Maier D, Pottinger R, Doan A, Tan W, Alawini A, Ngo HQ (eds) Proceedings of the International Conference on Management of Data (SIGMOD), ACM, pp 1939–1950, URL https://doi.org/10.1145/3318464.3380605

Nargesian F, Pu KQ, Bashardoost BG, Zhu E, Miller RJ (2023) Data lake organization. IEEE Trans Knowl Data Eng 35(1):237–250

Ouellette P, Sciortino A, Nargesian F, Bashardoost BG, Zhu E, Pu K, Miller RJ (2021) RONIN: data lake exploration. Proceedings of the VLDB Endowment (PVLDB) 14(12):2863–2866, URL http://www.vldb.org/pvldb/vol14/p2863-nargesian.pdf

Ribeiro LA, Härder T (2011) Generalizing prefix filtering to improve set similarity joins. Inf Syst 36(1):62–78, URL https://doi.org/10.1016/j.is.2010.07.003

Santana DR, Ribeiro LA (2023) Approximate similarity joins over dense vector embeddings. In: Proceedings of the Brazilian Symposium on Databases, SBBD 2023, Belo Horizonte, MG, Brazil, September 25-29, 2023, SBC, pp 51–62, URL https://doi.org/10.5753/sbbd.2023.232512

Santos ASR, Bessa A, Chirigati F, Musco C, Freire J (2021) Correlation sketches for approximate join-correlation queries. In: Li G, Li Z, Idreos S, Srivastava D (eds) Proceedings of the International Conference on Management of Data (SIGMOD), ACM, pp 1531–1544, URL https://doi.org/10.1145/3448016.3458456

Santos ASR, Bessa A, Musco C, Freire J (2022) A sketch-based index for correlated dataset search. In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE, pp 2928–2941, URL https://doi.org/10.1109/ICDE53745.2022.00264

Sarma AD, Fang L, Gupta N, Halevy AY, Lee H, Wu F, Xin R, Yu C (2012) Finding related tables. In: Candan KS, Chen Y, Snodgrass RT, Gravano L, Fuxman A (eds) Proceedings of the International Conference on Management of Data (SIGMOD), ACM, pp 817–828, URL https://doi.org/10.1145/2213836.2213962

Sarma AD, He Y, Chaudhuri S (2014) Clusterjoin: A similarity joins framework using map-reduce. Proceedings of the VLDB Endowment (PVLDB) 7(12):1059–1070

Shrivastava A (2017) Optimal densification for fast and accurate minwise hashing. In: Precup D, Teh YW (eds) Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017, PMLR, Proceedings of Machine Learning Research, vol 70, pp 3154–3163, URL http://proceedings.mlr.press/v70/shrivastava17a.html

Singh K, Paneri K, Pandey A, Gupta G, Sharma G, Agarwal P, Shroff G (2016) Visual bayesian fusion to navigate a data lake. In: FUSION, IEEE, pp 987–994

Suri S, Ilyas IF, Ré C, Rekatsinas T (2021a) Ember: No-code context enrichment via similarity-based keyless joins. Proceedings of the VLDB Endowment (PVLDB) 15(3):699–712, DOI https://doi.org/10.14778/3494124.3494149, URL http://www.vldb.org/pvldb/vol15/p699-suri.pdf

Suri S, Ilyas IF, Ré C, Rekatsinas T (2021b) Ember: no-code context enrichment via similarity-based keyless joins. Proceedings of the VLDB Endowment (PVLDB) 15(3):699–712, URL https://doi.org/10.14778/3494124.3494149

Venetis P, Sismanis Y, Reinwald B (2012) *CRSI*: a compact randomized similarity index for set-valued features. In: Rundensteiner EA, Markl V, Manolescu I, Amer-Yahia S, Naumann F, Ari I (eds) Proceedings of the International Conference on Extending Database Technology (EDBT), ACM, pp 384–395, URL https://doi.org/10.1145/2247596.2247642

Vernica R, Carey MJ, Li C (2010) Efficient parallel set-similarity joins using mapreduce. In: SIGMOD, ACM, pp 495–506

Walker J, Koutsiana E, Massey J, Thuermer G, Simperl E (2023) Prompting datasets: Data discovery with conversational agents. CoRR abs/2312.09947, URL https://doi.org/10.48550/arXiv.2312.09947

Wang J, Li G, Feng J (2012) Can we beat the prefix filtering?: an adaptive framework for similarity join and search. In: Candan KS, Chen Y, Snodgrass RT, Gravano L, Fuxman A (eds) Proceedings of the International Conference on Management of Data (SIGMOD), ACM, pp 85–96, URL https://doi.org/10.1145/2213836.2213847

Wang X, Qin L, Lin X, Zhang Y, Chang L (2017) Leveraging set relations in exact set similarity join. Proceedings of the VLDB Endowment (PVLDB) 10(9):925–936

Wang X, Qin L, Lin X, Zhang Y, Chang L (2019) Leveraging set relations in exact and dynamic set similarity join. VLDB Journal 28(2):267–292

Xiao C, Wang W, Lin X, Shang H (2009) Top-k set similarity joins. In: Ioannidis YE, Lee DL, Ng RT (eds) Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China, IEEE Computer Society, pp 916–927. URL https://doi.org/10.1109/ICDE.2009.111

Xiao C, Wang W, Lin X, Yu JX, Wang G (2011) Efficient similarity joins for near-duplicate detection. ACM Trans Database Syst 36(3):15:1–15:41, URL https://doi.org/10.1145/2000824.2000825

Yakout M, Ganjam K, Chakrabarti K, Chaudhuri S (2012) Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In: Candan KS, Chen Y, Snodgrass RT, Gravano L, Fuxman A (eds) Proceedings of the International Conference on Management of Data (SIGMOD), ACM, pp 97–108, URL https://doi.org/10.1145/2213836.2213848

Zhang M, Chakrabarti K (2013) Infogather+: semantic matching and annotation of numeric and time-varying attributes in web tables. In: Ross KA, Srivastava D, Papadias D (eds) Proceedings of the International Conference on Management of Data (SIGMOD), ACM, pp 145–156, URL https://doi.org/10.1145/2463676.2465276

Zhang Y, Ives ZG (2019) Juneau: Data lake management for jupyter. Proceedings of the VLDB Endowment (PVLDB) 12(12):1902–1905, DOI https://doi.org/10.14778/3352063.3352095, URL http://www.vldb.org/pvldb/vol12/p1902-zhang.pdf

Zhang Y, Ives ZG (2020) Finding related tables in data lakes for interactive data science. In: Maier D, Pottinger R, Doan A, Tan W, Alawini A, Ngo HQ (eds) Proceedings of the International Conference on Management of Data (SIGMOD), ACM, pp 1951–1966, URL https://doi.org/10.1145/3318464.3389726

Zhu E, Nargesian F, Pu KQ, Miller RJ (2016) LSH ensemble: Internet-scale domain search. Proceedings of the VLDB Endowment (PVLDB) 9(12):1185–1196, DOI https://doi.org/10.14778/2994509.2994534, URL http://www.vldb.org/pvldb/vol9/p1185-zhu.pdf

Zhu E, Deng D, Nargesian F, Miller RJ (2019) JOSIE: overlap set similarity search for finding joinable tables in data lakes. In: Boncz PA, Manegold S, Ailamaki A, Deshpande A, Kraska T (eds) Proceedings of the International Conference on Management of Data (SIGMOD), ACM, pp 847–864, URL https://doi.org/10.1145/3299869.3300065

Chapter 5
Data Discovery Systems



Abstract This chapter examines data discovery systems from an end-to-end systems perspective. Rather than focusing on individual discovery operators in isolation, it studies how complete systems integrate metadata extraction, indexing, query processing, ranking, interaction, and governance support to help users find, interpret, and reuse data. The chapter is motivated by user studies showing that data discovery in practice remains highly manual, iterative, and composition-oriented: Users often rely on scripting, repeated searches, and ad hoc combinations of tools to identify relevant and joinable datasets for downstream tasks. These findings motivate the need for discovery systems that go beyond keyword search and support richer workflows involving refinement, composition, and assessment of fitness for use. The chapter surveys both academic and industrial systems. On the academic side, it reviews representative end-to-end prototypes with distinct architectural choices. On the industrial side, it examines enterprise discovery layers and data market places. Across these systems, the chapter compares design decisions along dimensions such as indexing substrate, query interface, optimization strategy, extensibility, metadata maintenance, and support for downstream integration tasks. The main conclusion is that data discovery is fundamentally a systems problem. Effective solutions must combine expressive discovery abstractions with scalable metadata management, interaction support, and operational integration into real data platforms. At the same time, the chapter shows that current systems still reflect a divide between metadata-centric enterprise catalogs and research prototypes for relational discovery. Bridging this divide emerges as a key direction for future work on end-to-end data discovery systems.

Keywords Discovery systems · Lake management · Data catalogs · Blend · Aurum · Auctus

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027

Z. Abedjan et al., *Data Discovery in Data Lakes*,
https://doi.org/10.1007/978-3-032-30822-1_5

5.1 Introduction

This chapter takes a systems perspective on data discovery. Rather than surveying isolated algorithms and operations, we focus on complete, working prototypes that integrate multiple discovery functions into coherent systems for end users. We cover both academic prototypes and industrial systems, with an emphasis on end-to-end designs that operationalize discovery across ingestion, metadata capture, indexing, interaction, and governance constraints. The academic systems illustrate how novel ideas, such as new forms of semantic enrichment, interactive exploration, automated summarization, lineage-aware navigation, or workload-driven recommendations, can be realized in a functional pipeline. Industrial systems, in contrast, reveal which architectural and product choices survive contact with enterprise realities such as scale, heterogeneous toolchains, access control, compliance, and organizational adoption.

The following sections examine these questions from both the demand and the supply side. We begin by discussing evidence from user studies that show why current discovery workflows call for end-to-end systems rather than ad hoc combinations of scripts, SQL queries, and manual inspection. We then survey academic systems that explicitly pursue such an integrated design, highlighting their architectural choices and trade-offs. Finally, we contrast these research efforts with industrial discovery tools, whose cataloging, governance, and marketplace capabilities address related needs but often emphasize different objectives than relational discovery and downstream data integration.

5.2 The Demand for End-to-End Discovery Systems

In the following, we summarize the findings of two independent user studies that report how data practitioners engage with data discovery.

5.2.1 Survey by Esmailoghli et al. (2023)

Esmailoghli et al. (2025) conducted a user study in December 2023 and report responses from 45 participants at a 40% response rate (18 responses), spanning data practitioners from both research and industry settings, including health care, banking, technology, and universities. The questionnaire covered participant background, data discovery preferences and needs, and implementation preferences for simple and complex discovery tasks.

A central result is that data discovery is a recurring practical need, but current practice remains highly manual. The study reports that 77.8% of participants solve discovery problems through manual scripting and visual inspection, while 50% use SQL queries and database metadata. Manual work is especially pronounced among

research participants, where it reaches 100%, whereas industry participants rely somewhat more on SQL. Combined with the finding that 94.4% of participants know Python, this suggests that discovery today is often implemented as ad hoc procedural work rather than as a dedicated systems capability.

The study also indicates that discovery is rarely resolved in one shot. Most participants reported that they more often need multiple searches than a single search to find the required dataset. This is an important architectural signal: Discovery systems should not be optimized purely for one-off retrieval, but for iterative workflows in which users progressively refine hypotheses, inspect intermediate results, and reformulate queries.

Another key finding is that downstream tasks frequently require more than one table. Specifically, 66.6% of participants reported that their downstream applications are typically satisfied by *both* a single table and a composition of tables, and research participants show a stronger preference for compositions of tables than industry participants. This directly supports the need for end-to-end discovery systems that move beyond isolated table search toward composition-aware discovery, where retrieval, joinability analysis, and integration planning are first-class capabilities.

The storage and execution environment reported by the participants further reinforces this point. The study states that 77.8% of participants manage their data lake at least partly through traditional Database Management System (DBMS) technology, and 38.9% rely exclusively on DBMSs. Moreover, all participants answered that they would use a DBMS if it offered data discovery capabilities such as indexing, optimization, and discovery operations. Users do not merely want a separate search frontend but are receptive to discovery functionality embedded into the execution engine itself.

The open-text responses on desired support are especially revealing for end-to-end system design. Frequently mentioned needs include better indexing, discovery of joinable tables, and a user-friendly interface. These requests line up with a systems agenda in which indexing structures, JOIN discovery operators, and usable abstractions are not optional extras but core requirements. In other words, users are asking for discovery systems that couple physical support mechanisms with high-level workflow support.

Perhaps the strongest argument for end-to-end discovery systems comes from the reported task types. Participants commonly selected *complex* discovery needs rather than only simple keyword lookup. The two most common needs were finding tables containing a set of rows and finding tables with a correlating column for a target attribute, especially in industry. The study also notes that these complex needs are consistent with the earlier finding that many users require table compositions. This matters because end-to-end discovery systems must support semantic constraints, joinability, and utility for downstream analytics or machine learning in one integrated pipeline, rather than treating these as disconnected tasks.

