INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2025IPPAT046

Thèse de doctorat

# Mining Patterns on Tabular Data

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom Paris

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Palaiseau, le 5 Décembre 2025, par

**FRANÇOIS AMAT**

Composition du Jury :

| | |
|---|---|
| Fatiha Sais<br>Professeure, Université Paris-Saclay | Rapporteur |
| Pierre Senellart<br>Professeur, École normale supérieure | Rapporteur |
| Bernd Amann<br>Professeur, Sorbonne Université | Examinateur |
| Nathalie Pernelle<br>Professeure, Université Sorbonne Paris Nord | Examinatrice |
| Fabian Suchanek<br>Professeur, Télécom Paris | Directeur de thèse |
| Pierre-Henri Paris<br>Maître de conférences, Université Paris-Saclay | Co-directeur de thèse |

# Acknowledgments

Conceived during my time in industry, this project reflects a personal commitment to advance the state of the art. Rather than "just implementing," I chose to interrogate prevailing assumptions, pursue a rigorous formalization, and build systems that endure.

I am grateful to my advisors, **Fabian** and **Pierre-Henri**, for demanding yet trusting guidance. Beyond scientific excellence, they met with me *every week* and stood by me through *every difficult moment*; their steady support made this dissertation possible. From **Fabian**, I learned to insist on clarity, falsifiability, and careful semantics; from **Pierre-Henri**, I learned systems thinking, pragmatic judgment, and how to code properly, clean, maintainable, and tested—as well as how to turn ideas into working prototypes. They balanced freedom with focus, challenged every claim to precise evidence, and returned drafts with timely, actionable feedback: *always faster than I could iterate*. They encouraged ambitious questions while steering the work back to essential contributions, modeling integrity, patience, and kindness throughout. Their advice and introductions shaped collaborations, sharpened the evaluation protocol, and ultimately strengthened both MATILDA and MAHILDA.

I thank **Paolo** for timely advice on strategic choices, what to prioritize, when to cut scope, and how to frame the contributions. I am equally grateful to the **DIG team** for welcoming me, for candid discussions and board games after lunch, and for their kindness even on the day I accidentally brought down the servers (*oops*). Working with you made the research both sharper and more enjoyable.

This work was supported by **Hi! PARIS**. I am grateful not only for the grant, but also for the industry-focused conferences and meetups they organized with corporate partners, which provided concrete use cases, candid feedback, and opportunities to align the research with real-world constraints.

To the friends I made during the PhD: thank you for the day to day camaraderie: whiteboard debates that sharpened ideas, emergency debugging that rescued deadlines, coffee fueled mornings, late night rehearsal runs before talks, and gentle reminders to laugh when experiments failed. You turned long weeks into a community and made the hard parts livable.

To friends beyond academia: thank you for balance and perspective: checking in when I disappeared into the work, asking the questions that matter outside the bubble, pulling me outside when I needed air, and insisting on craft over cleverness. Your distance gave clarity.

To my family: thank you for unconditional trust: patient calls, practical help, and financial support when it mattered most; faith on the days I could not find it; celebrations of small wins; and the quiet conviction that setbacks are temporary. This dissertation rests on your steadiness.

To everyone who turns ideas into well-built, useful systems: thank you.

# Abstract

Relational databases remain the backbone of contemporary information systems. Schemas, keys, and declared constraints provide a disciplined vocabulary for storage and querying, yet a substantial portion of the regularity that governs how data are produced, linked, and consumed arises from evolving business processes and application logic that live outside the database management system (DBMS). Because such logic changes faster than schemas, many regularities, conditional co-occurrence across tables, workflow-induced implications, and organization-specific business rules, are only partially reflected in the database design. Rendering these latent, cross-table dependencies explicit is important for data quality, regulatory assurance and auditability, safe integration at scale, and dependable analytics.

This thesis adopts a symbolic, explainable approach that complements black-box learning by producing rule-shaped, verifiable artifacts. In practice, faithful explanations, feature attributions, counterfactuals, rule-based summaries, and causal checks, support debugging, bias and drift detection, data-quality triage, and model-risk governance without exposing proprietary internals. Explainable, rule-shaped outputs do not replace black boxes; they make them safer to deploy, easier to improve, and easier to integrate with the processes that actually run the enterprise. These contributions complement large language models by adding schema-grounded, rule-based structure that improves consistency, provenance, and auditability. They enable fact completion, generation of rule-consistent and counterfactual examples for training and testing, and acceptance checks and guardrails in retrieval-augmented pipelines; the same rules also support validation, imputation, and audit-ready documentation.

The central contribution of this thesis is a deterministic mining framework that operates on any relational database and is capable of discovering both first-order tuple-generating dependencies (with MATILDA) and multi-relation Horn rules (with MAHILDA). Unlike heuristic or black-box approaches, the frameworks guarantee reproducible results by grounding evidence, support, and confidence directly in database semantics. They generalize beyond declared constraints to capture complex cross-table implications, while remaining tractable through schema-guided search and principled pruning. In doing so, they provide a unified, auditable mechanism for surfacing hidden dependencies that are expressive enough to model realistic business logic yet structured enough to integrate with formal reasoning and compliance pipelines.

The scientific problem of this thesis is to mine, from any given relational database, as many meaningful dependencies as possible while prioritizing the most expressive ones, i.e., dependencies that subsume rich families of sub-rules, within finite, predictable time. The challenges are to separate genuine structure from artifacts (e.g., vacuous self-evidence or overlapping derivations),

balance coverage against redundancy among near-equivalent rules, and trade off expressiveness, quantity, and interpretability. We operationalize complexity so that "more expressive" is measured consistently across schemas, normalize scores to compare heterogeneous datasets of different scale and connectivity, and ground significance so that mined dependencies reflect real regularities rather than noise or sampling quirks. The outcome is a compact, non-redundant catalogue of cross-table dependencies with clear evidential grounding, suitable for downstream reasoning, governance, and integration.

# Contents

# Chapter 1

# Introduction

The relational database is one of the most enduring and widely used abstractions for managing data. In its simplest form, a relational database consists of a set of *tables* (also called relations) where each table has a name, a fixed set of columns, and an arbitrary number of rows. Each row represents a record, and each column corresponds to an attribute drawn from a specific domain. The columns of a table enforce a common structure on the rows, while *keys* uniquely identify records and allow tables to be linked. A typical relational database management system (DBMS) provides declarative query languages (e.g. SQL), indexing, schema design, and optimization to enable efficient retrieval and updates. As a result, relational databases make working with structured data more efficient and dependable for developers, data analysts and IT teams [1, 2, 3, 4].

Relational DBMSs remain by far the most prevalent data management technology. Public popularity rankings compiled by DB-Engines list the top database systems by an aggregate measure of community activity, job offers and public interest [5]. At the time of writing, the top four positions are held by Oracle, MySQL, Microsoft SQL Server and PostgreSQL, all of which are relational systems [6, 7]. Such dominance is reflected in industry practice: organisations adopt database technology to store, retrieve and analyse ever larger volumes of data, integrate multiple applications and support transactions with high reliability. Standard references and benchmark suites note that DBMS solutions provide robust functionality for storage, analysis and security, simplify integration with modern analytics, and deliver operational efficiency [3, 2, 8, 9].

## 1.1 Relational Data and Patterns

Most enterprise databases are not a single table but a constellation of inter-related tables. These tables encode real-world entities and their relationships; for example, in an e-commerce application one might find *Users*, *Orders* and *OrderItems*, with foreign-key links from orders back to users and from order items to products. Even in academic settings, benchmark schemas involve multiple tables with keys and joins. For instance, our running example (Figure 1.1) uses three relations: [1] ***MARRIAGE***, recording pairs of spouses; ***LINEAGE***, recording parent–child pairs; and ***RESIDENCE***, mapping people to cities and states. Tables are populated with a handful of rows,

[1] This toy schema is inspired by one of the database from the CTU Prague Relational Learning Repository[10], which hosts many relational datasets for research.

**Table MARRIAGE**

| **Partner1** | **Partner2** |
|---|---|
| Elvis | Priscilla |
| Priscilla | Elvis |
| Lisa | Danny |
| Danny | Lisa |

**Table LINEAGE**

| **Parent** | **Child** |
|---|---|
| Priscilla | Lisa |
| Elvis | Lisa |
| Lisa | Riley |
| Danny | Riley |

**Table RESIDENCE**

| **Person** | **Location** | **State** |
|---|---|---|
| Elvis | Memphis | TN |
| Priscilla | Memphis | TN |
| Lisa | Beverly Hills | CA |

**Figure 1.1:** Toy database used in examples: **MARRIAGE**, **LINEAGE**, and **RESIDENCE**.

*Elvis* Presley married *Priscilla*; *Lisa* is the child of *Elvis* and *Priscilla*; and so on, but the relational structure is evident.

When multiple tables interact, interesting regularities often appear across them. For example, in our running example we may observe that most people who appear as parents in `LINEAGE` also have a corresponding entry in `RESIDENCE`. Similarly, married couples tend to reside in the same city. Such observations are examples of *patterns*: recurring combinations of tuples that reflect underlying business rules, social norms or domain knowledge. Patterns can be expressed as logical implications that relate one set of facts (the body) to another set (the head). In the above example, "parents typically have a residence" can be formalised as:

$$\textsc{Lineage}(p, c) \;\Rightarrow\; \exists l, s \;:\; \textsc{Residence}(p, l, s),$$

which says that if $p$ appears as a parent of some child $c$, then there must be a city $l$ and state $s$ such that $p$ resides in $l$ and $s$. Patterns of this kind are valuable: they capture implicit assumptions that may not be encoded as database constraints, they provide insights into the data generating process, and they can reveal data quality issues when violated.

In many organisations, a gap exists between what the schema declares (e.g., primary keys and foreign keys) and the full set of rules that actually hold in the data. Hidden rules may govern compliance, for instance, in HR databases, any employee marked active must be covered by exactly one non-overlapping contract interval that includes today; in healthcare data, every prescription should be justified by at least one diagnosis from the same encounter, and any lab result flagged critical must be accompanied by a physician-notification record within a defined time bound. These

constraints are not captured by basic key/foreign-key structure, yet they materially affect data quality, auditability, and downstream analytics.

To illustrate, suppose a financial institution stores customer accounts, transactions and risk assessments in separate tables. Domain knowledge dictates that a large transaction must trigger both a risk assessment and an approval by a compliance officer. If the database contains a transaction without these accompanying records, we have found a violation that may require investigation. Conversely, mining the data may surface that every recorded large transaction indeed has the expected ancillary records, giving evidence of good compliance. Similar examples abound in healthcare (patient consent and audit trails), logistics (shipment requests and pick lists) and accounting (orders, payments, ledger entries and reconciliations). The common thread is that multi-table patterns capture relationships that transcend individual tables.

## 1.2 Towards Automatic Pattern Discovery

Discovering useful patterns by manual inspection is labour-intensive and does not scale: modern databases may contain dozens or hundreds of tables and millions of records. Automated pattern discovery aims to close this gap by mining relational data for logical implications that hold with high support and confidence. In this thesis we are interested in two classes of rules:

- **Tuple-generating dependencies (TGDs)** are first-order formulas of the form $\forall \vec{x}, \vec{y} : \phi(\vec{x}, \vec{y}) \Rightarrow \exists \vec{z} : \psi(\vec{x}, \vec{z})$, where $\phi$ (the body) and $\psi$ (the head) are conjunctions of atoms. TGDs allow multiple atoms in the head and existential variables, making them expressive enough to capture the multi-table patterns described above.

- **Horn rules** (or single-head TGDs) have exactly one atom in the head. These rules are widely studied in inductive logic programming and knowledge-graph completion. While less expressive than general TGDs, they are easier to mine and interpret.

These two families mark the expressiveness–tractability frontier relevant to multi-relational discovery. Our primary focus is on *first-order TGDs* with multi-atom heads and existential witnesses, which natively capture the cross-table regularities that motivate this thesis. Prior to this work, no practical, off-the-shelf systems mined such FO-TGDs from relational data; accordingly, we include *Horn rules* chiefly as a comparative baseline, leveraging mature ILP/knowledge-graph tooling to contextualise the costs and benefits of the added expressiveness.

From a practical perspective, a mined rule should be both *frequent* (it applies to many tuples) and *reliable* (it is rarely violated). Measures such as *support*, the fraction of head instances that satisfy the rule, and *confidence*, the fraction of body instances that satisfy the rule, quantify these desiderata. Mining algorithms must search the vast space of possible rules, prune unpromising candidates and handle noise and incompleteness in the data. The challenges are amplified when rules span multiple tables and contain existential variables.

## 1.3 Positioning with Respect to the State of the Art

There are different families of patterns in databases, each with its own aims, evidence notions, and mining toolkits.

*Integrity constraints* capture invariants that should hold in all admissible instances. Typical examples are functional dependencies (FDs), inclusion dependencies (the values of a set of attributes in one relation must be contained in those of another, generalizing foreign keys), conditional functional dependencies (CFDs), and denial constraints. Such patterns are commonly mined by schema-level discovery algorithms (e.g., level-wise or partition-based methods for FDs; inclusion-dependency miners that exploit sorted projections and merge-scans; DC discovery in Metanome-style pipelines). Their strength lies in crisp, auditable semantics and direct utility for data cleaning and integration, but they do not natively express multi-atom heads or recursion, and they typically do not come with support/coverage/confidence measures aligned with rule discovery.

*Correlation and co-occurrence patterns* summarize which attribute values or events tend to appear together. These include frequent itemsets and association rules (support, confidence, lift), sequential/episode patterns in temporal data, and frequent subgraphs in networked data. Such patterns are commonly mined by algorithms like Apriori/FP-Growth for itemsets, PrefixSpan for sequences, and gSpan-style approaches for graphs. They are effective at surfacing regularities and are robust to noise via frequency thresholds, but they do not encode first-order implication structure over joins, and they lack the database-native counting needed to justify multi-relation logical conclusions.

*Relational rule patterns* express implications between predicates with variables, usually without existential witnesses. In ILP and knowledge-graph mining, Horn rules with a single head atom are learned by systems such as FOIL/Progol/Aleph/Popper (supervised consistency-driven search) or AMIE-style miners (support/confidence/PCA confidence on binary facts). These approaches excel at deriving universally quantified implications and can incorporate background knowledge, but they typically assume binary predicates or KB triples, do not natively introduce new existential witnesses, and often rely on open-world or heuristic evidence semantics that do not transfer to n-ary relational schemas with multi-table joins.

*Probabilistic and causal pattern formalisms* (e.g., Markov logic, Bayesian networks, causal graphs) model uncertainty or directed dependence and are commonly learned by score-based or constraint-based structure search. They provide graded beliefs and interventions rather than auditable first-order TGDs grounded in relational counting; their objectives and evidence notions are therefore different from coverage/confidence for rule discovery.

This thesis closes the above gaps by mining first-order TGDs directly over relational schemas with principled, database-native evidence metrics; by extending to recursion with stabilized counting and disjoint semantics; and by constructing a fair Horn baseline to delineate when simpler single-head rules suffice versus when added expressiveness (multi-atom heads, existential witnesses, recursion) is necessary.

## 1.4 Contributions of this Thesis

This dissertation investigates how to automatically discover cross-table patterns in relational databases and evaluates the benefits and limitations of expressive rule mining. We make two main contributions:

1. **Expressive first-order TGDs (including recursion).** We develop algorithms to discover tuple-generating dependencies with multi-atom heads and existential witnesses directly from relational data, and extend them to *recursive* TGDs where a predicate may reappear across body and head. Our framework balances expressiveness and tractability by (i) restricting bodies to small, joinable patterns captured by a constraint graph; (ii) employing duplicate-free enumeration with canonicalization; and (iii) computing auditable support and confidence that remain stable under joins and existential witnesses. This contribution appears in Chapter 4 (*MATILDA*, manuscript under review [11]).

2. **A relational Horn-rule baseline.** We adapt and extend techniques from inductive logic programming and knowledge-graph completion to multi-table relational schemas, yielding a fair, apples-to-apples Horn baseline. Our system translates n-ary schemas into a form amenable to Horn miners, and we evaluate how well single-head rules approximate more expressive TGDs, identifying regimes where Horn rules suffice. To prevent self-witnessing inflation in recursive settings, we introduce *disjoint semantics* .This contribution is developed in Chapter 5 (*MAHILDA*, manuscript ready for submission [12]).

**Research Questions.** Mining expressive dependencies directly over multi-table schemas promises actionable structure (e.g., co-movement across relations, shared witnesses, and inductive patterns), but it raises intertwined challenges. We must first establish whether such rules can be mined on realistic schemas without prohibitive cost. Next, because existential and multi-atom heads complicate evidence, we need auditable quality notions that behave well under joins, duplication, noise, and missingness. Effective discovery further hinges on search spaces and pruning that preserve completeness and interpretability while avoiding duplicates. Recursion offers genuine modeling value but risks vacuity unless counting semantics prevent self-witnessing. Finally, practitioners need guidance on when simpler Horn rules suffice and when additional expressiveness is warranted. This motivates the following questions:

**RQ1:** Is it computationally feasible to mine FO-TGDs with connected bodies and multi-atom heads on realistic schemas?

**RQ2:** Which notions of support and confidence are appropriate for FO-TGDs with existential variables, and how do these measures behave under joins, duplication, noise, and missingness?

**RQ3:** What search spaces and pruning principles enable effective discovery without sacrificing completeness or interpretability, and how can duplicates be characterised and avoided?

**RQ4:** In what settings do recursive dependencies arise and add material value, and what evaluation/counting semantics prevent trivial or self-justifying rules?

**RQ5:** How do single-head Horn rules compare to FO-TGDs empirically and in practice, and under which regimes is additional expressiveness (multi-head, existential, recursive) necessary?