The implementation-preference portion of the study strengthens that conclusion. Users were presented with two alternatives scripts to perform a complex discovery task, combining JOIN search with data imputation and UNION search. The overwhelming majority in both participant groups preferred the implementation API

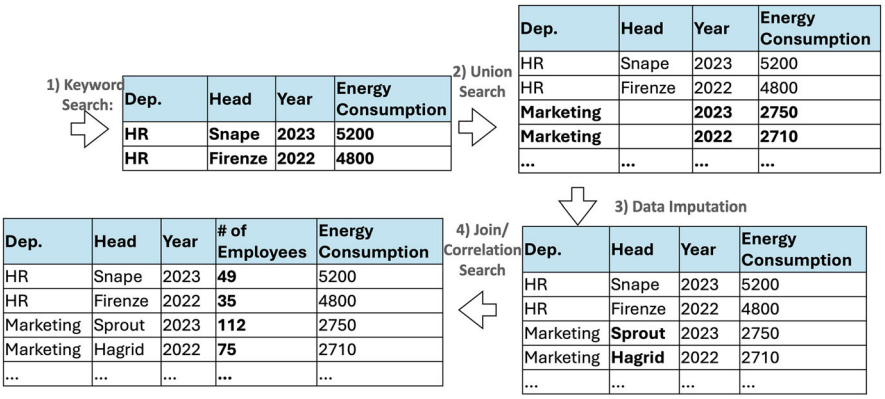


Fig. 5.1 Users were presented with the following discovery task: Find tables that contain the following keywords: "Bayern Munich," "Eintracht Frankfurt," "FC Köln," and "Hertha Berlin" but do not contain these keywords: "Barcelona," "Paris Saint Germain," "Arsenal," and "Juventus"

provided by the data disocovery prototype BLEND (Esmailoghli et al. 2025) over a native Python implementation as shown in Fig. 5.1. For a simpler keyword-based task, BLEND was still the most favored option at 44.4%, followed closely by SQL at 38.9%, while native Python was preferred by only 16.7%. These results suggest a clear pattern: Users may tolerate conventional solutions such as SQL for relatively simple discovery, but for complex multistep discovery they prefer a specialized abstraction that captures the workflow end to end.

The study supports several concrete arguments for end-to-end discovery systems. First, the prevalence of manual scripting shows that users currently bridge gaps between discovery, filtering, JOIN analysis, and downstream use themselves. An end-to-end system should absorb this orchestration burden. Second, the fact that multiple searches are commonly needed implies that discovery must support iterative refinement and not just ranked retrieval. Third, because the dominant downstream requirement is often a *composition* of tables, the system should expose operators for joinable-table discovery, composition planning, and integration-aware ranking. Finally, because users explicitly ask for indexing and JOIN discovery while also expressing universal willingness to use DBMS-backed discovery capabilities, the right abstraction is not an isolated catalog but a system that integrates logical discovery operators with physical support such as indexes and optimizations.

5.2.2 *Survey by Hulsebos et al. (2024)*

A second user study conducted by Hulsebos et al. (2024) arrives at a conclusion that closely mirrors the study by Esmailoghli et al. (2025): Practical dataset discovery is still dominated by multistep, human-driven workflows rather than by seamless support from dedicated systems. Based on a survey of data professionals from a range of roles and domains, the work shows that finding useful structured data

remains time-consuming, iterative, and difficult to express through current search interfaces. As in the previous study, the core message is that discovery is not well served by isolated keyword search or lightweight metadata lookup. Instead, it emerges as a broader systems problem that spans search, interpretation, refinement, and eventual integration.

Hulsebos et al. (2024) argue that discovery is not a narrow one-shot retrieval task. Users often search for an initial dataset that is only partially specified at the beginning, and they refine their understanding of what they need as they inspect candidates. This reinforces the same pattern observed earlier: Users rarely solve discovery with a single query, and a practical system must therefore support iterative reformulation rather than assuming that intent is fully known up-front. Their study also aligns with the previous one in showing that current practice depends heavily on general-purpose tools and manual work-arounds. Rather than relying on specialized discovery platforms, users search through database interfaces, SQL, repository portals, Web search, and conversations with colleagues. This matches the earlier observation that discovery is frequently implemented through ad hoc scripting, SQL, and manual inspection. Taken together, the two studies suggest that users are compensating for missing end-to-end discovery functionality by stitching together existing tools themselves.

Another strong point of overlap is that discovery depends on much richer criteria than simple topic matching. In this study, users care not only about the semantic content of a dataset but also about its granularity, dimensions, freshness, update frequency, schema, and other operational characteristics. This complements the previous study's finding that users often need tables that are not merely relevant in isolation, but suitable for downstream composition and analysis. In both cases, the implication is the same: Discovery systems must reason jointly about semantics, metadata, and downstream utility.

A further commonality is the prominence of usability and interaction issues. This study describes discovery as an iterative and collaborative activity involving reformulation, browsing, filtering, and consultation with domain experts. That directly parallels the previous study's evidence that users often need multiple searches and prefer specialized support especially for more complex discovery tasks. In both studies, users are effectively asking for systems that guide them through discovery rather than simply returning ranked lists.

The reported pain points by Hulsebos et al. (2024) fit the same overall pattern. Users struggle with inconsistent metadata, ambiguous schemas, unclear granularity, excessive result sets, and limited query expressiveness. These issues are consistent with the previous study's finding that users want better indexing, joinable-table discovery, and more user-friendly interfaces. Across both studies, the shortcomings of current practice are not confined to one layer. They span physical support, query abstractions, ranking, and result comprehension, which is exactly why the case for end-to-end discovery systems is so strong. Taken together, the user study argues that practical data discovery is not primarily a metadata lookup problem. It is an end-to-end systems problem involving iterative search, joinability reasoning, table composition, feature-oriented relevance, and tight integration with execution

infrastructure. The BLEND results therefore support the design of discovery systems that unify discovery operators, optimization support, and user-facing abstractions in a single framework rather than leaving users to assemble these steps manually in Python or scattered tools.

5.3 Academic Systems

End-to-end data discovery systems have not been so far the main focus of the research community in the realm of data discovery. Few attempts exist that not only consider algorithmic details of individual tasks or operations but also aim at a novel assembly of end-to-end components as depicted in Fig. 5.2. Among those attempts are the prototypes Aurum (Fernandez et al. 2018), BLEND (Esmailoghli et al. 2025), and Auctus (Castelo et al. 2021), which we survey next.

5.3.1 AURUM

AURUM was among the very first systems that positioned itself as an end-to-end solution for data discovery in data lakes (Fernandez et al. 2018). In close exchange with industrial partners, the researchers identified prominent challenges that arose within their organizational ecosystems: Data is being produced frequently, but its curation and organization is often neglected due to different factors, such as business priorities and staff fluctuation. The latter poses a significant problem as new staff that joins an organization might not effectively find existing relevant datasets prompting them to create redundant datasets that again are added to the heap of existing datasets. To mitigate these problems, the authors pursued a systemic approach to serve end users. Instead of trying to develop algorithmic solutions for well-defined research problems related to individual discovery operations, they

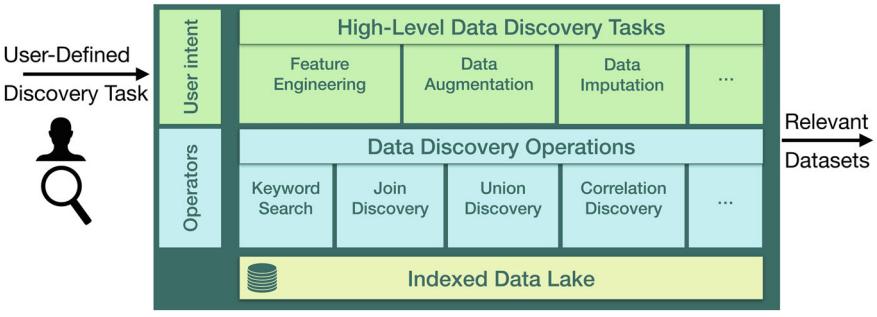


Fig. 5.2 Vanilla data discovery system

surveyed users and collected requirements (Fernandez 2018). Combining state-of-the-art in information retrieval and information theory, AURUM emerged as a discovery system with the core component of the enterprise knowledge graph (EKG). As it was not a priori clear what types of relationships are of interest to a specific user, the solution should be adaptable to custom table relationships. The idea was that inter-tabular relationships can be modeled as edges between columns and tables, thus introducing hyperedges between tables. While the EKG scoped the semantics of possible discovery operations, AURUM also covered system considerations regarding ease of use in interacting with the EKG through the so-called *source retrieval query language (SRQL)* and maintaining it via heuristic-based incremental index updates.

Building the EKG
The EKG build process is designed to minimize costly passes over underlying data sources by decomposing construction into a two-step pipeline. First, a profiler produces compact Minhash (Broder 1997) signatures that summarize the source data and stores them as profiles, which become the graph's nodes. The profiler uses data and task parallelism for fast sweeping through large lakes. Second, a graph builder derives relationships between columns using only these signatures, avoiding additional data scans and replacing expensive all-pairs comparisons with an approximate nearest-neighbor formulation that enables near-linear construction time. The index is based on *ElasticSearch*,[1] which treats data objects as documents. A reimplmentation by the authors of LakeBench (Deng et al. 2024) uses an HNSW Index (Malkov and Yashunin 2020) for this step—for more details, see Sect. 4.8.

Maintaining the EKG
To keep the EKG up-to-date as sources evolve, AURUM emphasizes lightweight maintenance rather than full recomputation. When data changes, the system uses its so-called *Resource Efficient Signature Sampling (RESS)* to determine which sources have materially changed by reading only samples of the data. It then selectively re-indexes the affected sources, recomputes their relationships, and updates the EKG accordingly. This approach limits disruption and reduces the operational burden of keeping discovery metadata fresh.

Querying the EKG
For querying, AURUM exposes the EKG through a set of composable discovery primitives expressed as property and relation constraints. While these operations could be mapped onto general-purpose graph query languages, AURUM introduces the dedicated language SRQL to meet discovery-specific requirements. In particular, the execution engine is built to support ranked results, provenance access, debugging, and interactive "human-latency" path exploration. A core concept in SRQL is the Discovery Result Set (DRS) abstraction to carry provenance and intermediate

[1] https://www.elastic.co/docs/reference/elasticsearch

results. A space-efficient in-memory graph index, coined *G-Index*, is used for fast discovery of column relationships.

The core novel idea behind AURUM is the representation of the lake as a hypergraph where table relationships can be encoded as specific edge types. However this has to happen at design time. A clear limitation of the framework is the requirement that queries for JOIN and UNION discovery first require the input table to be part of the EKG. Thus every new table has to undergo the indexing process for finding related tables in a full scan manner. Recently, there have been several extensions on top of ARUM, such as Nexus (Gong et al. 2024), METAM (Galhotra et al. 2023), and Ver (Gong et al. 2023). Nexus extends the discovery setting to correlation analysis over collections of spatiotemporal tables by building offline discovery indices and identifying variables that can be joined and correlated efficiently. METAM shifts the problem from generic related table discovery to goal-oriented discovery, using feedback from a downstream task to iteratively steer the search toward datasets that maximize augmentation utility. Ver focuses on view discovery in pathless table collections, identifying PROJECT-JOIN views even when JOIN paths are missing and user queries are noisy or inaccurate.

5.3.2 AUCTUS

AUCTUS is an open-source dataset search engine designed by researchers at NYU for both *data discovery* and *data augmentation* (Castelo et al. 2021). Its architecture is organized as a modular pipeline with five main stages: ingestion, profiling and summary construction, indexing, querying and ranking, and augmentation.

Data Ingestion and Profiling

AUCTUS uses source-specific plugins to retrieve datasets from external repositories through their APIs, which makes the platform extensible to many different data sources. The work mentions support for sources such as Socrata,[2] Zenodo,[3] and World Bank Open Data,[4] and the system also allows users to upload their own datasets directly.

After ingestion, datasets are processed by a *profiling* component. This profiler extracts metadata from the actual dataset contents rather than relying only on publisher-provided descriptions. In particular, it detects whether columns are categorical, numerical, spatial, or temporal, computes type-dependent statistics, and builds compact summaries of dataset contents. These summaries are central to the architecture because they support both result presentation and efficient query processing.

[2] https://dev.socrata.com/

[3] https://zenodo.org/

[4] https://data360.worldbank.org

Instead of storing full datasets in the core search engine, AUCTUS stores *dataset summaries* together with provenance metadata. The summaries are concise representations of attribute contents, typically expressed as ranges for categorical, numerical, spatial, and temporal attributes. To estimate whether two attributes are likely to be joinable, AUCTUS refines these summaries using *k-means*-based clustering so that intersections can be estimated more accurately. Provenance information is kept, so the original datasets can later be retrieved for download or augmentation, and datasets may also be cached for efficiency.

The next stage is *indexing*. AUCTUS uses different indexing mechanisms depending on attribute type. Metadata and summaries are indexed in *Elasticsearch*, as is the case for AURUM. Numerical and temporal summaries use range types, while spatial summaries use geo-shape types. For categorical attributes, AUCTUS uses *Lazo* (Fernandez et al. 2019), a MinHash/LSH-based set-overlap index, to efficiently identify columns with overlapping value sets. This type-aware indexing strategy allows the system to support richer queries than keyword-only dataset search engines.

Query Processing and Result Ranking

On top of these indices, AUCTUS provides a *querying and ranking* layer. Users can issue discovery queries that combine keyword, temporal, spatial, data-type, and source constraints. The system also supports *data integration queries*, where a user provides an input dataset and asks AUCTUS to find datasets that are *joinable* or *unionable* with it. For JOIN search, AUCTUS compares each attribute of the input dataset against the corresponding type-specific index and treats datasets with intersecting attribute summaries as candidates. For UNION search, it looks for datasets with compatible attribute types and similar attribute names, using fuzzy matching in Elasticsearch. Candidate results from the different indices are merged and ranked: JOIN results are ranked by estimated summary intersection, while UNION results are ranked by similarity between attribute names.