## 1.5 Structure of the Thesis

This thesis is organised as follows. Chapter 2 (Preliminaries) introduces the relational data setting, the logical notation used throughout, and the evidence metrics of support and confidence that ground our evaluation. Chapter 3 (An Overview of Pattern Mining) surveys four families of approaches: (i) constraint and dependency mining; (ii) existential rule discovery; (iii) knowledge-graph miners and inductive logic programming; and (iv) probabilistic and representation-learning hybrids. It positions *MATILDA* within this landscape and closes with a high-level comparison to representative systems. Chapter 4 (Mining TGDs on Tabular Data) presents the core definitions and the *constraint-graph* abstraction, details the *MATILDA* algorithm together with its theoretical analysis, and extends the framework to recursive TGDs; it also reports optimisations and ablation studies, including search strategies and runtime analyses. Chapter 5 (Mining Horn Rules with Disjoint Semantics) develops a Horn-rule baseline tailored to relational data, uses *MATILDA*'s rule metrics and *disjoint semantics*, describes the *MAHILDA* design, and provides an empirical evaluation. The General Conclusion summarises the main findings, discusses limitations, and outlines directions for future work. Appendix A documents the datasets used, with schema snippets and basic statistics, and Appendix B offers a French-language summary of the problem, contributions, evaluation, impact, and perspectives.

# Chapter 2

# Preliminaries

In order to formalize the problem of rule mining, we introduce the basic concepts and notations used throughout the thesis. Readers familiar with relational databases and logic may skip to Chapter 4.

## 2.1 Relational Databases

We recall the definition of a relational database following standard references such as [1, 13]. A relational database consists of a finite set of named *tables*. Each table $T$ has an ordered list of attribute names $A(T) = \{a_1, \ldots, a_n\}$ with corresponding domains $D_1, \ldots, D_n$. A tuple (row) in $T$ is an element of the Cartesian product $D_1 \times \cdots \times D_n$. We write $T(v_1, \ldots, v_n)$ to denote that the tuple $\langle v_1, \ldots, v_n \rangle$ belongs to $T$. Attributes are qualified by their relation name (e.g., $T.a$); when unambiguous, we freely write $a$. For a tuple $t$ in $T$, we denote its $i$th component by $a_i(t) = v_i$. Keys and foreign keys impose uniqueness and referential constraints but are not discussed further here.

> **Example 2.1.1.** *In the toy instance of Figure 1.1, the table* Lineage *has two attributes, Parent and Child. For example,* Lineage(*Elvis*, *Lisa*) *is a stored tuple. The attribute-projection notation* Lineage.Parent = {*Elvis*, *Priscilla*, *Lisa*, *Danny*} *denotes all parent names that occur in the table.*

We work under a closed-world, active-domain setting: instances are fully materialised and any fact not present is treated as false; variable assignments range over values that occur in the instance. Distinct constants denote distinct objects (unique-name assumption) and there are no nulls or Skolem terms unless stated otherwise. The vocabulary is purely relational: relation symbols appear in small caps (e.g., Residence); variables are italic ($x$, $y$, $z$); and constants appear in italics (e.g., *Elvis*). We use no function symbols, and built-in predicates, except =, such as $\neq$, or $<$ are disallowed unless explicitly enabled later. Rules are range-restricted (safe): every universally quantified variable must occur in the body of its rule. Unless noted otherwise, all counts use set semantics over distinct bindings of the free tuple $\vec{x}$ (introduced in Definitions 2.2.1–2.2.3 below); multiplicities are ignored.

**Definition 2.1.1** (Tuple-Generating Dependency)**.** *A* tuple-generating dependency *(TGD) is a first-order sentence*

$$\forall \vec{x}, \vec{y} \colon \ \phi(\vec{x}, \vec{y}) \ \Rightarrow \ \exists \vec{z} \colon \ \psi(\vec{x}, \vec{z}),$$

*where $\phi$ (the* body*) and $\psi$ (the* head*) are finite conjunctions of relational atoms of the form $t(x_1, \ldots, x_n)$, one per table $t$. Variables $\vec{x}$ appear in both body and head; variables $\vec{y}$ appear only in the body; variables $\vec{z}$ appear only in the head. The head is non-empty.*

Intuitively, a TGD says that whenever the body holds for some assignment, there must exist *witnesses* (values for $\vec{z}$) that make the head true.

**Definition 2.1.2** (Non-Recursive Tuple-Generating Dependency)**.** *A TGD is* non-recursive *if each relation appears at most once in the entire rule (counting all occurrences across body and head together).*

**Definition 2.1.3** (Recursive Tuple-Generating Dependency)**.** *A TGD is* recursive *if at least one relation appears more than once in the rule. A relation may appear multiple times in the body, multiple times in the head, or across both body and head. Throughout the thesis, we focus on connected TGDs: disconnected rules decompose into independent subrules, tend to express weak or accidental co-occurrences, and can artificially inflate counts without adding semantic cohesion.*

**Example 2.1.2.** *"Every parent has a residence" can be written as*

$$\forall p, c \colon \ \textsc{Lineage}(p, c) \ \Rightarrow \ \exists \ell, s \colon \ \textsc{Residence}(p, \ell, s).$$

*Here, $p$ is shared between body and head, $c$ appears only in the body, and $\ell, s$ are existential head variables.*

*Remark (multi-head semantics).* When a TGD has multiple head atoms, the existential witnesses $\vec{z}$ are *shared across all head atoms*. This coupling is stronger than holding each head atom separately and is central to the expressiveness we target.

**Canonicalisation and subsumption.** Two rules are *isomorphic* if one can be obtained from the other by a renaming of variables ($\alpha$-equivalence). Rule $R_1$ *subsumes* $R_2$ if there exists a substitution $\theta$ such that $\theta(\text{body}(R_1)) \subseteq \text{body}(R_2)$ and $\theta(\text{head}(R_1)) \subseteq \text{head}(R_2)$. These notions underlie duplicate-free generation and pruning used later.

**Vacuous rules and pruning.** Rules like $T(x) \Rightarrow T(x)$ are always satisfied and thus *vacuous*. We avoid such tautologies via structural constraints (connectedness), later *disjoint semantics* , and scoring (Section 2.2).

## 2.2 Quality Metrics: Support and Confidence

**Intuition and motivation.** A rule should be judged along two complementary axes: (i) *How much of what already exists does it explain?* (coverage of the observed head), and (ii) *How reliable is it when it fires?* (success rate over the observed body). We capture these with *support* (a recall-like coverage of the head domain) and *confidence* (a precision-like reliability over the body domain). Both metrics are instance-based, unsupervised, and auditable: they count distinct assignments of the free tuple $\vec{x}$ under closed-world, set semantics.

Given a database instance and a TGD

$$R\colon \forall \vec{x}, \vec{y}\colon\ \phi(\vec{x}, \vec{y}) \ \Rightarrow\ \exists \vec{z}\colon\ \psi(\vec{x}, \vec{z}),$$

we quantify its empirical strength using *support* (head coverage) and *confidence*.

**Definition 2.2.1** (Prediction)**.** *The* prediction *of $R$ is the set of $\vec{x}$-tuples for which both the body and the head hold:*

$$\text{Prediction}(R) \ =\ \left\{ \vec{x} \ \middle|\ \exists \vec{y}, \vec{z}\colon\ \phi(\vec{x}, \vec{y}) \wedge \psi(\vec{x}, \vec{z}) \right\}.$$

Intuitively, $\text{Prediction}(R)$ collects the $\vec{x}$-tuples for which the rule both *fires* (the body holds for some $\vec{y}$) and *succeeds* (there exist witnesses $\vec{z}$ that make the whole head true). It sits at the intersection of two natural domains for $\vec{x}$:

$$\text{Evidence}(R) = \{\vec{x} \mid \exists \vec{y}\, \phi(\vec{x}, \vec{y})\} \quad \text{and} \quad \text{Head}(R) = \{\vec{x} \mid \exists \vec{z}\, \psi(\vec{x}, \vec{z})\}.$$

For multi-atom heads, the *same* witnesses $\vec{z}$ must satisfy *all* head atoms simultaneously (not one per atom), so multi-head prediction is not reducible to a conjunction of independent single-head rules.

The two quality scores below normalise the same numerator $|\text{Prediction}(R)|$ by these reference sets: *support* divides by $|\text{Head}(R)|$, while *confidence* divides by $|\text{Evidence}(R)|$.

**Definition 2.2.2** (Support)**.** *The* support *(head coverage) of $R$ is the fraction of head assignments covered by the rule:*

$$\text{Support}(R) \ =\ \frac{|\text{Prediction}(R)|}{\left|\{\vec{x} \mid \exists \vec{z}\colon \psi(\vec{x}, \vec{z})\}\right|}.$$

*This measure coincides with the usual notion of* head coverage *in the knowledge-graph literature.*

Dually, replacing the head-domain denominator with the evidence-domain one yields *confidence*:

**Definition 2.2.3** (Confidence)**.** *The* confidence *of $R$ is the fraction of body assignments that satisfy the rule:*

$$\text{Confidence}(R) \ =\ \frac{|\text{Prediction}(R)|}{\left|\{\vec{x} \mid \exists \vec{y}\colon \phi(\vec{x}, \vec{y})\}\right|}.$$

*If the body $\phi$ is syntactically empty, set* $\text{Confidence}(R) = 1$ *when the head is satisfiable in the instance and* $0$ *otherwise.*

**Counting semantics and edge cases.** All three quantities above are taken over *distinct* $\vec{x}$-tuples (set semantics). If multiple $(\vec{y}, \vec{z})$ witnesses yield the same $\vec{x}$, that $\vec{x}$ contributes only once to Prediction$(R)$. We adopt a closed-world view (absent facts are false). When a denominator is zero, the ratio is undefined; in practice, we discard such rules. For the special case of an empty body, Definition 2.2.3 already handles it explicitly.

Indeed, When $\phi$ is empty, $\{\vec{x} \mid \exists \vec{y}\, \phi(\vec{x}, \vec{y})\} = |\{()\}| = 1$.

**Example 2.2.1.** *Consider* $R: \forall p, c: \textsc{Lineage}(p, c) \Rightarrow \exists s: \textsc{Marriage}(c, s)$. *In Figure 1.1, married people are* $\{Elvis, Priscilla, Lisa, Danny\}$ *(size* 4*); married children are* $\{Lisa\}$ *(size* 1*). Hence* Support$(R) = 1/4 = 0.25$. *The set of children in* Lineage *is* $\{Lisa, Riley\}$ *(size* 2*), of which exactly one is married, so* Confidence$(R) = 1/2 = 0.5$.

One might wonder whether TGDs are needed at all, or whether the same effect can be obtained by combining single-head Horn rules.

**Multi-atom heads do not factor.** If a TGD has several head atoms, its confidence does *not* equal the product of confidences of the corresponding single-head Horn rules. For example, even if $\textsc{Marriage}(x, y) \Rightarrow \textsc{Residence}(x, \ell, s)$ has high confidence and $\textsc{Marriage}(x, y) \wedge \textsc{Residence}(x, \ell, s) \Rightarrow \textsc{Residence}(y, \ell, s)$ has confidence 1, the combined TGD

$$\textsc{Marriage}(x, y) \Rightarrow \exists \ell, s: \textsc{Residence}(x, \ell, s) \wedge \textsc{Residence}(y, \ell, s)$$

may still have lower overall confidence because both head atoms must hold *together* with the *same* witnesses $(\ell, s)$.

**Example 2.2.2** (Multi-head example)**.** *Let* $R': \textsc{Marriage}(x, y) \Rightarrow \exists \ell, s: \textsc{Residence}(x, \ell, s) \wedge \textsc{Residence}(y, \ell, s)$. *Here* $\vec{x} = (x, y)$ *and* $\vec{z} = (\ell, s)$.

$$\text{Prediction}(R') = \left\{(x, y) \;\middle|\; \exists \ell, s: \textsc{Marriage}(x, y) \wedge \textsc{Residence}(x, \ell, s) \wedge \textsc{Residence}(y, \ell, s)\right\}.$$

*A SQL sketch under set semantics (using* `DISTINCT`*) is:*

- *Numerator* |Prediction$(R')$|

```
SELECT COUNT(DISTINCT m.Partner1, m.Partner2)
FROM MARRIAGE m
JOIN RESIDENCE rx ON rx.Person = m.Partner1
JOIN RESIDENCE ry ON ry.Person = m.Partner2
WHERE rx.Location = ry.Location AND rx.State = ry.State;
```

- *Denominator for Support:* $|\{(x, y) \mid \exists \ell, s\, \psi(x, y, \ell, s)\}|$

```
SELECT COUNT(DISTINCT rx.Person, ry.Person)
FROM RESIDENCE rx JOIN RESIDENCE ry
 ON rx.Location = ry.Location AND rx.State = ry.State;
```

- *Denominator for Confidence:* $|\{(x, y) \mid \exists\, body\}| = |\textsc{Marriage}|$

*This makes explicit that the same $(\ell, s)$ must witness both head atoms.*

**Proposition 2.2.1** (Support–confidence symmetry)**.** *Let* $R\colon \forall\vec{x},\vec{y}\colon\ \phi(\vec{x},\vec{y}) \implies \exists\vec{z}\colon\ \psi(\vec{x},\vec{z})$ *where* $\phi$ *is non-empty, and define its* symmetric counterpart $R^{-}\colon \psi(\vec{x},\vec{z}) \Rightarrow \exists\vec{y}\colon \phi(\vec{x},\vec{y})$. *Then* $\mathrm{Support}(R) = \mathrm{Confidence}(R^{-})$ *and* $\mathrm{Confidence}(R) = \mathrm{Support}(R^{-})$.

*Proof.* By Definitions 2.2.2 and 2.2.3 and the fact that $\mathrm{Prediction}(R) = \mathrm{Prediction}(R^{-})$,

$$\mathrm{Confidence}(R) = \frac{|\mathrm{Prediction}(R)|}{|\{\vec{x} \mid \exists\vec{y}\colon \phi(\vec{x},\vec{y})\}|} \quad \text{and} \quad \mathrm{Support}(R^{-}) = \frac{|\mathrm{Prediction}(R^{-})|}{|\{\vec{x} \mid \exists\vec{z}\colon \psi(\vec{x},\vec{z})\}|}.$$

The equalities follow by identifying the common numerator and swapping denominators; the other identity is symmetric. □

## 2.3 Chapter summary

This chapter introduces the basic notions of relational databases and the rule language considered in the thesis. The setting adopts a closed-world assumption over the active domain, unique-name and no-nulls conventions, and set semantics: counts range over distinct bindings of the free tuple $\vec{x}$.

The rule language consists of tuple-generating dependencies of the form $\phi(\vec{x},\vec{y}) \Rightarrow \exists\vec{z}\,\psi(\vec{x},\vec{z})$ with possibly multi-atom heads, where all head atoms share the same existential witnesses $\vec{z}$. Attention is restricted to connected, non-vacuous rules; tautological rules, e.g., those whose head is already contained in the body up to renaming, are excluded.

Given a TGD $R$, the **prediction** set collects those $\vec{x}$ for which both the body and the head hold under the same witnesses. Two instance-level quality measures are used:

$$\mathrm{Support}(R) = \frac{|\mathrm{Prediction}(R)|}{|\{\vec{x} \mid \exists\vec{z}\,\psi(\vec{x},\vec{z})\}|} \quad \text{and} \quad \mathrm{Confidence}(R) = \frac{|\mathrm{Prediction}(R)|}{|\{\vec{x} \mid \exists\vec{y}\,\phi(\vec{x},\vec{y})\}|}.$$

These definitions exhibit a symmetry under swapping body and head. For multi-atom heads, the same witnesses must satisfy all head atoms simultaneously; consequently, confidence does not factor into products of single-head confidences.

The presentation contrasts closed-world, instance-based counting with open-world knowledge-graph settings and supervised ILP pipelines, where evaluation is example-based.

# CHAPTER 3

# An Overview of Pattern Mining

**Guiding questions.** This chapter reviews prior work *around* the research questions stated in Chapter 1: (i) **RQ1** feasibility and tractability of mining expressive first-order TGDs with connected bodies and possibly multi-atom heads on real schemas, together with auditable statistics, and (ii) **RQ2** evidence semantics, that is, which support and confidence notions remain stable under joins and existential variables, how robust they are to incompleteness and noise, and how they can be computed efficiently. Rather than a neutral survey, the selection below is organized to illuminate what parts of the literature already address RQ1 or RQ2, what remains open, and how this motivates our framework.

**Scope and structure.** We first recall dependency families classically used in databases (Figure 3.1 provides a taxonomy of these dependency types), then move to existential rules and fragments with decidable reasoning, then to representative mining systems on relational data and knowledge graphs, and finally to probabilistic and representation-learning hybrids. Throughout, we indicate how each thread bears on RQ1 and RQ2.

## 3.1 Constraint-oriented pattern mining

In this subsection, we introduce the different types of patterns that can be mined on databases.

**Exact dependencies.** *Functional Dependencies (FDs)*, *Equality-Generating Dependencies (EGDs)*, and more general *Embedded Dependencies (EDs)* are foundational constraints that capture, respectively, attribute determinacy, enforced equalities among attributes, and implication patterns expressed as first-order dependencies over relations [14, 13]. *Unique Column Combinations (UCCs)* and (candidate) *keys* identify minimal attribute sets that functionally determine tuple identity and thus uniquely identify records [15, 16]. *Inclusion Dependencies (INDs)* and *foreign keys* extend integrity across relations by requiring that values from one attribute set are contained within the values of another (e.g., referential containment) [17]. *Denial Constraints (DCs)* forbid specific value combinations, possibly across multiple tuples, using built-ins such as (in)equality and comparison predicates; every witness to a forbidden pattern constitutes a concrete violation [18, 19]. *Order Dependencies (ODs)* assert that the lexicographic order induced by one attribute list determines (is consistent with) the order induced by another list, thereby capturing order-preserving relationships that FDs cannot express [20]. *Join Dependencies (JDs)* formalize

multiway decompositions by requiring that a relation equals the natural join of its projections on specified attribute subsets, generalizing multivalued dependencies and underpinning higher normal forms [13]. These constraints yield crisp, tuple-level counterexamples when violated, which directly supports auditability. With respect to **RQ1**, specialized profilers scale well but typically target one dependency family at a time rather than general first-order TGDs. For **RQ2**, FD/IND discovery already leverages frequency and violation counts, yet the prevailing semantics are predominantly single-relation or single-head, whereas our setting extends these ideas to $n$-ary joins and expressive heads while preserving enumerability of evidence.