Finally, AUCTUS includes an *augmentation* layer that performs the actual data integration operation. Once a user selects a returned dataset, the system uses its stored provenance to materialize the original data and execute the requested JOIN or UNION with the user's input dataset. If multiple candidate JOIN keys exist, the user can choose which attribute pairs to use. For spatial and temporal JOINs, AUCTUS first translates attributes to a common resolution before performing the merge.

From a deployment perspective, AUCTUS is built for scalability. The system is containerized with Docker, each ingestion plugin can run as an independent container, and profiling and query services can be replicated as load increases. The platform is accessible through both a Web user interface and programmatic Python and REST APIs.

5.3.3 BLEND

BLEND (Esmailoghli et al. 2025) is designed as an end-to-end data discovery system that supports complex discovery tasks over large collections of tables by combining (i) a relational indexing scheme implemented inside a DBMS, (ii) a compositional plan-based query interface for expressing multistep discovery pipelines, and (iii) a system-level optimization layer that reduces execution cost by exploiting intermediate results and reordering work.

BLEND's Indexing
BLEND's indexing strategy centers on a unified relational layout that serializes multiple discovery-oriented indexes into a single table, `AllTables`. The idea of using a relational backend is motivated by the indexing and tracing opportunities that are incumbent to native DBMS and can be leveraged for implementing the discovery operations. The table stores the core inverted index fields `CellValue`, `TableId`, `ColumnId`, and `RowId`, so that content-to-table lookups can be expressed directly as relational queries. To accelerate lookup and loading, BLEND builds in-database indexes on `CellValue` and `TableId`. The same layout also materializes a row-level "super key" derived from XASH (Esmailoghli et al. 2022), which aggregates row values into a compact hash to support efficient multicolumn match discovery. For correlation discovery, BLEND adapts QCR-style correlation estimation (Santos et al. 2022) to an in-database formulation by adding a `Quadrant` attribute per cell that captures whether a numeric value lies above or below its column average, enabling correlation estimation to be executed without exporting data to the application layer.

Query Formulation in BLEND
BLEND exposes discovery as a query-plan construction interface in which users create a `Plan` and compose operators that are compiled into a DAG. Operators are divided into *seekers* and *combiners*. Seekers implement primitive retrieval tasks and return the top-k candidate tables for a query input, including keyword search, single-column overlap for JOIN/UNION signals, multicolumn overlap for composite JOIN signals, and correlation search. Combiners implement set-style composition over seeker outputs, allowing users to build multistage discovery pipelines that express tasks such as UNION discovery, enrichment, or imputation by combining intermediate candidate sets through operations like UNION, intersection, difference, and counting/aggregation. BLEND's modular API enables the formulation of a wide range of discovery tasks, including prefetching tables for question answering (Malaguti et al. 2025).

Discovery Pipeline Optimization
BLEND includes an explicit optimization layer that acts before the DBMS optimizer by revising discovery plans and rewriting the SQL generated by seekers. First, BLEND predicts and ranks seeker runtimes using lightweight regression models trained offline and then prioritizes faster seekers when multiple seekers feed a common downstream combiner. Second, BLEND performs combiner-aware query rewriting that injects predicates derived from intermediate results to reduce

Table 5.1 Abstract comparison of AURUM, AUCTUS, and BLEND

System	Indexing	Query language/interface	Optimization	Extensibility
AURUM	Graph-centric index; Minhash signatures; document retrieval index	Discovery language over graph primitives	no inter-task optimization	Extending requires adding new signals/edges to the graph; new data must be indexed to participate fully
AUCTUS	Summary-centric document retrieval index; Minhash signatures; topical clustering	Constraint-based search and JOIN/UNION search	no inter-task optimization	Plugin-based ingestion makes new repositories easy to add
BLEND	Relational index structure	API for formulating discovery plans	Operator reordering and SQL rewriting	New discovery primitives can be added provided they compile to relational queries

subsequent search space, e.g., restricting candidate `TableId`s for intersections or excluding them for differences. For count-based aggregation, BLEND rewrites queries to group by `TableId` and order by frequency to surface strong candidates. Together, operator reordering and SQL rewriting leverage intermediate results to prune computation early while still executing the resulting relational workload inside the DBMS.

Table 5.1 contrasts the three described end-to-end discovery systems through four system-level lenses, indexing, query language, optimization, and extensibility. AURUM and AUCTUS both rely on signature structures for capturing similarities of table elements. AURUM uses a graph-based approach to pre-calculate the relationships, while AUCTUS uses clustering to organize similar data elements. BLEND, in contrast, has a DBMS backend: It encodes discovery signals in relational structures, exposes discovery as composable operator plans, and relies on plan reordering and SQL rewriting to keep multistage pipelines efficient while executing inside the database. AURUM and BLEND enable extensibility of operators through definition of new edge types or seekers, respectively. AUCTUS rather focuses on flexible repository ingestion.

5.4 Industrial Discovery Tools

Data lake management and data discovery have received considerable attention from major software vendors, although the resulting solutions differ substantially in their underlying objectives. One line of development has focused on the provision,

hosting, and cataloging of datasets for exploratory use. Such platforms typically prioritize discoverability, metadata enrichment, access control, and governance mechanisms that help users identify and interpret relevant data assets within large and heterogeneous repositories. These capabilities are not only valuable for exploration in their own right but also establish important preconditions for subsequent forms of value creation, since data must first be locatable, interpretable, and sufficiently governed before it can be shared commercially or processed reliably at scale. A second line of development concerns data marketplaces (Fernandez 2023), which represent a more explicit commercial exploitation of raw and processed datasets. In this context, vendors provide mechanisms for data publishing, exchange, licensing, and monetization, positioning data as a tradable economic good rather than merely an operational resource. A third line of development has concentrated on scalable processing over data lakes, often by introducing data warehouse-like capabilities for otherwise weakly integrated environments. These systems typically add layers for structured querying, optimization, schema enforcement, and transactional consistency, thereby attempting to reconcile the flexibility of data lakes with the reliability and performance traditionally associated with data warehouses. These different trajectories indicate that the software industry has not treated the data lake as a single problem domain, but rather as a foundation for distinct technical and business-oriented solution spaces.

5.4.1 Enterprise Discovery Platforms

Enterprise data discovery arises from the need to help users find, interpret, and safely reuse datasets across large, fragmented data estates. In practice, organizations need systems that can surface what datasets exist, who owns them, how they are structured, how they relate to other assets, and whether they are appropriate for analytics or machine learning tasks. This makes dataset cataloging a prime use case for data discovery. Furthermore, market place platforms naturally resemble systems that organize and curate lake-scale repositories.

In this book we discuss two representative catalog systems, GOODS and Unity Catalog. GOODS is Google's internal dataset catalog, built to organize datasets across heterogeneous storage systems by collecting and inferring metadata such as schema, provenance, ownership, and similarity. Unity Catalog plays a comparable role in the Databricks ecosystem, providing a centralized governance and catalog layer for data and AI assets. Both systems therefore illustrate enterprise solutions that support dataset cataloging as a core enabler of data discovery. Finally, we briefly dive into the world of data market places to assess their connection to data discovery in data lakes.

5.4.1.1 GOODS

GOODS (Halevy et al. 2016) is an enterprise dataset discovery and management layer for a very large, heterogeneous, and fast-changing internal data ecosystem. Its central use case is to help engineers find datasets, understand their contents and context, monitor them for regressions, and trace dependencies across teams without requiring dataset producers to migrate to a single storage platform or manually register metadata at creation time.

From the consumer side, GOODS acts as a search engine over the company's datasets, augmented with faceted navigation and dataset profile pages. These profile pages expose schema information, access hints, provenance links, and similarity signals that help users understand unfamiliar data and identify candidate datasets for reuse or JOINs. From the producer side, GOODS supports dashboards, alerting, and collaborative annotation, so dataset owners can describe datasets, surface operational issues, and communicate caveats to downstream users. The overall purpose is therefore not merely cataloging, but enabling operational data understanding and reuse at organizational scale.

Architecturally, GOODS is a metadata-centric system built on post hoc metadata collection and inference. A catalog stores multiple metadata families, including basic properties such as size and format, content-derived metadata such as schema and fingerprints, provenance links to reading and writing jobs, user-supplied annotations, team and project information, and temporal change history. This design allows GOODS to unify information from many storage backends while preserving a common logical view of datasets.

A key architectural feature is that much of the metadata is inferred rather than directly declared. GOODS derives schema associations, candidate keys, content summaries, and semantic hints by analyzing dataset contents, consulting usage logs, and leveraging other infrastructure signals. It also incorporates owner and project information and accepts additional metadata contributed by users or other teams.

To manage scale and redundancy, GOODS organizes physical datasets into logical clusters. The paper notes that many datasets are repeated versions, replicas, or shards, so clustering allows the system to present a cleaner abstraction to users and reduce metadata extraction cost by propagating metadata across related members of a cluster. Importantly, clustering is designed to be cheap, relying largely on path-level regularities such as timestamps and version identifiers rather than expensive content analysis. This supports both usability and computational efficiency in a catalog that spans tens of billions of datasets.

The backend catalog is implemented on Bigtable, with rows representing datasets or dataset clusters. In addition to dataset metadata, each row stores status metadata recording which modules processed the entry, when they ran, and whether they succeeded. GOODS uses this status layer to coordinate a large set of batch crawlers and analyzers that run independently, often with inter-module dependencies. For example, a fingerprinting module may wait until a schema analyzer has successfully processed a row, and freshness windows determine when modules need to recompute metadata. This row-level coordination lets the system remain modular,

extensible, and resilient even when some jobs are slow, temporarily broken, or running concurrently.

Operationally, the architecture emphasizes robustness. The system also includes aggressive garbage collection because the dataset universe has extremely high churn, with massive daily creation and deletion rates. Since metadata extraction and provenance maintenance happen concurrently, GOODS prevents reintroducing partially deleted entries, so life cycle management is treated as a first-class backend concern rather than an afterthought.

In summary, the use case of GOODS is to provide enterprise-scale dataset discovery, comprehension, monitoring, and governance across a fragmented internal data landscape, while its architecture is a large-scale metadata platform that combines heterogeneous source ingestion, metadata inference, logical clustering, Bigtable-backed catalog storage, status-driven batch processing, and user-facing services such as search, profiles, dashboards, and provenance visualization.

5.4.1.2 Unity Catalog

Unity Catalog (Chandra et al. 2025) is Databricks' unified governance layer for the lakehouse (Zaharia et al. 2021). Its primary use case is to centralize the management of data and AI assets so that organizations can discover assets, control access consistently, and trace how those assets are used across analytics and machine learning workflows. In practice, it is meant to replace fragmented, workspace-local governance with a shared control plane for tables, views, volumes, models, functions, and related metadata, while also exposing lineage and auditability as first-class capabilities.

From the user's perspective, Unity Catalog solves a governance and interoperability problem rather than a pure storage problem. Architecturally, the top-level container in Unity Catalog is the metastore. Databricks recommends using a single metastore per cloud region, with catalogs registered inside that metastore and schemas nested underneath catalogs. The metastore stores the governed assets together with the permissions that regulate access to them. This creates a centralized metadata and authorization layer that can be shared across workspaces in the same region, which is why Unity Catalog is best understood as the governance backbone of the Databricks platform.

A central architectural distinction in Unity Catalog is between managed and external data objects. Managed tables are fully governed and physically managed by Unity Catalog in Unity Catalog-managed storage, and Databricks notes that they always use Delta Lake format. External tables and external volumes, by contrast, are registered in Unity Catalog for access control and auditing, but their storage life cycle and layout remain under the customer's cloud storage or other platforms. To make this work, Unity Catalog introduces storage credentials and external locations as securable objects: The storage credential abstracts the cloud credential, while the external location binds that credential to a specific storage path.

This means the system architecture separates governance metadata from the physical data plane. Unity Catalog governs access, lineage, and discovery, while the underlying data may live either in Unity Catalog-managed storage or in customer-managed cloud object storage referenced through external locations. The same pattern extends beyond tabular data: Unity Catalog also governs unstructured files through volumes and increasingly covers AI artifacts such as models and features, so its architectural role is broader than a traditional metastore.

Another important architectural capability is lineage and federation. Unity Catalog includes lineage features that let users inspect how data moves through transformations, which supports impact analysis, debugging, and governance review. Databricks also positions Unity Catalog as the central governance layer for a broader lakehouse platform, including the ability to integrate external databases into its governance model through Lakehouse Federation. In that sense, Unity Catalog is not just a catalog of Databricks-native objects, but it is intended to be the policy and metadata plane spanning native lakehouse assets and selected external systems.

In summary, the use case of Unity Catalog is to provide centralized governance, discovery, access control, and lineage for data and AI assets across the Databricks lakehouse, while its system architecture is a hierarchical metadata and authorization layer organized around metastores, catalogs, schemas, and securable objects, with a clean separation between governance control and physical storage through managed storage, storage credentials, and external locations.

Although systems such as GOODS and Unity Catalog provide strong foundations for enterprise data discovery by cataloging datasets, metadata, lineage, and ownership information, their discovery functionality is still largely oriented toward lookup, governance, and browsing rather than deeper relational discovery operations. In particular, they are not primarily designed to support search tasks such as finding datasets that can be unioned with a target dataset, identifying joinable tables based on key overlap or semantic compatibility, or ranking candidate datasets according to their usefulness for downstream integration.