**Functional Dependencies (FDs).** An FD $X \to Y$ states that equality on attributes in $X$ implies equality on $Y$. Profiling evidence is violation-based (exact or approximate), and FDs underpin keys and normalization.

For relation $R$ and attribute sets $X, Y$, an FD is written $R : X \to Y$. In first-order form as an EGD:

$$\forall t_1, t_2 : R(t_1) \wedge R(t_2) \wedge t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y].$$

In `RESIDENCE(Person, Location, State)` we have the FD Person $\to$ {Location, State}: the same person must live at the same location and state. In the toy data, *Elvis*, *Priscilla*, and *Lisa* each occur once, so the FD is satisfied (no counterexample). Many algorithms can automatically mine FDs from data, such as TANE [21], HyFD [16].

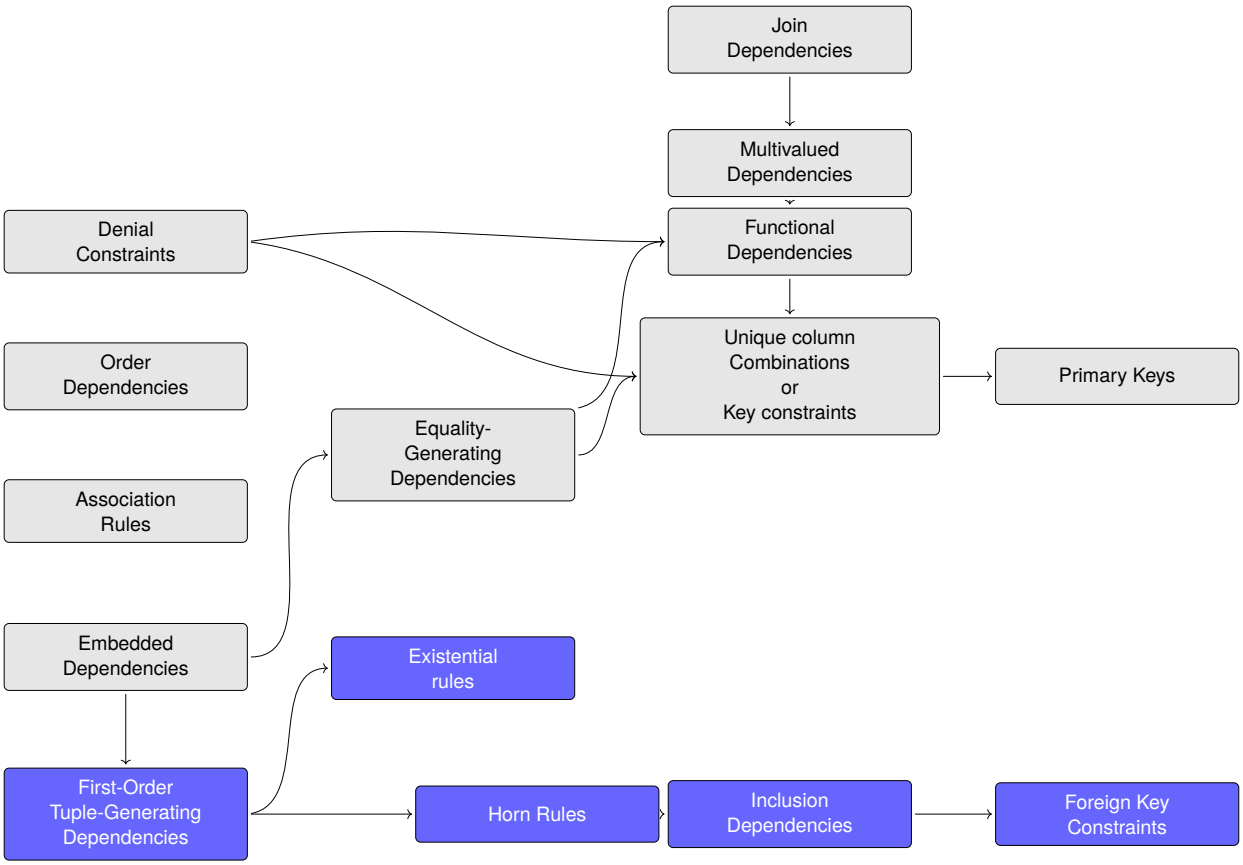


**Figure 3.1:** Taxonomy of dependency types and how TGDs relate to other constraints.

**Equality-Generating Dependencies (EGDs).** EGDs enforce equalities derived from satisfied patterns, e.g., $\phi(\vec{x}) \Rightarrow u = v$. They subsume FDs and are typically assessed via counterexample detection.

An EGD has the general form $\forall \vec{x} : \ \phi(\vec{x}) \Rightarrow u = v$, where $u, v$ are variables among $\vec{x}$.

The FD Person $\rightarrow$ State on RESIDENCE can be encoded as the EGD:

$$\forall p, \ell_1, \ell_2, s_1, s_2 : \ \texttt{RESIDENCE}(p, \ell_1, s_1) \wedge \texttt{RESIDENCE}(p, \ell_2, s_2) \ \Rightarrow \ s_1 = s_2.$$

In the toy data there is no person with two different states, so the EGD holds. In practice, EGDs (often keys) are usually provided by users and validated via the chase in integration systems (e.g., Clio [22]) rather than mined directly.

**Embedded Dependencies (EDs).** EDs combine TGDs and EGDs into a unifying logical framework that subsumes many classical constraints. Direct mining of full EDs is uncommon; practitioners target their fragments.

An ED has the shape $\forall \vec{x} : \ \phi(\vec{x}) \Rightarrow \exists \vec{z} : \ \psi(\vec{x}, \vec{z})$, where $\phi, \psi$ are conjunctions of relational atoms; equalities may also appear.

An illustration is "co-residence of spouses in the same state" as an ED:

$$\forall x, y : \ \texttt{MARRIAGE}(x, y) \ \Rightarrow \ \exists \ell, s : \ \texttt{RESIDENCE}(x, \ell, s) \wedge \texttt{RESIDENCE}(y, \ell, s).$$

For the pair (*Elvis*, *Priscilla*), a witnessing assignment is $\ell$ =*Memphis*, $s$ =*TN*, which is satisfied by the toy data.

Schema mapping tools (e.g., Clio [22]) rely on EDs specified by experts to ensure data consistency across sources, but no general ED miner exists.

**Unique Column Combinations (UCCs) and Keys.** UCCs are attribute sets that uniquely identify tuples; minimal UCCs are candidate keys. Discovery checks uniqueness (often with external-memory runs) and informs schema design.

$X$ is a UCC of relation $R$ iff:

$$\forall t_1 \neq t_2 : \ R(t_1) \wedge R(t_2) \ \Rightarrow \ t_1[X] \neq t_2[X].$$

Equivalently, the FD $X \rightarrow \mathsf{sch}(R)$ holds.

In RESIDENCE, $X$ = {Person} is a UCC (and key): no two distinct rows share the same *Person*. In MARRIAGE, {Partner1, Partner2} is a UCC for ordered pairs; the dataset contains both (Elvis, Priscilla) and (Priscilla, Elvis), which are distinct ordered pairs and each appears once.

Modern profiling platforms (e.g., Metanome) include dedicated algorithms to find UCCs; for instance, the Ducc algorithm [23] efficiently enumerates all minimal unique column combinations in large datasets.

**Inclusion Dependencies (INDs) and Foreign Keys.** An IND $R[X] \subseteq S[Y]$ asserts projection-level containment across relations and underlies foreign keys. Profilers validate containment via merge-scans or indexes; approximate INDs bound violation rates.

Unary IND between attributes $X$ of $R$ and $Y$ of $S$:

$$\pi_X(R) \ \subseteq \ \pi_Y(S).$$

Concrete instances: $\pi_{\text{Partner1}}(\texttt{MARRIAGE}) \subseteq \pi_{\text{Parent}}(\texttt{LINEAGE})$ holds since every Partner1 value (Elvis, Priscilla, Lisa, Danny) appears among LINEAGE.Parent. By contrast, $\pi_{\text{Child}}(\texttt{LINEAGE}) \subseteq$

$\pi_{\text{Person}}(\texttt{RESIDENCE})$ fails because *Riley* has no RESIDENCE row; similarly, $\pi_{\text{Partner1}}(\texttt{MARRIAGE}) \subseteq \pi_{\text{Person}}(\texttt{RESIDENCE})$ fails due to *Danny*. Specialized algorithms such as SPIDER [24](from Metanome) leverage sorting and pruning to efficiently discover INDs even on wide schemas.

**Denial Constraints (DCs).** DCs state that certain value combinations must not occur, often with built-ins ($\neq, <$, etc.). They capture data quality rules and are evaluated by searching for counterexample tuples.

A DC is a universally quantified negation: $\forall \vec{x} : \neg\phi(\vec{x})$, equivalently $\forall \vec{x} : \ \phi(\vec{x}) \Rightarrow \bot$.