5.4.2 Marketplace Infrastructures

Data marketplaces extend enterprise discovery systems by introducing economic and governance constraints into the discovery process. While enterprise catalogs such as GOODS and Unity Catalog focus on helping users locate and interpret datasets within an organization, marketplaces operate in settings where datasets are owned by independent providers and may be exchanged, licensed, or monetized (Fernandez 2023).

A defining characteristic of these systems is that discovery takes place under restricted visibility. Unlike enterprise settings, where systems can profile and index raw data, marketplace participants typically do not expose tuple-level access due to privacy, regulatory, or competitive concerns. As a result, discovery must rely on metadata, summaries, and controlled interaction mechanisms, rather than direct

inspection of data. This fundamentally changes the semantics of discovery: Users must assess relevance, compatibility, and utility without ever seeing the full dataset.

A central architectural pattern that enables this model is the data clean room. Clean rooms allow multiple parties to analyze data collaboratively while enforcing strict access controls. Instead of retrieving datasets, users submit queries that are executed within a controlled environment and only restricted outputs—typically aggregates or privacy-preserving statistics (Dwork 2008) are returned. Platforms such as Snowflake Clean Rooms, AWS Clean Rooms, and Google BigQuery Analytics Hub exemplify this approach, where discovery becomes a process of probing datasets indirectly through constrained queries.

From a systems perspective, clean rooms are implemented as secure execution layers tightly integrated with metadata and governance services. A catalog layer maintains dataset descriptors, schemas, and access policies, extending traditional metadata management with licensing terms and privacy constraints. Query execution is sandboxed within isolated compute environments or mediated through query rewriting, ensuring that only permitted operations are executed and that sensitive data is never exposed. To prevent leakage, systems enforce restrictions such as aggregate-only outputs, minimum group sizes, and query rate limits, often complemented by differential privacy mechanisms.

Because raw data cannot be accessed directly, discovery relies on shareable summaries and proxy representations. These include schema-level information, statistical sketches, or learned embeddings that approximate dataset content while remaining safe to expose. Some systems additionally support secure protocols, such as privacy-preserving set intersection or trusted execution environments, to evaluate overlap or joinability across datasets without revealing underlying values, though such techniques are typically applied selectively due to their computational cost.

Unlike enterprise discovery systems, marketplaces embed discovery within an economic and policy-driven framework. Users must evaluate datasets in terms of not only relevance but also cost, licensing conditions, and access restrictions, while systems must enforce auditability and compliance across all interactions. As a result, discovery shifts from a retrieval-centric task to a process of inference and decision-making under uncertainty, where users iteratively refine their understanding of partially observable datasets within a governed environment.

5.5 Conclusion

This chapter examined data discovery from a systems perspective and argued that the central challenge is not merely to design better isolated discovery operators but also to assemble complete end-to-end systems that support users throughout the discovery life cycle. The motivating user studies showed that practical discovery remains dominated by manual scripting, repeated querying, visual inspection, and tool chaining. At the same time, users frequently require more than simple keyword lookup: They need to identify joinable and unionable tables, reason about

correlations and compatibility, assess dataset quality and provenance, and iteratively refine their search as their understanding of the task evolves. These findings make clear that data discovery is fundamentally a systems problem, requiring integrated support for metadata extraction, indexing, ranking, interaction, and downstream-oriented reasoning.

A recurring lesson across the academic systems is that the choice of indexing substrate strongly shapes the kinds of discovery workflows a system can support. Precomputed graph structures such as AURUM's EKG enable rich navigational queries and explicit representation of inter-table relationships, but they also require up-front indexing and make it harder for previously unseen tables to participate immediately in discovery. Summary- and signature-based systems such as AUCTUS are more flexible with respect to heterogeneous repositories and can support content-aware search and augmentation efficiently, but they rely on compact abstractions whose fidelity bounds the quality of discovery. Relational approaches such as BLEND benefit from mature DBMS infrastructure, native indexing, and optimization opportunities, but they must encode discovery signals in forms that compile naturally to relational execution. In all cases, discovery quality depends heavily on what metadata is extracted, how current it remains, and how efficiently it can be maintained as data evolves.

The industrial systems surveyed in this chapter exhibit a different emphasis. GOODS and Unity Catalog show that, in enterprise environments, discovery is tightly coupled with governance, ownership, lineage, access control, and operational metadata management. Their primary value lies in making large and fragmented data estates legible and governable. Users can find datasets, inspect profiles, identify owners, understand provenance, and determine whether they are permitted to use an asset. Enterprise catalogs typically have a strong focus on metadata-centric search and navigation, yet they offer only limited support for discovery tasks such as finding unionable datasets, identifying strong JOIN candidates, or ranking assets by downstream integration utility. This gap suggests that metadata governance and relational discovery should not be seen as competing goals, but as complementary layers that future systems must combine more effectively. Marketplaces resemble a clear use case for data discovery in data lakes, and existing solutions cover constraint-based keyword search yet lack the functionality for more sophisticated discovery operations.

Overall, the chapter exposes a design space structured by several recurring tensions: offline precomputation versus online flexibility, general metadata search versus task-specific discovery operators, human-guided interaction versus automated recommendation, and catalog governance versus integration-oriented reasoning. Robust discovery systems must balance these tensions rather than optimize for only one dimension. They must provide expressive abstractions for complex discovery tasks while remaining operationally feasible at scale. They must automate metadata collection and ranking where possible, while still keeping users in the loop for interpretation, validation, and task refinement. And they must connect discovery not only to search interfaces but also to the execution substrates, governance policies,

and collaborative processes that determine whether discovered data can actually be reused.

Future discovery systems will likely need to unify enterprise metadata platforms with richer relational reasoning, support incremental participation of newly observed datasets without expensive full rescans, and incorporate workload- and task-awareness more directly into ranking and recommendation. Just as importantly, they will need to treat discovery as an iterative and context-dependent activity rather than a one-shot retrieval problem. End-to-end data discovery remains an open architectural challenge: one that sits at the intersection of data management, information retrieval, human–data interaction, and governance.

References

Broder AZ (1997) On the resemblance and containment of documents. In: Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171), IEEE, pp 21–29

Castelo S, Rampin R, Santos ASR, Bessa A, Chirigati F, Freire J (2021) Auctus: A dataset search engine for data discovery and augmentation. Proceedings of the VLDB Endowment (PVLDB) 14(12):2791–2794, URL http://www.vldb.org/pvldb/vol14/p2791-castelo.pdf

Chandra R, Chen H, Matharu R, Cai S, Chen J, Dutta P, Ghita B, Greenstein T, Holla G, Huang P, Huo Y, Ionescu A, Ispas A, Januschowski T, Karajgaonkar V, Leone S, Lewis D, Li A, Li N, Lian C, Link S, Lu Q, Ma Y, Pettitt C, Prabhakaran V, Raducanu B, Rong K, Roome P, Shetty S, Smith S, Sun X, Tang Y, Wen W, Xia L, Zeng J, Zhang B, Xin R, Zaharia M (2025) Unity catalog: Open and universal governance for the lakehouse and beyond. In: Proceedings of the ACM on Managanagement of Data (PACMMOD), Association for Computing Machinery, New York, NY, USA, SIGMOD/PODS '25, p 310–322, URL https://doi.org/10.1145/3722212.3724459

Deng Y, Chai C, Cao L, Yuan Q, Chen S, Yu Y, Sun Z, Wang J, Li J, Cao Z, Jin K, Zhang C, Jiang Y, Zhang Y, Wang Y, Yuan Y, Wang G, Tang N (2024) Lakebench: A benchmark for discovering joinable and unionable tables in data lakes. Proceedings of the VLDB Endowment (PVLDB) 17(8):1925–1938, URL https://www.vldb.org/pvldb/vol17/p1925-chai.pdf

Dwork C (2008) Differential privacy: A survey of results. In: International conference on theory and applications of models of computation, Springer, pp 1–19

Esmailoghli M, Quiané-Ruiz J, Abedjan Z (2022) MATE: multi-attribute table extraction. Proceedings of the VLDB Endowment (PVLDB) 15(8):1684–1696, URL https://www.vldb.org/pvldb/vol15/p1684-esmailoghli.pdf

Esmailoghli M, Schnell C, Miller RJ, Abedjan Z (2025) BLEND: A Unified Data Discovery System . In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE Computer Society, Los Alamitos, CA, USA, pp 737–750, DOI https://doi.org/10.1109/ICDE65448.2025.00061, URL https://doi.ieeecomputersociety.org/10.1109/ICDE65448.2025.00061

Fernandez RC (2018) Aurum: a story about research taste, Association for Computing Machinery and Morgan & Claypool, p 387–391. URL https://doi.org/10.1145/3226595.3226631

Fernandez RC (2023) Data-sharing markets: Model, protocol, and algorithms to incentivize the formation of data-sharing consortia. Proceedings of the ACM on Managanagement of Data (PACMMOD) 1(2):172:1–172:25, URL https://doi.org/10.1145/3589317

Fernandez RC, Abedjan Z, Koko F, Yuan G, Madden S, Stonebraker M (2018) Aurum: A data discovery system. In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE Computer Society, pp 1001–1012, URL https://doi.org/10.1109/ICDE.2018.00094

Fernandez RC, Min J, Nava D, Madden S (2019) Lazo: A cardinality-based method for coupled estimation of jaccard similarity and containment. In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE, pp 1190–1201, URL https://doi.org/10.1109/ICDE.2019.00109

Galhotra S, Gong Y, Fernandez RC (2023) Metam: Goal-oriented data discovery. In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE, pp 2780–2793, URL https://doi.org/10.1109/ICDE55515.2023.00213

Gong Y, Zhu Z, Galhotra S, Fernandez RC (2023) Ver: View Discovery in the Wild . In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE Computer Society, Los Alamitos, CA, USA, pp 503–516, DOI https://doi.org/10.1109/ICDE55515.2023.00045, URL https://doi.ieeecomputersociety.org/10.1109/ICDE55515.2023.00045

Gong Y, Galhotra S, Fernandez RC (2024) Nexus: Correlation discovery over collections of spatio-temporal tabular data. Proceedings of the ACM on Managanagement of Data (PACMMOD) 2(3):154, URL https://doi.org/10.1145/3654957

Halevy AY, Korn F, Noy NF, Olston C, Polyzotis N, Roy S, Whang SE (2016) Goods: Organizing google's datasets. In: Özcan F, Koutrika G, Madden S (eds) Proceedings of the International Conference on Management of Data (SIGMOD), ACM, pp 795–806, URL https://doi.org/10.1145/2882903.2903730

Hulsebos M, Lin W, Shankar S, Parameswaran A (2024) It took longer than i was expecting: Why is dataset search still so hard? In: Proceedings of the 2024 Workshop on Human-In-the-Loop Data Analytics, Association for Computing Machinery, New York, NY, USA, HILDA 24, p 1–4, URL https://doi.org/10.1145/3665939.3665959

Malaguti G, Mozzillo A, Simonini G (2025) OrQA – open data retrieval for question answering dataset generation. In: Chang S, Hulsebos M, Liu Q, Chen W, Sun H (eds) Proceedings of the Table Representation Learning Workshop (TRL), Association for Computational Linguistics, Vienna, Austria, pp 172–181, DOI https://doi.org/10.18653/v1/2025.trl-1.14, URL https://aclanthology.org/2025.trl-1.14/

Malkov YA, Yashunin DA (2020) Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE Trans Pattern Anal Mach Intell 42(4):824–836, URL https://doi.org/10.1109/TPAMI.2018.2889473

Santos ASR, Bessa A, Musco C, Freire J (2022) A sketch-based index for correlated dataset search. In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE, pp 2928–2941, URL https://doi.org/10.1109/ICDE53745.2022.00264

Zaharia M, Ghodsi A, Xin R, Armbrust M (2021) Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics. In: Proceedings of the Conference on Innovative Data Systems Research (CIDR), www.cidrdb.org, URL http://cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf

Chapter 6
Benchmarking Data Discovery Systems



Abstract This chapter examines the main evaluation dimensions for data discovery, including response time, result quality, storage overhead, indexing cost, and architectural complexity. It argues that traditional information retrieval metrics remain valuable but require careful adaptation because relevance in tabular discovery is not limited to topical similarity. A central focus is the distinction between closed-world and open-world evaluation. Closed-world benchmarks enable reproducible measurement against fixed ground truth, often through synthesized lakes derived from partitioned or perturbed tables, but risk being noisy or arbitrary. Open-world evaluation better reflects real data lakes, where relevance is incomplete and contextual, and therefore often relies on manual validation or downstream utility proxies such as machine learning augmentation performance. The chapter further reviews benchmark construction strategies for JOIN and UNION discovery, highlights known biases and limitations in existing datasets, and discusses the importance of measuring deployment and query time efficiency across diverse lake dimensions. Overall, it advocates a multidimensional evaluation framework that combines effectiveness, efficiency, and downstream utility to compare data discovery systems more fairly and realistically.