No self-marriage:

$$\forall x : \ \texttt{MARRIAGE}(x, x) \Rightarrow \bot \quad \text{(satisfied in the toy data).}$$

Single state per person:

$$\forall p, \ell_1, \ell_2, s_1, s_2 : \ \texttt{RESIDENCE}(p, \ell_1, s_1) \wedge \texttt{RESIDENCE}(p, \ell_2, s_2) \wedge s_1 \neq s_2 \Rightarrow \bot.$$

**Order Dependencies (ODs).** ODs express that ordering by $X$ implies ordering by $Y$. They are useful for optimization and quality checks; evidence is again violation-based rather than probabilistic.

For relation $R$ and lists of attributes $\vec{X}, \vec{Y}$, an OD $\vec{X} \rightsquigarrow \vec{Y}$ means:

$$\forall t_1, t_2 : \ t_1[\vec{X}] \leq_{\text{lex}} t_2[\vec{X}] \ \Rightarrow \ t_1[\vec{Y}] \leq_{\text{lex}} t_2[\vec{Y}].$$

On RESIDENCE, the trivial OD [Person] $\rightsquigarrow$ [Person] always holds. A non-trivial check: with the two cities *Beverly Hills* and *Memphis*, ordering by [Location] yields states in order [CA, TN], so [Location] $\rightsquigarrow$ [State] is satisfied on the toy rows.

Techniques now exist to mine ODs from data; for example, Szlichta et al. [25] present a sound and complete axiomatization along with an algorithm that finds all valid ODs by leveraging set-based inference.

**Join Dependencies (JDs).** JDs characterize lossless decompositions of a relation into projections. While central in normalization theory, mining full JDs is rare; special cases inform schema refactoring.

A JD $*(X_1, \ldots, X_k)$ on relation $R$ states that $R = \pi_{X_1}(R) \bowtie \cdots \bowtie \pi_{X_k}(R)$.

Because RESIDENCE has key {Person}, the decomposition into $X_1 = \{\text{Person}, \text{Location}\}$ and $X_2 = \{\text{Person}, \text{State}\}$ is lossless:

$$\texttt{RESIDENCE} = \pi_{\{P,L\}}(\texttt{RESIDENCE}) \ \bowtie \ \pi_{\{P,S\}}(\texttt{RESIDENCE}) \ (\text{on } P = \text{Person}).$$

Rejoining the two projections by *Person* reconstructs the original rows for Elvis, Priscilla, and Lisa. In practice, no general JD miners exist; designers rely on theoretical algorithms and schema design principles. Lossless join decompositions are typically validated via the schema's FDs or by using the chase test for join dependencies [26], rather than discovered through brute-force data profiling.

**Approximate and quality-aware dependencies.** Real data are noisy. Approximate FDs and Conditional FDs tolerate controlled error rates and report violations [27, 28, 19]. Approximate INDs and DCs similarly trade exactness for usefulness [18, 17]. Association rules evaluate co-occurrence via support, confidence, lift, and leverage [29, 30]. These measures foreshadow the evidence semantics in **RQ2**, but they are not designed for multi-relation existential heads.

**Association Rules.** Rules of the form $X \Rightarrow Y$ quantify co-occurrence with support, confidence, lift, and leverage. They operate on itemsets/attributes and optimize frequency-based evidence, not existential heads with shared witnesses.

For an item universe $\mathcal{I}$ and transactions $\mathcal{T}$, an association rule $X \Rightarrow Y$ with $X, Y \subseteq \mathcal{I}, X \cap Y = \emptyset$ has support $\text{supp}(X \cup Y) = |\{T \in \mathcal{T} \mid X \cup Y \subseteq T\}|/|\mathcal{T}|$ and confidence $\text{conf}(X \Rightarrow Y) = \text{supp}(X \cup Y)/\text{supp}(X)$.

Viewing each `RESIDENCE` row as a transaction with items like (Location = Memphis) and (State = TN), the rule

$$\text{(Location = Memphis)} \Rightarrow \text{(State = TN)}$$

has support $2/3$ (two of three rows) and confidence 1.0 in the toy data.

## 3.2 Existential rules and TGDs

**TGDs and EDs.** EDs subsume TGDs and EGDs; TGDs assert that body patterns imply head patterns that may introduce existential witnesses [31, 32]. TGDs subsume INDs and many mapping constraints, and they naturally span multiple relations. This expressivity is attractive for **RQ1** but raises concerns about search and reasoning complexity.

**Tuple-Generating Dependencies (TGDs).** TGDs have the form $\forall \vec{x}, \vec{y} : \phi(\vec{x}, \vec{y}) \Rightarrow \exists \vec{z} : \psi(\vec{x}, \vec{z})$, possibly with multi-atom heads. The existential witnesses $\vec{z}$ are *shared* across head atoms, coupling them.

Formally, we assume $\phi$ and $\psi$ are conjunctions of relational atoms over a fixed schema, that all universally quantified variables occur in the body (safety), and we write the *frontier* as the variables shared between body and head, $\mathsf{fr}(\phi \Rightarrow \psi) = \mathsf{vars}(\phi) \cap \mathsf{vars}(\psi)$. Under a closed-world interpretation, we reuse the evidence notions from Preliminaries, *prediction*, *support*, and *confidence*, as defined in Definitions 2.2.1, 2.2.2, and 2.2.3, computed over the free tuple $\vec{x}$.

From spouse to spouse-in-same-state:

$$\forall x, y, \ell, s : \texttt{MARRIAGE}(x, y) \wedge \texttt{RESIDENCE}(x, \ell, s) \Rightarrow \exists \ell' : \texttt{RESIDENCE}(y, \ell', s).$$

In the toy database, the body holds for three ordered pairs $(x, y)$: $(Elvis, Priscilla)$, $(Priscilla, Elvis)$, and $(Lisa, Danny)$; it fails for $(Danny, Lisa)$ as *Danny* has no residence. The head is satisfied for the first two pairs (e.g., with $s = TN$, a witness is $\ell' = Memphis$ for both *Elvis* and *Priscilla*), but not for $(Lisa, Danny)$ since *Danny* has no residence in state *CA*. By Definition 2.2.1, $\text{Prediction}(R)$ therefore consists of exactly those two pairs, and by Definition 2.2.3 we obtain $\text{Confidence}(R) = 2/3$; $\text{Support}(R)$ follows analogously from Definition 2.2.2.

**Horn Rules.** Horn clauses are single-head, non-existential rules and form a strict fragment of TGDs. They dominate ILP and KG mining; evidence is often standard/PCA confidence on binary predicates.

A Horn rule has the form $B_1 \wedge \cdots \wedge B_k \Rightarrow H$ with a single non-existential head.

Symmetry in marriage as a Horn rule over binary predicates:

$$\forall x, y : \texttt{MARRIAGE}(x, y) \Rightarrow \texttt{MARRIAGE}(y, x).$$

In the toy data both directions are materialized for each pair, so the rule is satisfied.

**Decidable fragments for reasoning.** Guarded, frontier-guarded, weakly-acyclic, and sticky TGDs restrict variable propagation to guarantee chase termination and decidable query answering [33, 34, 35]. These lines are orthogonal to mining, yet they inform post-mining usage: mined rules that fall into a decidable fragment enable downstream reasoning. Second-order TGDs enable composition in schema mapping but are beyond our mining scope [33].

**Guarded TGDs.** All universally quantified variables must occur together in a single *guard* atom of the body, which disciplines propagation and aids chase termination. Concretely, in a TGD $\forall \vec{x}, \vec{y} : \ \phi(\vec{x}, \vec{y}) \Rightarrow \exists \vec{z} : \ \psi(\vec{x}, \vec{z})$, there exists an atom $G$ in $\phi$ whose argument list contains every universally quantified variable.

A guarded rule over the toy schema is:

$$\forall x, y : \ \texttt{MARRIAGE}(x, y) \ \Rightarrow \ \exists \ell, s, \ell' : \ \texttt{RESIDENCE}(x, \ell, s) \wedge \texttt{RESIDENCE}(y, \ell', s).$$

Here the atom $\texttt{MARRIAGE}(x, y)$ guards all universals $x, y$. For $(x, y) = (\textit{Elvis}, \textit{Priscilla})$, witnesses are $\ell = \ell' = \textit{Memphis}$, $s = \textit{TN}$. The rule fails on $(\textit{Lisa}, \textit{Danny})$ since *Danny* has no residence.

**Frontier-Guarded TGDs.** Only variables shared between body and head (the *frontier*) must appear together in a guard, relaxing guardedness while preserving decidability. Formally, if $\mathsf{fr}(\phi \Rightarrow \psi) = \mathsf{vars}(\phi) \cap \mathsf{vars}(\psi)$, there exists an atom in the body that contains all variables in $\mathsf{fr}(\phi \Rightarrow \psi)$.

An illustrative frontier-guarded rule is:

$$\forall x, \ell, s : \ \texttt{RESIDENCE}(x, \ell, s) \ \Rightarrow \ \exists \ell' : \ \texttt{RESIDENCE}(x, \ell', s).$$

The frontier is $\{x, s\}$, which appears together in the guard atom $\texttt{RESIDENCE}(x, \ell, s)$. For *Elvis* and *Priscilla*, one can take $\ell' = \textit{Memphis}$; for *Lisa*, $\ell' = \textit{Beverly Hills}$.