Keywords Discovery benchmarks · Union discovery · Join discovery · OWA · CWA

6.1 Challenges in Evaluation

Discovery is a user-facing systems problem in which usefulness depends on how effectively a system supports an end-to-end search process. This immediately raises the need for explicit evaluation metrics. First, the *response time* of discovery is crucial because discovery is rarely a one-shot activity. Users typically issue multiple queries, inspect intermediate candidates, reformulate their intent, and compare alternatives before selecting data for downstream use. A system that produces high-quality results but requires long delays at each interaction step can severely disrupt this iterative workflow. For this reason, discovery systems should be evaluated not

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027

Z. Abedjan et al., *Data Discovery in Data Lakes*,
https://doi.org/10.1007/978-3-032-30822-1_6

only by result quality but also by latency, throughput, and the ability to sustain interactive exploration under realistic workloads.

A second major challenge is the *notion of relevance and quality* itself. In classical information retrieval, relevance is often approximated by topical similarity between a query and a document. For data discovery, this is too narrow. A dataset may be relevant because it contains certain values, because it can be joined or unioned with another table, because it exhibits a useful correlation, or because it satisfies structural and semantic constraints required by a downstream task. Closely related is the broader criterion of *fitness for use*. Even if a dataset is relevant in principle, it may still be unsuitable in practice due to poor coverage, missing values, inappropriate granularity, outdated content, weak provenance, or incompatible schema structure. Similar to other research problems in data integration, there is a trade-off between resource consumption and quality of potential techniques. While measuring resource consumption and runtime is straightforward, it is often hard to set the proper criteria for the expected quality of data discovery results.

Finally, data discovery systems must also be assessed as *systems*, which means that storage overhead and implementation complexity matter alongside effectiveness. Many discovery approaches rely on rich indexes, summaries, graphs, embeddings, or auxiliary metadata structures to accelerate search and support advanced discovery operations. These structures can significantly improve discovery quality and speed, but they also introduce additional storage costs, maintenance burdens, and system complexity. A method that achieves strong discovery performance only by requiring incompatible and large auxiliary indexes may be impractical in real deployments. Accordingly, evaluation metrics for data discovery should capture not only discovery quality and latency but also storage footprint, update cost, extensibility, and overall architectural complexity.

In this chapter, we want to delve into these evaluation dimensions and hint at trade-offs. In particular, we want to discuss two different approaches for evaluating the result quality in data discovery: (1) close-world: A ground truth is created, often by fragmenting larger tables and trying to resynthesize the original tables; with this approach, metrics such as precision and recall can be calculated, and (2) open-world: Data discovery aims at finding serendipitous datasets that are useful for a downstream use case. Both approaches can bring insights into the capabilities of different techniques. We will have a deeper look into evaluation criteria for JOIN and UNION discovery in particular, present existing popular benchmarks, and highlight potential pitfalls that have been noted in the existing literature.

6.2 Effectiveness in Data Discovery

Before discussing existing benchmark datasets, we review the intuitions for evaluating data discovery systems and relate the methodology to existing research in information retrieval.

6.2.1 Background: Keyword Search in Information Retrieval

The standard evaluation setup uses a *test collection* consisting of (1) a document corpus, (2) a set of queries, and (3) relevance judgments indicating which documents are relevant to which queries as discussed in standard IR textbooks (Manning et al. 2008). This framework enables a systematic, repeatable comparison between retrieval systems.

A first class of evaluation measures focuses on *relevance set quality*. The two classical metrics are *precision* and *recall*. Precision measures the fraction of retrieved documents that are relevant, while recall measures the fraction of all relevant documents that were successfully retrieved. These metrics are useful for understanding the trade-off between retrieving many relevant items and avoiding nonrelevant ones.

For ranked keyword search, rank-sensitive metrics are usually more informative. Common measures include:

- **Precision@k**: the proportion of relevant documents among the top-k results
- **Recall@k**: the proportion of all relevant documents found within the top-k results
- **Mean Average Precision (MAP)**: the mean, across queries, of the average precision values computed over the ranked list. MAP rewards systems that rank relevant documents early and consistently.
- **Mean Reciprocal Rank (MRR)**: the average reciprocal of the rank position of the first relevant result. This is especially useful when users primarily want one correct answer quickly.
- **Normalized Discounted Cumulative Gain (nDCG)**: a metric that accounts for graded relevance and discounts relevant documents appearing lower in the ranking. It is often preferred when relevance is not purely binary.

In practice, metric choice should align with the application. If users usually inspect only the first page of results, metrics such as Precision@10 or nDCG@10 are appropriate. If the task requires finding as many relevant documents as possible, recall-oriented measures become more important.

A second major evaluation technique is *human relevance assessment*. In benchmark-style IR evaluation, assessors label retrieved documents as relevant or nonrelevant or assign graded relevance levels. A common methodology is *pooling*, where top results from multiple systems are combined and judged. This reduces labeling cost while providing a robust basis for comparison.

A third technique is *per-query error analysis*. Aggregate scores such as MAP or nDCG are useful, but they can hide failure modes. Query-level inspection helps identify whether a system performs poorly on ambiguous, rare, navigational, or long-tail queries.

A fourth technique is *online evaluation*. In real search systems, offline relevance metrics are often complemented by online measures such as click-through rate, dwell time, query reformulation rate, abandonment rate, and downstream task

completion. These are commonly evaluated via A/B testing to determine whether improvements in offline ranking metrics correspond to better user outcomes.

Overall, a sound evaluation methodology for keyword search combines offline relevance-based metrics, careful human judgment, per-query diagnostic analysis, and online user-centered validation. A practical default is to report nDCG@10, Recall@100, and query-level error analysis and then verify improvements through online experiments.

6.2.2 World Assumptions

In evaluating data discovery operations, the distinction between the *closed-world assumption* (CWA) and the *open-world assumption* (OWA) is fundamental because it determines what a benchmark is allowed to treat as a false negative. Under a *closed-world* evaluation, the benchmark assumes that the corpus and the relevance judgments are complete: If a joinable table, unionable table, or otherwise useful dataset is not annotated as relevant, then the system is penalized for returning it. This makes evaluation straightforward and supports standard IR-style metrics such as Precision@k, Recall@k, MAP, and nDCG, but it can be misleading in data lakes, where relevance sets are rarely exhaustive and many valid discovery results remain unlabeled. Under an *open-world* evaluation, by contrast, the absence of a labeled match is interpreted as "unknown" rather than "nonrelevant," reflecting the broader semantic Web view that lack of evidence is not evidence of absence. This is often the more realistic assumption for data lakes, because useful tables may exist outside the benchmark pool, may only become useful in a downstream integration context, or may represent alternative JOIN and UNION opportunities that annotators did not enumerate. The challenge is that OWA weakens the reliability of conventional recall-style evaluation: A system may appear to produce false positives when it is actually surfacing previously undiscovered valid results. As a result, evaluation of data discovery operations increasingly has to acknowledge incomplete ground truth, for example, by combining benchmark relevance sets with manual validation of sampled "false positives," by distinguishing confirmed nonrelevance from unlabeled candidates, or by measuring downstream task success in addition to ranking accuracy. In effect, CWA favors clean, reproducible benchmarking, whereas OWA better reflects the uncertainty and incompleteness of real data lakes.

6.2.3 Adaptation of Evaluation Metrics for Tabular Data Discovery

Unlike document retrieval, tabular data discovery must decide not only *whether* a keyword appears but also *where* and *how* it appears in the table. A match in a

schema element such as a table title, column header, or attribute name usually carries different evidence from a match in an instance value inside a cell, because schema-level matches often signal semantic intent, while instance-level matches may be sparse, noisy, or ambiguous (Zhang and Balog 2021; Zhang and Ives 2020).

Moreover, co-occurrence is more complex for tables than for flat text: Two query keywords may appear in the same row, in different columns, only in headers, or be distributed across schema and cell values, and these cases imply different kinds of relevance. As a result, table retrieval systems must reason over table structure, balancing evidence from captions, headers, columns, rows, and cell contents rather than treating the table as an unordered bag of words (Zhang and Balog 2021). This also creates new ambiguity around what counts as a relevant result: A table may be useful because it contains the queried entities in its cells, because its schema matches the desired attributes, or because it is joinable or unionable with another table needed for downstream analysis (Deng et al. 2024). Accordingly, adaptations of classical IR to data lakes increasingly distinguish schema- and value-based signals, incorporate structural and semantic matching at the column and table level, and evaluate relevance in terms of analytical usefulness rather than mere lexical overlap (Deng et al. 2024; Zhang and Ives 2020).

6.2.4 Evaluating JOIN *Discovery*

The strategy for evaluating JOIN discovery depends on the type of JOIN and the underlying world assumption. Under closed-world assumption, all information retrieval metrics introduced before are applicable if the goal of the discovery is to only identify exact joins. In this setup, the assumption is that all joinable tables can be identified. Usually, for this type of setup, the remaining evaluation criterion is runtime, which boils down to the inherent indexing and filtering mechanisms. Inverted indexes guarantee complete results, and approximations through summaries again introduce an uncertainty that can be measured via *Recall@k*. Precision is usually expected to remain at its maximum because all results even based on approximate and summary indexes undergo post-processing validation steps. A common approximation strategy for exact join search is to leverage bloom filter resembling techniques. For example, MATE, which is an algorithm for *multicolumn exact* JOIN discovery, uses a hash function to readily omit candidate tables and rows where multivalue join keys are guaranteed to be missed (Esmailoghli et al. 2022).

As we relax the equality aspect of JOIN discovery in direction of approximate or semantics JOINs, CWA and OWA introduce distinctive challenges. A closed-world evaluation assumes that all valid join partners for a query table have been enumerated in the benchmark. Yet, joinability is often latent: Two tables may be validly joinable through synonymous attributes, hidden key/foreign-key relations, approximate entity matches, or contextual semantic alignment that the ground truth does not capture. As a result, a system that surfaces a novel but valid join path may be penalized as if it had produced a false positive.

6.2.5 *Evaluating* UNION *Discovery*

Unionability depends on whether two tables share sufficiently compatible schemas or semantically aligned columns, and these correspondences are often partial, approximate, or only useful for certain downstream tasks. A closed-world benchmark therefore risks underestimating systems that identify previously unannotated but genuinely compatible tables. Under an open-world assumption, these unlabeled joinable or unionable tables are treated as *unknown* rather than *nonrelevant*, which better reflects the incompleteness of real data lakes. However, the evaluation consequences differ by task: For join discovery, the main challenge is hidden relational compatibility, while for UNION discovery, it is latent schema and column equivalence. In both cases, standard precision- and recall-based metrics remain useful, but only if interpreted with caution and complemented by methods that explicitly account for incomplete ground truth, such as manual assessment of sampled false positives or downstream validation of whether the discovered tables actually support integration and analysis.

6.3 Available Benchmarks

Now that we have introduced the relevant concepts on benchmarking data discovery systems, we review existing benchmarks and discuss pitfalls that are worth considering for fair evaluation of data discovery algorithms and systems.

6.3.1 *Synthesized Lakes for CWA*

For both JOIN and UNION discovery, there are curated benchmarks that have been derived from larger open table corpora.

6.3.1.1 UNION Discovery

A recent study on table UNION search benchmarks identifies for UNION table discovery four main categories of benchmarking strategies (Boutaleb et al. 2025): *partitioning-based*, *corpus-derived*, *LLM-generated*, and *hybrid*.

Partitioning-based benchmarks, such as TUS (Nargesian et al. 2018), SANTOS (Khatiwada et al. 2023), VALENTINE (Koutras et al. 2021), and LakeBench (Deng et al. 2024), construct unionable pairs by splitting existing seed tables, yielding clean and scalable labels but often creating strong schema and value overlap that can overestimate the performance of methods based on lexical or structural similarity.

LLM-generated benchmarks, such as Alt-gen (Pal et al. 2024), seek to introduce harder and more controlled examples, particularly challenging negatives, yet they also risk inconsistency in how unionability is defined and labeled across generated instances. Hybrid benchmarks combine synthetic and real-corpus strategies to improve coverage and realism, but they frequently inherit the limitations of both, including overlap artifacts, incomplete ground truth, and evaluation ambiguity. As a result, benchmark construction is not merely a practical detail: It directly shapes which methods appear effective and what notion of unionability is actually being evaluated.

6.3.1.2 Join Discovery

In general, benchmark datasets for JOIN discovery follow intuitions that are similar to benchmarks for UNION discovery. For exact JOIN evaluation, all sorts of open data lakes can be used. For measuring IR-specific metrics, such as Precision@k and Recall@k, typically a ground truth subset is generated. A common approach is to sample columns from lakes and label joinable columns (Dong et al. 2023; Guo et al. 2025). Some other lines of research generate ground truth by replicating tables and perturbing the join columns (Suri et al. 2021; Hu et al. 2025). Finally, a common approach for obtaining ground truth is to split existing tables vertically with or without value perturbation (Deng et al. 2024; Koutras et al. 2021).

6.3.2 Limitations of CWA Benchmarks

While the introduced approaches for evaluating UNION and (semantic) JOIN discovery enable the measurement of accuracy for the corresponding approach, they generally remain limited with regard to the assumptions that produced the ground truth. A recent paper analyzed the limitations of benchmarks for UNION discovery in breadth (Boutaleb et al. 2025), concluding that most benchmarks are significantly biased toward columns with high numbers of value overlap. This is confirmed by an analysis with simple baselines that achieve competitive performance by leveraging benchmark characteristics. Second, they notice that the ground truth labels are often arbitrary and conflicting. Seemingly joinable or unionable tables are not part of the ground truth, which makes it hard to properly evaluate metrics. The authors of the study suggest to actively reducing artifactual overlap while introducing controlled semantic heterogeneity to better reflect real-world schema and value divergence. Further, they recommend the incorporation of so-called realistic domain complexity beyond general vocabularies, addressing nondescriptive schemas and proprietary terms. Finally, they suggest to rethink ground truth representation by replacing binary labels with more nuanced labels.

In essence, while the criticism shows weaknesses regarding the generalization and practicality of the benchmarks, the observation is valid for other data integration

tasks that rely on similarity functions, such as schema matching (Koutras et al. 2021) and entity resolution (Papadakis et al. 2024). It is generally to be expected that similarity functions and features are domain and use case dependent and as such no heuristic will perform always best in predicting ground truth UNION and JOIN candidates. That is why, we argue that beyond identification of real-world benchmarks the goodness of UNION and JOIN discovery algorithms should be evaluated with regard to downstream benefits within data pipelines as we discuss next.

6.3.3 Utility-Driven Data Discovery for OWA

In OWA settings, the quality of discovery results is not compared against a ground truth. Rather the quality of results is either checked manually after retrieval or measured through downstream application proxy scores.

Typical benchmarks for proxy scores include experimental setups where the discovered datasets are used for augmenting a machine learning dataset. In these setups, the quality of the discovered results is measured through the accuracy scores of the downstream machine learning model (Esmailoghli et al. 2021; Santos et al. 2021, 2022; Galhotra et al. 2023; Chepurko et al. 2020). There are two benefits of such setups. First, there is no need for curating ground truth and running into aforementioned fallacies. Second, the obtained score is more expressive regarding the usefulness of the obtained tables and is closer to the application. However, also this approach has its own fallacies. First, the choice of the input datasets that are used as queries is crucial. A poor dataset that initially has a poor set of features is more likely to improve upon augmentation than a curated dataset. In the former case, approaches that are less conservative with their results typically lead to better results as they always retrieve candidates even with spurious correlations that improve an initial F1 or R2 score, while more conservative approaches might miss chances by filtering them out. On the other hand, if a dataset is already well-curated, it is not possible at all to identify any additional useful features that change the downstream scores, making the evaluation pointless.

6.4 Efficiency in Data Discovery

Data discovery usually undergoes an offline indexing deployment phase and an online query phase. Most research has so far focused on improving the latter, while the former has been rather neglected

6.4.1 Deployment Costs

Efficiency aspects in the offline phase refer to index size and indexing time. Depending on the type of index, the complexity of indexing is driven by number of tables, number of columns, or average number of rows.

Most research on discovery algorithms pays little attention to the required index size, as one assumes abundance of cheap storage. The typical index structures for data discovery are either variations of inverted indexes (Fernandez 2018; Becktepe et al. 2023) or HNSW-based vector indexes (Ma et al. 2023; Bogatu et al. 2020). As a result, the index size has been usually a side note in most papers. While the argument on downplaying storage requirements is in general a pragmatic one, there are two scenarios where index size can become a bottleneck. First, some approaches, such as JOSIE (Zhu et al. 2019), require the index to be maintained in main memory. Memory is clearly more limited than nonvolatile storage mediums. Second, so far the proposed indexes often serve one task at a time. As a result, a holistic discovery system would require multiple indexes to be deployed and kept consistent. Recent holistic systems, such as BLEND (Esmailoghli et al. 2025), try to highlight this aspect by providing unified indexes that cater multiple discovery tasks instead of deploying parallel operation-specific indexes.

Similar to index size, the aspect of indexing time has also been of low priority for most approaches as long as the indexing effort remains within feasible bounds, in particular, indexing techniques that focus on metadata, scale linearly with the number of tables and columns, and have the smallest runtime (Nargesian et al. 2020), followed by approaches that are based on inverted indexes that require a full scan of the lake tables (Esmailoghli et al. 2022; Santos et al. 2021, 2022). Indexing that requires a pairwise comparison of columns for a priori JOIN path materialization (Fernandez et al. 2018) is typically bounded by the number of columns due to the quadratic complexity. More recent approaches for indexing that also enable question answering (Balaka et al. 2025) introduce an additional table and column summarization overhead that can become substantial for large lakes. Here, the number of rows can also become a bottleneck. Finally, learning-based approaches further depend on the parameterization of the embedding models used for pretraining and fine-tuning models (Fan et al. 2023; Hu et al. 2025).

Incremental updates to data lake indexes is rather straightforward for inverted indexes. Both types of systems, whether they rely on document retrieval indexes (Castelo et al. 2021) or those implemented as database relations (Esmailoghli et al. 2025) are straightforward to extend. Aurum (Fernandez et al. 2018) and Pneuma (Balaka et al. 2025) mitigate the required quadratic computation per table and the summarization overhead by invoking index calculation after a threshold of data updates has been reached.

With the aforementioned considerations, benchmarking the efficiency of indexes formally requires the existence of lake variations that scale in all three dimensions: number of tables, number of columns, and number of rows. However, with the lack of real discovery workloads and repositories, it is generally difficult to assess

Table 6.1 Data lakes used in the experiments

Data lake	Tables	Columns	Rows
DWTC (Eberius et al. 2015)	145M	760M	1.5B
Lakebench Webtable Large (Deng et al. 2024)	2.8M	14.8M	63.7M
Gittables (Hulsebos et al. 2023)	1.5M	16.8M	345M
German Open Data (Govdata 2025)	17,144	440K	62M
WDC (Bizer et al. 2015)	–	163M	1.6B
Canada, US, and UK Open Data Zhu et al. (2019)	–	745K	1.1B
TUS (Nargesian et al. 2018)	1,530	14.8K	6.8M
TUS Large (Nargesian et al. 2018)	5,043	55K	9.6M
SANTOS (Khatiwada et al. 2023)	550	6,322	3.8M
SANTOS Large (Khatiwada et al. 2023)	11,090	121K	85M
NYC open data (cityofnewyork.us 2025)	1,063	16K	290M

trade-offs realistically and spark interest in researching efficient indexing and lake preparation techniques.

Table 6.1 provides an overview of the common lakes that are used for benchmarking data discovery systems. It shows that Web tables (Cafarella et al. 2008) and Gittables (Hulsebos et al. 2023) are among those with large numbers of tables and columns. While the former contains mostly small tables of on average 12 records per table, it contains more than 100 million tables. Gittables comprises fewer tables which on average contain an order of magnitude more rows. The NYC Open Data dataset is comparably much smaller yet also contains very large tables as noticable by the high ratio of rows divided by tables.

6.4.2 Query Runtime

Query runtime depends on both the dimensions of the input and the dimensions of the lake. While the same repositories mentioned for evaluating efficiency aspects of deployment cover a broad and diverse range of dimensions for data lakes, benchmarking query runtime also requires the consideration of large input tables. Depending on the underlying index, the initial retrieval of relevant tables might be even of constant runtime complexity. An important parameter in evaluation setups is the top-k filter upon which some approaches might be highly efficient for the head tail of results because retrieval metrics align well with the top-k parameter. Others might show better performance for large k because the cost of a k independent intermediate step will be amortized. This phenomenon has been observed when comparing Starmie (Fan et al. 2023) with the UNION discovery engine in BLEND (Esmailoghli et al. 2025).

For UNION discovery, an initial pass over the table for identifying table representations is typically needed which should be a linear scan. For JOIN discovery,

the situation becomes more complicated because the materialization of the JOIN might require additional passes of the input table scaling with the average size of a retrieved table, which is why most discovery techniques foremostly focus on calculating the joinability score and defer the actual joining step (Zhu et al. 2019; Esmailoghli et al. 2021). An important consideration, when comparing JOIN discovery techniques, is to track whether the runtime performance includes the actual JOIN or not. Some techniques might be slower in identifying joinability but provide a better end-to-end performance.

Another major aspect that might impact query runtime is discovery that is entangled with utility functions, as in the case of augmentation for ML (Chepurko et al. 2020; Galhotra et al. 2023; Santos et al. 2022). Here additional trade-offs come into play as the more candidate tables an approach retrieves the more JOIN materializations become necessary and additional computationally heavy post-processing feature selection steps might arise. Assessing query runtime in a transparent way requires a clear ablation of the runtime for the individual steps with the pipeline and trade-off discussions.

Finally, the assessment of efficiency at query time might also require workloads that require multiple interdependent discovery operations. So far, there have only been a handful of such workloads in the literature (Esmailoghli et al. 2025) and no systematic evaluation of so-called discovery pipelines.

6.4.3 Ease of Use

So far, there is no literature in data discovery that focuses on usability of data discovery systems. Accordingly, there are no established evaluation criteria. Deployment costs already capture usability aspects from a developer or lake administration perspective. On the user side, the completeness of a language, its comprehensibility, and interaction model play an important role. The completeness of a discovery language usually requires a formal proof or qualitative comparisons in case of languages that do not subsume each other's capabilities. The comprehensibility of a language and the interaction model with a data discovery system can be assessed through quantities, such as lines of code per query, or qualitatively through user studies.

6.5 Conclusion

Evaluating data discovery systems requires a slightly different perspective on result quality than classical information retrieval because discovery is not merely about retrieving topically relevant items but also about supporting an iterative, user-facing process of finding data that is actually useful for downstream work. Throughout this chapter, we argued that no single metric is sufficient to capture this objective. It

turns out that meaningful evaluation must jointly consider effectiveness, efficiency, and system-level practicality.

On the effectiveness side, classical IR metrics such as Precision@k, Recall@k, MAP, MRR, and nDCG remain important building blocks, but they must be interpreted carefully when transferred to tabular data discovery. In particular, relevance in data lakes is multidimensional: A table may be useful because of lexical overlap, schema compatibility, joinability, unionability, semantic relatedness, or downstream analytical value. This makes benchmark design nontrivial. Under the closed-world assumption, curated ground truth enables reproducible comparisons, but it often oversimplifies the discovery problem and risks rewarding methods that exploit benchmark artifacts rather than genuine discovery capability. Under the open-world assumption, evaluation better reflects realistic data lakes, yet designing such benchmarks remains a hard problem. The discussion of existing benchmarks further shows that current evaluation practice is still maturing. Synthesized benchmarks for JOIN and UNION discovery have enabled substantial progress, yet many inherit strong assumptions about overlap, schema regularity, and completeness of labels. At the same time, utility-driven open-world evaluations offer a promising alternative, but they introduce their own sensitivities, for example, to the choice of downstream task and the initial quality of the query data. This suggests that future benchmarking efforts should move toward more diverse, realistic, and multidimensional evaluation settings that better capture the heterogeneity of real data lakes and the end-to-end value of discovery.

We also highlighted that data discovery systems must be evaluated as systems. Query latency, throughput, indexing time, storage overhead, update cost, and implementation complexity all shape whether a method is deployable in practice. This is particularly important because many approaches achieve strong retrieval performance only through rich auxiliary structures, expensive preprocessing, or task-specific indexes. A fair comparison must therefore account not only for the quality of discovered results but also for the resources and engineering assumptions needed to obtain them.

Overall, the main takeaway of this chapter is that progress in the field depends on evaluation frameworks that make trade-offs explicit: between closed-world reproducibility and open-world realism, between retrieval accuracy and downstream usefulness, and between search quality and system efficiency. Only by embracing these trade-offs can future work be compared fairly and can data discovery systems be developed that are not only accurate in isolation but also truly useful in practice.

References

Balaka MIL, Alexander D, Wang Q, Gong Y, Krisnadhi A, Fernandez RC (2025) Pneuma: Leveraging llms for tabular data representation and retrieval in an end-to-end system. Proceedings of the ACM on Managanagement of Data (PACMMOD) 3(3):200:1–200:28, URL https://doi.org/10.1145/3725337

Becktepe J, Esmailoghli M, Koch M, Abedjan Z (2023) Demonstrating MATE and COCOA for data discovery. In: Proceedings of the International Conference on Management of Data (SIGMOD), ACM, pp 119–122, URL https://doi.org/10.1145/3555041.3589716

Bizer C, Meusel R, Lehmberg O, Ritze D, Zope S (2015) Wdc web table corpora. https://webdatacommons.org/webtables/, accessed: 2025-03-26

Bogatu A, Fernandes AAA, Paton NW, Konstantinou N (2020) Dataset discovery in data lakes. In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE, pp 709–720, URL https://doi.org/10.1109/ICDE48307.2020.00067

Boutaleb A, Amann B, Naacke H, Angarita R (2025) Something's fishy in the data lake: A critical re-evaluation of table union search benchmarks. In: Chang S, Hulsebos M, Liu Q, Chen W, Sun H (eds) Proceedings of the Table Representation Learning Workshop (TRL), Association for Computational Linguistics, Vienna, Austria, pp 71–85, DOI https://doi.org/10.18653/v1/2025.trl-1.7, URL https://aclanthology.org/2025.trl-1.7/

Cafarella MJ, Halevy A, Wang DZ, Wu E, Zhang Y (2008) Webtables: exploring the power of tables on the web. Proceedings of the VLDB Endowment (PVLDB) 1(1):538–549

Castelo S, Rampin R, Santos ASR, Bessa A, Chirigati F, Freire J (2021) Auctus: A dataset search engine for data discovery and augmentation. Proceedings of the VLDB Endowment (PVLDB) 14(12):2791–2794, URL http://www.vldb.org/pvldb/vol14/p2791-castelo.pdf

Chepurko N, Marcus R, Zgraggen E, Fernandez RC, Kraska T, Karger DR (2020) ARDA: automatic relational data augmentation for machine learning. Proceedings of the VLDB Endowment (PVLDB) 13(9):1373–1387, DOI https://doi.org/10.14778/3397230.3397235, URL http://www.vldb.org/pvldb/vol13/p1373-chepurko.pdf

cityofnewyorkus (2025) Nyc open data. https://opendata.cityofnewyork.us//, accessed: 2024-07-17