**Weakly-Acyclic TGDs.** Defined via a position dependency graph on predicate positions: ordinary edges capture variable flow between positions within and across rules, while *special* edges capture creation of fresh nulls by existentials and their propagation. A set of TGDs is *weakly-acyclic* (WA) if the graph contains no cycle that passes through a special edge, which ensures termination of the (oblivious) chase.

The single-rule sets given above are WA. For example, in $\texttt{MARRIAGE}(x, y) \ \Rightarrow \ \exists \ell, s, \ell' : \ \texttt{RESIDENCE}(x, \ell, s) \wedge \texttt{RESIDENCE}(y, \ell', s)$, existentials only populate RESIDENCE positions; there is no flow back from those positions to any position that would, in turn, generate new existentials, hence no special cycle.

**Sticky TGDs.** Sticky properties prevent values introduced during the chase from being lost in joins, bounding propagation and enabling decidable query answering. Intuitively, variables that occur more than once in a body (join variables) are *marked* and must persist: once a value is bound to a marked variable, it cannot be "dropped" by subsequent joins without appearing in the head.

A sticky rule over the toy schema is:

$$\forall x, y, \ell, s : \ \texttt{MARRIAGE}(x, y) \wedge \texttt{RESIDENCE}(x, \ell, s) \ \Rightarrow \ \exists \ell' : \ \texttt{RESIDENCE}(x, \ell', s).$$

The join variable $x$ appears in both body atoms and also in the head, satisfying stickiness. For $(x, y) = (\textit{Elvis}, \textit{Priscilla})$ with $s = \textit{TN}$, a witness is $\ell' = \textit{Memphis}$. By contrast, the spouse-same-state rule $\texttt{MARRIAGE}(x, y) \wedge \texttt{RESIDENCE}(x, \ell, s) \Rightarrow \exists \ell' \, \texttt{RESIDENCE}(y, \ell', s)$ is *not* sticky because the marked join variable $x$ is lost from the head.

**Discovery systems.** Before this work, general-purpose miners for arbitrary first-order TGDs with multi-atom heads and existential variables were lacking. Toolchains like Metanome cover FDs, UCCs, INDs, ODs, DCs [17], and schema-mapping systems specify TGDs but do not discover them from raw data [22]. This gap is central to **RQ1**.

## 3.3 Probabilistic and representation-learning hybrids

Hybrid approaches combine logical structure with probabilistic reasoning or learned representations, seeking the predictive strength of statistical models while retaining some of the interpretability of symbolic rules; here we summarize representative families and clarify how they differ from our auditable TGD setting.

**Probabilistic formalisms.** *Markov Logic Networks (MLNs)* attach real-valued weights to first-order formulas and interpret them as soft constraints in a Markov network, so that higher-weighted formulas bias the distribution toward worlds in which they are satisfied [36]. *Statistical schema induction* mines association-style regularities (e.g., subclass, domain/range axioms) from instance data using support/confidence thresholds, thereby proposing ontology/schema axioms that reflect frequent inclusion or co-typing patterns [37]. *PC* performs constraint-based causal discovery by using conditional independence tests to prune edges and orient directions in a directed acyclic graph (DAG), targeting causal, not merely correlational, structure [38, 39]. *FCI* extends PC to handle latent confounders and selection bias, outputting equivalence classes of partially oriented graphs consistent with observed independences [38, 39]. *NOTEARS* formulates DAG learning as a continuous optimization with a differentiable acyclicity constraint, enabling gradient-based search over graph structures [40]. These paradigms emphasize uncertainty, causality, or concept induction; by design they trade exact, tuple-level audit trails for probabilistic semantics and are not aimed at mining deterministic TGDs with $n$-ary joins.

**Embedding-guided rule mining.** *TransE* represents relations as translation vectors so that a true triple $(h, r, t)$ satisfies $\mathbf{h} + \mathbf{r} \approx \mathbf{t}$, capturing many compositional and hierarchical regularities [41]. *DistMult* scores triples via a diagonal bilinear form $\langle \mathbf{h}, \mathbf{r}, \mathbf{t} \rangle$, efficiently modeling symmetric interactions but struggling with asymmetric ones [42]. *ComplEx* extends bilinear scoring to complex vectors, enabling the modeling of asymmetric relations while retaining efficient inference (often grouped with the DistMult family) [42]. *Hybrid systems* use embedding scores to steer Horn-rule search, e.g., to rank candidate bodies, propose promising atoms, or augment incomplete graphs, thereby discovering rules that the observed data alone might not surface [42, 43, 44]. These approaches excel at link prediction, but their evidence is *soft*: support derives from high embedding scores rather than exact counts, so **for RQ2** they provide numerical plausibility rather than the deterministic support and confidence we report over concrete joins.

## 3.4 Cross-cutting evaluation criteria

Because applications value different facets of quality, prevalence, logical soundness, or cleanliness, rule evaluation draws on three complementary paradigms, each answering a distinct question about a pattern's usefulness.

**Frequency-based measures.** *Support* is the number (or proportion) of instances that satisfy both body and head, indicating how often a rule materializes in the data and guarding against spurious, low-frequency coincidences [29, 30]. *Confidence* is the conditional probability of the head given the body, measuring reliability: among all body instances, what fraction also satisfy the head [29, 30]. *Lift* compares observed co-occurrence to independence,

$$\text{lift} = \frac{\Pr(\text{body} \wedge \text{head})}{\Pr(\text{body})\,\Pr(\text{head})},$$

so values $> 1$ indicate positive association and $= 1$ indicate no association [29, 30]. *PCA confidence* adapts confidence to incomplete knowledge bases by only penalizing a body instance if the knowledge base is known to contain some head value for that subject, making estimates less sensitive to open-world incompleteness [45]. In our setting we adopt standard *support* and *confidence* generalized to $n$-ary joins (and to existential heads) and report both exact and high-confidence approximate regularities, preserving tuple-level auditability.

**Consistency-based evaluation (ILP).** ILP assesses hypotheses by *covering positives* while *excluding negatives*: a good rule entails many labeled positives and few (ideally zero) labeled negatives, often optimized via heuristics such as information gain or failure-driven pruning [46, 47, 48]. This yields rules that are logically consistent with a supervised specification, trading distributional prevalence for adherence to labeled examples. For **RQ2**, this perspective stresses that when bodies span joins or heads include existentials, "positives" and "negatives" must be counted under well-defined, database-native semantics to remain stable and interpretable.

**Violation-based profiling (dependencies).** *Functional dependencies (FDs)* $X \rightarrow Y$ are evaluated by counting violating tuple pairs that agree on $X$ but disagree on $Y$, with fewer violations indicating stronger regularity [27, 28]. *Conditional FDs (CFDs)* constrain FDs to context patterns (e.g., a country-specific rule), so violations are computed within the conditioning slice [27, 28]. *Inclusion dependencies (INDs)* require that values of one attribute set are contained in another (e.g., foreign-key containment), and violations count values lacking a counterpart [17]. *Denial constraints (DCs)* forbid specified value combinations (possibly across tuples); each forbidden witness is a violation instance [19, 18]. This paradigm reports *violation counts/rates* rather than prevalence, complementing frequency and consistency views and highlighting data quality aspects. Aligned with **RQ2**, we ensure that when joins and existentials are present, our evidence remains auditable (support sets and violation witnesses are concretely enumerable), so stability and interpretability carry over to expressive patterns.

### 3.4.1 Mining expressive pattern dependencies (related to RQ1)

Recent work has pushed beyond single-head Horn rules toward richer, multi-atom implications on complex schemas and graphs:

- **Graph-Generating Dependencies (GGDs) on property graphs.** *GGDMiner* discovers GGDs that generalize TGDs to property graphs: whenever a source pattern (nodes/edges) appears, a target pattern, possibly introducing new nodes/edges, must also exist [49]. The mining pipeline combines pre-processing, candidate pattern generation, and GGD extraction, and reports auditable support/confidence. Empirically, many multi-edge regularities missed

by single-head Horn miners (e.g., AMIE) are recovered, supporting **RQ1** feasibility for complex, multi-atom heads.

- **Graph Pattern Association Rules (GARs).** Extending association rules to single-graph settings, GARs use pattern antecedents and consequents with frequency-based evidence (support/confidence) [50]. While scalable and able to mix edges and vertex attributes, the consequent is restricted to a single edge/attribute, highlighting a limitation vs. TGDs/GGDs for full subgraph implications.
- **Scaling ILP to "big" rules.** *JOINER* composes many small learned clauses into large Horn rules (dozens to 100+ body literals) via constraint solving [51]. Although supervised and still single-head/non-existential, this decomposition strategy demonstrates tractability for very expressive patterns, informing **RQ1** by suggesting modular enumeration/combination in unsupervised mining as well.