Deng Y, Chai C, Cao L, Yuan Q, Chen S, Yu Y, Sun Z, Wang J, Li J, Cao Z, Jin K, Zhang C, Jiang Y, Zhang Y, Wang Y, Yuan Y, Wang G, Tang N (2024) Lakebench: A benchmark for discovering joinable and unionable tables in data lakes. Proceedings of the VLDB Endowment (PVLDB) 17(8):1925–1938, URL https://www.vldb.org/pvldb/vol17/p1925-chai.pdf

Dong Y, Xiao C, Nozawa T, Enomoto M, Oyamada M (2023) Deepjoin: Joinable table discovery with pre-trained language models. Proceedings of the VLDB Endowment (PVLDB) 16(10):2458–2470, DOI https://doi.org/10.14778/3603581.3603587, URL https://www.vldb.org/pvldb/vol16/p2458-dong.pdf

Eberius J, Thiele M, Braunschweig K, Lehner W (2015) Top-k entity augmentation using consistent set covering. In: Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM), p 8

Esmailoghli M, Quiané-Ruiz J, Abedjan Z (2021) COCOA: correlation coefficient-aware data augmentation. In: Velegrakis Y, Zeinalipour-Yazti D, Chrysanthis PK, Guerra F (eds) Proceedings of the International Conference on Extending Database Technology (EDBT), OpenProceedings.org, pp 331–336, URL https://doi.org/10.5441/002/edbt.2021.30

Esmailoghli M, Quiané-Ruiz J, Abedjan Z (2022) MATE: multi-attribute table extraction. Proceedings of the VLDB Endowment (PVLDB) 15(8):1684–1696, URL https://www.vldb.org/pvldb/vol15/p1684-esmailoghli.pdf

Esmailoghli M, Schnell C, Miller RJ, Abedjan Z (2025) BLEND: A Unified Data Discovery System . In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE Computer Society, Los Alamitos, CA, USA, pp 737–750, DOI https://doi.org/10.1109/ICDE65448.2025.00061, URL https://doi.ieeecomputersociety.org/10.1109/ICDE65448.2025.00061

Fan G, Wang J, Li Y, Zhang D, Miller RJ (2023) Semantics-aware dataset discovery from data lakes with contextualized column-based representation learning. Proceedings of the VLDB Endowment (PVLDB) 16(7):1726–1739, URL https://www.vldb.org/pvldb/vol16/p1726-fan.pdf

Fernandez RC (2018) Aurum: a story about research taste, Association for Computing Machinery and Morgan & Claypool, p 387–391. URL https://doi.org/10.1145/3226595.3226631

Fernandez RC, Abedjan Z, Koko F, Yuan G, Madden S, Stonebraker M (2018) Aurum: A data discovery system. In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE Computer Society, pp 1001–1012, URL https://doi.org/10.1109/ICDE.2018.00094

Galhotra S, Gong Y, Fernandez RC (2023) Metam: Goal-oriented data discovery. In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE, pp 2780–2793, URL https://doi.org/10.1109/ICDE55515.2023.00213

Govdata (2025) Das datenportal für deutschland. https://www.govdata.de/, accessed: 2024-07-17

Guo Y, Mao Y, Hu Z, Chen L, Gao Y (2025) Snoopy: Effective and Efficient Semantic Join Discovery via Proxy Columns . IEEE Transactions on Knowledge and Data Engineering (TKDE) 37(05):2971–2985, DOI https://doi.org/10.1109/TKDE.2025.3545176, URL https://doi.ieeecomputersociety.org/10.1109/TKDE.2025.3545176

Hu X, Lei C, Qin X, Katsifodimos A, Faloutsos C, Rangwala H (2025) Polyjoin: Semantic multi-key joinable table search in data lakes. In: Chiruzzo L, Ritter A, Wang L (eds) Findings of the Association for Computational Linguistics: (NAACL), Association for Computational Linguistics, pp 384–395, URL https://doi.org/10.18653/v1/2025.findings-naacl.23

Hulsebos M, Demiralp Ç, Groth P (2023) Gittables: A large-scale corpus of relational tables. Proc ACM Manag Data 1(1):30:1–30:17, URL https://doi.org/10.1145/3588710

Khatiwada A, Fan G, Shraga R, Chen Z, Gatterbauer W, Miller RJ, Riedewald M (2023) SANTOS: relationship-based semantic table union search. Proceedings of the ACM on Managanagement of Data (PACMMOD) 1(1):9:1–9:25, URL https://doi.org/10.1145/3588689

Koutras C, Siachamis G, Ionescu A, Psarakis K, Brons J, Fragkoulis M, Lofi C, Bonifati A, Katsifodimos A (2021) Valentine: Evaluating matching techniques for dataset discovery. In: 37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021, IEEE, pp 468–479, URL https://doi.org/10.1109/ICDE51399.2021.00047

Ma X, Teofili T, Lin J (2023) Anserini gets dense retrieval: Integration of lucene's HNSW indexes. In: Frommholz I, Hopfgartner F, Lee M, Oakes M, Lalmas M, Zhang M, Santos RLT (eds) Proceedings of the International Conference on Information and Knowledge Management (CIKM), ACM, pp 5366–5370, URL https://doi.org/10.1145/3583780.3615112

Manning CD, Raghavan P, Schütze H (2008) Introduction to Information Retrieval. Cambridge University Press, USA

Nargesian F, Zhu E, Pu KQ, Miller RJ (2018) Table union search on open data. Proceedings of the VLDB Endowment (PVLDB) 11(7):813–825, DOI https://doi.org/10.14778/3192965.3192973, URL http://www.vldb.org/pvldb/vol11/p813-nargesian.pdf

Nargesian F, Pu KQ, Zhu E, Bashardoost BG, Miller RJ (2020) Organizing data lakes for navigation. In: Maier D, Pottinger R, Doan A, Tan W, Alawini A, Ngo HQ (eds) Proceedings of the International Conference on Management of Data (SIGMOD), ACM, pp 1939–1950, URL https://doi.org/10.1145/3318464.3380605

Pal K, Khatiwada A, Shraga R, Miller RJ (2024) Alt-gen: Benchmarking table union search using large language models. In: In Proceedings of the VLDB 2024 Workshop: Tabular Data Analysis Workshop (TaDA)

Papadakis G, Kirielle N, Christen P, Palpanas T (2024) A critical re-evaluation of record linkage benchmarks for learning-based matching algorithms. In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE, pp 3435–3448, URL https://doi.org/10.1109/ICDE60146.2024.00265

Santos ASR, Bessa A, Chirigati F, Musco C, Freire J (2021) Correlation sketches for approximate join-correlation queries. In: Li G, Li Z, Idreos S, Srivastava D (eds) Proceedings of the International Conference on Management of Data (SIGMOD), ACM, pp 1531–1544, URL https://doi.org/10.1145/3448016.3458456

Santos ASR, Bessa A, Musco C, Freire J (2022) A sketch-based index for correlated dataset search. In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE, pp 2928–2941, URL https://doi.org/10.1109/ICDE53745.2022.00264

Suri S, Ilyas IF, Ré C, Rekatsinas T (2021) Ember: no-code context enrichment via similarity-based keyless joins. Proceedings of the VLDB Endowment (PVLDB) 15(3):699–712, URL https://doi.org/10.14778/3494124.3494149

Zhang S, Balog K (2021) Semantic table retrieval using keyword and table queries. ACM Transansactions on the Web 15(3), URL https://doi.org/10.1145/3441690

Zhang Y, Ives ZG (2020) Finding related tables in data lakes for interactive data science. In: Maier D, Pottinger R, Doan A, Tan W, Alawini A, Ngo HQ (eds) Proceedings of the International Conference on Management of Data (SIGMOD), ACM, pp 1951–1966, URL https://doi.org/10.1145/3318464.3389726

Zhu E, Deng D, Nargesian F, Miller RJ (2019) JOSIE: overlap set similarity search for finding joinable tables in data lakes. In: Boncz PA, Manegold S, Ailamaki A, Deshpande A, Kraska T (eds) Proceedings of the International Conference on Management of Data (SIGMOD), ACM, pp 847–864, URL https://doi.org/10.1145/3299869.3300065

Chapter 7
Conclusions



Abstract This chapter concludes the book with a summary of all previous chapters and an outlook into future research in data discovery.

Keywords Holistic systems · Multimodality · Temporality · Agentic discovery

7.1 Summary

In this book we discussed data lakes and argued why they require specialized discovery mechanisms. Since users often lack prior knowledge of available schemas, value distributions, and inter-table relationships, discovery systems must support ad hoc access through search, matching, and ranking rather than through carefully curated warehouse schemas. We distinguished data discovery from standard information retrieval by emphasizing that discovered datasets must be usable as analytical assets, not merely readable results.

We provided taxonomies of discovery settings, operations, and indexes. A discovery setting characterizes discovery environments along dimensions such as locality, visibility, and mutability, covering centralized, federated, and hybrid repositories, as well as privacy- and governance-constrained scenarios. On top of these settings, we highlighted the core discovery operations: keyword search, union search, join search, natural-language querying, and composed multistep discovery tasks. We discussed assumptions and desiderata.

We then surveyed the algorithmic and systems foundations needed to scale discovery. Among the indexing strategies that were covered, there were structures that support efficient matching for joinability, unionability, correlation detection, and semantic similarity in large repositories, as well as recent trends such as hybrid retrieval, dense representations, and LLM-augmented discovery pipelines, positioning foundation models as semantic aids rather than replacements for database-style indexing and optimization.

Beyond algorithms, we review end-to-end discovery systems from both academia and industry, highlighting how practical systems combine metadata management, indexing, retrieval, and interaction support. We examine benchmarking and evalua-

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027

Z. Abedjan et al., *Data Discovery in Data Lakes*,
https://doi.org/10.1007/978-3-032-30822-1_7

tion, noting that discovery quality cannot be assessed solely with traditional retrieval metrics. Utility for downstream tasks, open-world assumptions, deployment cost, and runtime efficiency all matter.

Overall, we believe that data discovery is a core systems problem for modern data management research with many innovative solutions. Still there are many open problems as we discuss next.

7.2 Future Directions

Throughout the previous chapters, we hinted at several hard problems that are either topics of current research or, from our point of view, deserve more broad attention. It is important to stress that the prerequisite for any impactful future research is the establishment of curated real-world discovery workloads and data lakes that can serve as benchmarks.

7.2.1 Holistic Data Discovery Systems

Existing data discovery systems (Esmailoghli et al. 2025; Castelo et al. 2021; Fernandez et al. 2018) provide strong foundations, but most are still optimized for narrow discovery tasks rather than full exploratory workflows. BLEND (Esmailoghli et al. 2025) strides into that direction, but major limitations remain.

While existing systems already try to unify indexes for multiple operations, there is still no system that smoothly glues together value-based, sketch-based, and embedding-based indexes. Exact and value-based indexes remain essential for high-precision keyword lookup, schema filtering, and join-oriented search. Sketch- and overlap-based indexes remain important for efficient union and join candidate generation. At the same time, embedding-based indexes add a capability that classical structures lack: retrieval by semantic similarity rather than only syntactic overlap. Accordingly, future systems should maintain symbolic and vector indexes in parallel and treat retrieval as a fusion problem rather than a single-method lookup.

A second avenue is *hybrid retrieval and generation*. Recent work already suggests that the most practical architecture is not to replace indexing with foundation models, but to combine the two. Fast index structures should generate candidate datasets with predictable latency, while large language models or learned rerankers should be reserved for steps that require semantic judgment, contextual reasoning, or disambiguation. This division of labor is critical for large-scale lakes: Indexes provide sublinear access and cost control, while language models contribute the semantic flexibility needed for vocabulary mismatch, paraphrase, and intent interpretation.

Third, holistic systems should expose a *natural-language interface*. Users do not simply want “similar tables”; they often want a joinable customer table, a union-

compatible collection of logs, or a dataset that can augment an existing analysis. A natural-language interface should therefore translate user requests into explicit discovery operators such as keyword search, join search, union search, augmentation, or multistep discovery. For example, a request such as "find churn data I can join with billing" should compile into a workflow that retrieves semantically relevant churn datasets, detects plausible join keys, estimates join quality, and presents ranked alternatives with explanations. This makes natural language a control plane for discovery, not merely another query modality.

Fourth, future systems must support *iterative discovery as a first-class workflow*. Recent studies emphasize that practical data discovery remains highly manual, iterative, and composition-oriented: Users repeatedly search, inspect, reformulate, compare, and combine candidate datasets. A holistic system should therefore preserve session state across these steps. It should remember which tables the user accepted or rejected, which columns looked promising as keys, which interpretations of a query were ambiguous, and which branch of the search process the user is currently exploring. Conceptually, this suggests maintaining a *discovery session graph* that records user actions, intermediate candidates, inferred relationships, and downstream analysis intent.

Fifth, such systems should incorporate *prefetching and speculative execution*. In current systems, discovery often feels slow because every refinement step triggers a new round of expensive computation: candidate generation, overlap testing, joinability estimation, semantic reranking, sample extraction, or explanation generation. A more advanced architecture should anticipate likely next actions and proactively compute the artifacts needed for them. For example, once a user opens a candidate table, the system can prefetch likely join partners, union-compatible neighbors, approximate key profiles, representative samples, and brief natural-language summaries. Once a user expresses a semantic intent, the system can speculatively retrieve both lexical and vector neighbors, so that reformulations and follow-up questions can be answered with low latency. Prefetching is especially valuable in iterative discovery because the user's next step is often predictable from the current one.

Sixth, ranking must evolve from topical relevance toward *fitness for use*. A holistic discovery system should rank results not only by semantic similarity but also by joinability, unionability, key quality, provenance, freshness, coverage, policy constraints, and expected downstream utility. In practice, this means that the ranking function should be operator-aware. A table that is semantically close to the query but impossible to integrate may be less useful than a slightly less relevant table with clean keys, good coverage, and trustworthy lineage. Embeddings and language models are valuable here because they help estimate semantic match and generate explanations, but they should be integrated with structural and governance-aware signals rather than used alone.

In summary, both research and engineering should not focus solely on embeddings, natural-language querying, or iterative search as isolated features. Rather, the goal is to build systems in which these capabilities reinforce one another.

7.2.2 Time and Lake Versioning

So far, data discovery considers datasets within a lake as static artifacts, yet data lakes routinely accumulate multiple versions of the same entity as sources are updated, corrected, or restructured over time. Throughout the life cycle of a data lake, multiple versions of the same dataset might reside inside a lake. As a result, a join query may now return several candidates that are structurally compatible but reflect entirely different historical snapshot of the data. At the same time, the user might not always be interested in the most up-to-date dataset but depending on the use case might also be interested in older version of a dataset that exists in the lake. Current system, neither from academic nor from the industrial tools, provide any mechanism to reason about which version of a dataset actually fits the temporal context of a query (Esmailoghli and Weidlich 2025). Research on change discovery (Bleifuß et al. 2017, 2018) and data versioning (Bhardwaj et al. 2015) provides methods for managing and versioning datasets. These methods have so far not been considered in the context of data discovery.

7.2.3 Decentralized Data Discovery

Modern data ecosystems are increasingly decentralized, spanning multiple organizations, platforms, and governance regimes. In such settings, discovery must operate under partial visibility and autonomy constraints, where each participant exposes only limited metadata or query interfaces. As a result, global indexing is infeasible, and discovery becomes a federated coordination problem rather than a centralized search task.

A practical approach is to rely on metadata exchange and summary-based indexing, where nodes publish schemas, sketches, or embeddings that can be queried without exposing raw data. Query federation techniques can then route discovery requests across nodes and aggregate results, approximating global discovery under constraints (Halevy et al. 2006; Fernandez 2023). However, these methods must balance completeness, efficiency, and privacy.

Decentralization also introduces trust and incentive challenges. Since data providers may have conflicting objectives, discovery systems must incorporate signals such as provenance, reliability, and usage history into ranking (Song et al. 2025). Emerging paradigms such as data spaces and federated data markets further emphasize the need for discovery mechanisms that are robust to heterogeneity and governance constraints (Halevy et al. 2006).

An important emerging direction in decentralized settings is the use of *operator-aware, sketch-based discovery*. Since raw data cannot be shared, nodes can expose compact summaries such as MinHash signatures, sketches, or embeddings that enable approximate evaluation of discovery operators including join search, union search, and correlation detection across nodes (Broder 1997; Indyk and Motwani

1998). This suggests a shift toward sketch-first architectures, where summaries serve as the primary interface for indexing and query processing, and more expensive secure computation or query execution is reserved for a small set of high-confidence candidates. Recent approaches suggest to expose embedding pipelines in place of sketches Behme et al. (2026). This way one could discover joinable and unionable tables across multiple sites without probing the raw data. While the assumptions might still be too strong on these requirements, they are already paving the way forward toward decentralized setups and useful site profiles for data discovery. Designing unified, operator-aware sketches that support multiple discovery tasks under restricted visibility remains an open and promising research direction.

7.2.4 Multimodal Data Discovery

So far, data management research has mostly focused on setups with tabular datasets. This focus has been productive because tables are omnipresent (Van Breugel and Van Der Schaar 2024) and also from a technical point of view provide a clear abstraction for discovery operations: Users search for relevant datasets, identify union-compatible artifacts to extend coverage, or detect joinable artifacts to enrich an existing analysis. Much of the current literature and most practical systems are therefore built around table-centric assumptions about schema, keys, overlap, and structural compatibility.

However, modern data lakes are not predominantly tabular repositories. They contain mixtures of tables, text collections, images, geospatial layers, time series, graphs, and other domain-specific artifacts. There are domains in which such retrieval problems are already highly relevant and practically important. Health is a particularly clear example: Analysts may need to relate structured hospital records, textual clinical narratives, medical images, or sensor traces within a single analytical process. Similar needs arise in environmental monitoring, scientific data management, smart cities, and other domains where heterogeneous evidence must be combined. As a result, the future of data discovery must shift from *tabular discovery in isolation* toward *multimodal discovery*.

This shift should not be misunderstood as a call to replace existing discovery methods with cross-modal retrieval (Wang et al. 2024). Cross-modal retrieval already exists as an active and well-established area of research. In many settings, it is already possible to retrieve multiple modalities and even extract text from images, images from text, or structured and unstructured artifacts through shared embedding spaces (Radford et al. 2021). What remains missing, however, is a conception of cross-modal retrieval from the same lens that has made tabular data discovery so useful. In tabular discovery, retrieval is not an end in itself. The point is not merely to find a relevant artifact, but to discover artifacts that can be *used* to produce new analytical artifacts. Join search is valuable because it enables horizontal enrichment. Union search is valuable because it enables vertical extension. More generally, discovery matters because it supports the construction of downstream analytical

objects: integrated tables, derived views, features, evidence sets, or query-specific data products. Current cross-modal retrieval systems (Eltabakh et al. 2023) typically stop too early. They retrieve semantically related objects across modalities, but they do not yet treat those objects as operands in a broader discovery-and-integration workflow.

A multimodal discovery system should therefore ask questions analogous to those that have shaped tabular discovery research. Instead of only asking whether an image, text document, or time series is relevant to a query, the system should ask whether it is *operationally compatible* with an existing analytical artifact. Can a text corpus be linked to a table through shared entities, events, or temporal anchors? Can a geospatial raster be aligned with a tabular dataset through location and resolution constraints? Can an image collection be aggregated into features that can augment a predictive model built from tables? Can multiple modalities be combined to synthesize a new artifact with a well-defined analytical role? These are the multimodal analogs of joinability, unionability, and augmentation, but they have not yet been developed with the same clarity or system support as in the tabular setting.

This implies that the next step for the field goes beyond better shared embeddings or better cross-modal ranking. It is the development of *multimodal discovery operators*. Such operators should capture how artifacts from different modalities can be related, transformed, and composed to produce analysis-ready outputs. In a multimodal lake, an operator may produce a table enriched with features extracted from text, images, or spatial data. It might create a temporally aligned panel built from sensor streams and relational events. The key point is that discovery should be evaluated not just by relevance but also by its ability to support artifact creation for downstream use.

The central challenge is to generalize the notion of compatibility beyond schemas and keys toward cross-modal notions of alignment, linkability, aggregatability, and semantic consistency. This will require modality-aware indexes, richer metadata, learned representations, and new planning mechanisms that can decide not only what to retrieve but also how retrieved artifacts can be transformed into a common analytical form. Natural-language interfaces may become especially important here because users will often express goals at the level of analysis rather than at the level of modality-specific operations.

7.2.5 Agentic Data Discovery

Recent advances in large language models enable a shift from query-based discovery to goal-driven, agentic discovery workflows. Instead of issuing isolated queries, users express high-level analytical goals, and the system decomposes them into sequences of discovery operations such as search, join detection, and augmentation.

A central challenge is inferring information needs from underspecified queries (Kennedy et al. 2022). This requires combining semantic interpretation

with structural reasoning over datasets and their relationships. Recent work on LLM-based data integration and table reasoning suggests that models can assist in identifying relevant datasets and plausible joins but must be grounded in indexing and execution frameworks for scalability (Narayan et al. 2022; Freire et al. 2025).

Agentic discovery is inherently incremental and interactive. Systems must refine results based on user feedback and intermediate outcomes, maintaining context across steps rather than treating discovery as a stateless operation. This suggests a shift toward maintaining stateful discovery sessions and orchestrating multistep workflows.

From a systems perspective, agentic discovery requires combining fast candidate generation with selective use of expensive reasoning components. Indexes provide efficiency and coverage, while language models contribute semantic understanding and explanation. The key challenge is coordinating these components under latency and cost constraints.

References

Behme L, Badura E, Geißler L, Boehm M, Abedjan Z, Markl V (2026) FedAugment: Table augmentation search over decentralized data repositories. Proceedings of the VLDB Endowment 19(10):2672–2685, URL https://doi.org/10.14778/3828612.3828623

Bhardwaj A, Deshpande A, Elmore AJ, Karger D, Madden S, Parameswaran A, Subramanyam H, Wu E, Zhang R (2015) Collaborative data analytics with datahub. Proceedings of the VLDB Endowment (PVLDB) 8(12):1916–1919, URL https://doi.org/10.14778/2824032.2824100

Bleifuß T, Johnson T, Kalashnikov DV, Naumann F, Shkapenyuk V, Srivastava D (2017) Enabling change exploration: Vision paper. In: Proceedings of the ExploreDB, Association for Computing Machinery, New York, NY, USA, ExploreDB'17, URL https://doi.org/10.1145/3077331.3077340

Bleifuß T, Bornemann L, Johnson T, Kalashnikov DV, Naumann F, Srivastava D (2018) Exploring change: a new dimension of data analytics. Proceedings of the VLDB Endowment (PVLDB) 12(2):85–98, URL https://doi.org/10.14778/3282495.3282496

Broder AZ (1997) On the resemblance and containment of documents. In: Proceedings of the Compression and Complexity of Sequences, pp 21–29, https://doi.org/10.1109/SEQUEN.1997.666900

Castelo S, Rampin R, Santos ASR, Bessa A, Chirigati F, Freire J (2021) Auctus: A dataset search engine for data discovery and augmentation. Proceedings of the VLDB Endowment (PVLDB) 14(12):2791–2794, URL http://www.vldb.org/pvldb/vol14/p2791-castelo.pdf

Eltabakh MY, Kunjir M, Elmagarmid AK, Ahmad MS (2023) Cross modal data discovery over structured and unstructured data lakes. Proceedings of the VLDB Endowment (PVLDB) 16(11):3377–3390, URL https://doi.org/10.14778/3611479.3611533

Esmailoghli M, Weidlich M (2025) The past still matters: A temporally-valid data discovery system. CoRR abs/2510.13662, URL https://doi.org/10.48550/arXiv.2510.13662

Esmailoghli M, Schnell C, Miller RJ, Abedjan Z (2025) BLEND: A Unified Data Discovery System . In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE Computer Society, Los Alamitos, CA, USA, pp 737–750, https://doi.org/10.1109/ICDE65448.2025.00061, URL https://doi.ieeecomputersociety.org/10.1109/ICDE65448.2025.00061

Fernandez RC (2023) Data-sharing markets: Model, protocol, and algorithms to incentivize the formation of data-sharing consortia. Proceedings of the ACM on Managanagement of Data (PACMMOD) 1(2):172:1–172:25, URL https://doi.org/10.1145/3589317

Fernandez RC, Abedjan Z, Koko F, Yuan G, Madden S, Stonebraker M (2018) Aurum: A data discovery system. In: Proceedings of the International Conference on Data Engineering (ICDE), IEEE Computer Society, pp 1001–1012, URL https://doi.org/10.1109/ICDE.2018.00094

Freire J, Fan G, Feuer B, Koutras C, Liu Y, Peña E, Santos A, Silva CT, Wu E (2025) Large language models for data discovery and integration: Challenges and opportunities. IEEE Data Engineering Bulletin

Halevy AY, Franklin MJ, Maier D (2006) Dataspaces: A new abstraction for information management. In: International Conference on Database Systems for Advanced Applications, Springer, pp 1–2

Indyk P, Motwani R (1998) Approximate nearest neighbors: Towards removing the curse of dimensionality. STOC

Kennedy J, Subramaniam P, Galhotra S, Fernandez RC (2022) Revisiting online data markets in 2022: A seller and buyer perspective. SIGMOD Rec 51(3):30–37, URL https://doi.org/10.1145/3572751.3572757

Narayan A, et al. (2022) Can foundation models wrangle your data? VLDB

Radford A, Kim JW, Hallacy C, Ramesh A, Goh G, Agarwal S, Sastry G, Askell A, Mishkin P, Clark J, Krueger G, Sutskever I (2021) Learning transferable visual models from natural language supervision. In: Meila M, Zhang T (eds) Proceedings of the International Conference on Machine Learning (ICML), PMLR, Proceedings of Machine Learning Research, pp 8748–8763, URL http://proceedings.mlr.press/v139/radford21a.html

Song Z, Galhotra S, Mehnaz S (2025) Benchmarking robust aggregation in decentralized gradient marketplaces. arXiv preprint arXiv:250905833

Van Breugel B, Van Der Schaar M (2024) Position: why tabular foundation models should be a research priority. In: Proceedings of the International Conference on Machine Learning (ICML), JMLR.org

Wang T, Li F, Zhu L, Li J, Zhang Z, Shen HT (2024) Cross-modal retrieval: A systematic review of methods and future directions. Proceedings of the IEEE 112(11):1716–1754, https://doi.org/10.1109/JPROC.2024.3525147