### 3.4.2 Robust evidence semantics and hybrid approaches (related to RQ2)

Stability of support/confidence under joins, existentials, and imperfect data has been addressed along three lines:

- **Rule mining under incompleteness (PCA and embeddings).** PCA confidence remains standard in KG mining to mitigate missing facts. Studies show that naïve KG completion with embeddings can distort evidence and yield spurious rules (e.g., with TransE) [44]. Controlled hybrids instead augment the KG with high-precision predictions from pre-trained embeddings before rule mining, uncovering new valid rules while retaining classical support/confidence [43]. This cautions against uncurated imputations and directly targets **RQ2** robustness.
- **Noise-tolerant dependency measures in relational profiling.** Dynamic approximate dependencies adapt tolerance per attribute to data quality. *DAFDiscover* mines Dynamic Approximate FDs by learning per-attribute thresholds, improving robustness to noise while keeping a clear link to exact FDs [52]. This graded view of satisfaction aligns with **RQ2** goals of stable, auditable measures under noise.
- **Neural/probabilistic hybrids with aggregation.** Modern neuro-symbolic pipelines (e.g., AnyBURL with SAFRAN-style clustering/aggregation, as discussed in [53]) combine multiple rule variants via Noisy-OR or max aggregation, trading a bit of direct auditability for resilience to missing edges and complex joins. Embedding-guided miners such as *RLvLR* and *TyRuLe* bias the search toward promising joins using vector-space cues while still reporting standard support/confidence for the final rules [54, 55]. These hybrids strengthen **RQ2** by improving both efficiency and reliability under incompleteness.

**Take-away.** Together, these advances demonstrate that (i) expressive, multi-atom dependencies (TGDs/GGDs) can be mined at scale on real graphs and complex schemas (**RQ1**), and (ii) evidence semantics can be made robust to joins, existential witnesses, and imperfect data via principled counting, careful completion, and hybrid guidance (**RQ2**). This trajectory reinforces the need for a unified framework (as pursued in this thesis) that mines general TGDs with auditable support/confidence on noisy, real-world data.

## 3.5 Representative miners and learners

This closing section consolidates the representative miners and learners cited throughout the chapter and situates them with respect to our guiding questions **RQ1** and **RQ2**. It serves as a compact, system-centric summary with small illustrative examples, complementing the thematic discussion above.

### 3.5.1 SPIDER (Metanome)

SPIDER (2007–2010) was among the first I/O-efficient algorithms for large-scale IND discovery and underpinned Metanome, a unifying platform for constraint profiling.

SPIDER targets unary INDs $A \subseteq B$ from relational data [56, 24, 17]. It deduplicates and externally sorts each attribute to obtain runs $L_A$, then validates inclusion by a merge-scan. Cheap prefilters on distinct counts and min–max ranges prune pairs early. Producing runs costs $O\big(\sum_A |V(A)| \log |V(A)|\big)$; validation of a surviving pair is linear in list sizes. SPIDER established an I/O-efficient baseline for IND discovery and was integrated in Metanome. For **RQ1**, SPIDER demonstrates scalability but only for a narrow TGD fragment. For **RQ2**, its evidence is violation counting rather than general support and confidence.

In particular, SPIDER uses lightweight prefilters and early termination: (i) *domain-size/range* checks to skip infeasible attribute pairs; (ii) *sorting-based* merge scans that stop at the first mismatch [56, 24, 17].

On the toy schema, the unary inclusion

$$\pi_{\text{Partner1}}(\texttt{MARRIAGE}) \subseteq \pi_{\text{Parent}}(\texttt{LINEAGE})$$

is satisfied: every `Partner1` value also occurs in `LINEAGE.Parent`.

By default, SPIDER checks *exact* inclusion. In noise-tolerant variants, one typically reports a violation rate or an "IND confidence"

$$\text{conf}_{\text{IND}}(A \subseteq B) = \frac{|\pi_A(R) \cap \pi_B(S)|}{|\pi_A(R)|},$$

and/or a target-side coverage $\text{cov}(A \subseteq B) = \frac{|\pi_A(R) \cap \pi_B(S)|}{|\pi_B(S)|}$. Systems also report the number of counterexamples, candidate keys (to weed out false positives), and thresholds (e.g., min conf). These metrics remain violation-centric and do not generalize to multi-atom existential heads. *How rule value is judged.* **SPIDER** reduces evaluation to checking set inclusion $\pi_A(R) \subseteq \pi_B(S)$; confidence-style metrics are generally not used beyond violation rates, and composite or approximate dependencies require extensions [56, 24, 17].

### 3.5.2 AMIE3

AMIE (followed by AMIE+ and AMIE3) was among the first systems to mine Horn rules at knowledge-graph scale and popularized *PCA confidence* to mitigate open-world incompleteness in KBs.

AMIE3 mines Horn rules $B_1 \wedge \cdots \wedge B_k \Rightarrow H$ over triple knowledge bases with exact support and both standard and PCA confidence [45, 57]. Exploration is level-wise with canonical generation, anti-monotone support pruning, confidence upper bounds, and multi-index counting. PCA confidence

addresses open-world incompleteness. With respect to **RQ1**, AMIE3 scales well for single-head binary predicates but cannot directly express multi-head existential TGDs. For **RQ2**, it motivates robust confidence notions; our closed-world setting generalizes standard support and confidence to n-ary joins. In particular, AMIE3 is tailored to triple-formatted knowledge bases (binary predicates) and therefore assumes low-arity relations; it does not natively handle n-ary relations or multi-atom existential heads that span several tables in a relational schema.

Concretely, AMIE3 combines syntactic and semantic pruning [58]: (i) *redundancy pruning* collapses equivalent permutations/renamings; (ii) *bounds-driven pruning* cuts rule prefixes whose support or confidence *upper bounds* fall below thresholds; (iii) *closedness* maintains variable grounding/connectedness during refinement.

On triples one typically sees patterns like

$$\texttt{hasSpouse}(x, y) \wedge \texttt{livesIn}(x, z) \Rightarrow \texttt{livesIn}(y, z),$$

or

$$\texttt{parent}(x, z) \wedge \texttt{spouse}(x, y) \Rightarrow \texttt{parent}(y, z).$$

These rules are single-head and non-existential.

AMIE3 evaluates rule quality with a compact set of frequency-based metrics: *exact support*, i.e., the number of groundings $\theta$ for which both body and head hold $\big(\text{supp}(B \Rightarrow H) = |\{\theta \mid B\theta \wedge H\theta\}|\big)$; *standard confidence* $\big(\text{conf}(B \Rightarrow H) = \frac{\text{supp}(B \wedge H)}{\text{supp}(B)}\big)$; and *PCA confidence* (Partial Completeness Assumption), which replaces the denominator with the number of body assignments for which at least one potential head fact is known or assumed complete, thereby mitigating open-world bias; AMIE3 also reports *head coverage*. During search, tight *upper bounds* on confidence guide pruning, while canonical rule generation and multi-index counting accelerate evaluation. These choices provide precise guidance under open-world incompleteness but still do not address multi-atom existential heads on n-ary schemas. *How rule value is judged.* **AMIE3** also reports exact support and confidence, and introduces *PCA confidence* to mitigate open-world incompleteness in KBs [58]. This is well-suited to triple KBs (Wikidata/YAGO) but does not address multi-atom existential heads on n-ary schemas.

### 3.5.3 POPPER (ILP)

POPPER revived a "generate–test–constrain" ILP lineage, leveraging ASP to learn exclusion constraints, while aligning with the MIL (Meta-Interpretive Learning) tradition for rich biases and recursion. POPPER learns Prolog programs from positives and negatives using a generate–test–constrain loop with ASP-backed constraint learning [48]. It supports recursion and rich biases, yielding compact Horn theories. For **RQ1**, completeness within bounds is attractive, but unsupervised database profiling lacks labeled examples and requires strong joinability biases to remain tractable. For **RQ2**, evaluation is largely Boolean consistency rather than frequency-based evidence.

Beyond the loop itself, POPPER employs *failure-driven pruning*: each failed hypothesis yields constraints (no-goods) that prune entire regions of the search generalizations or specializations, accumulated incrementally by the ASP solver [48].

A learned Horn rule may take the form

$$\texttt{ancestor}(x, y) \Leftarrow \texttt{parent}(x, y) \quad \text{or} \quad \texttt{ancestor}(x, y) \Leftarrow \texttt{parent}(x, z) \wedge \texttt{ancestor}(z, y).$$

The setting is supervised (positive/negative examples) rather than support/confidence–oriented. *How rule value is judged.* POPPER targets *logical consistency*: covering positives and excluding negatives, under parsimony constraints (minimize clauses/literals) and structural bounds. It does not optimize a frequency metric; instead it seeks correct programs under the specified bias, sometimes with simple objectives (e.g., program size) for tie-breaking. Correctness is checked by Prolog execution over background knowledge and examples [48].

## 3.6 Positioning with respect to RQ1 and RQ2

**Guiding questions (reminder).** **RQ1** asks about the *feasibility and tractability* of mining expressive first-order TGDs with connected bodies (and possibly multi-atom existential heads) on real, n-ary schemas while keeping enumeration and statistics auditable. **RQ2** asks which *evidence semantics* (support and confidence) remain stable under joins and existential variables, how robust they are to incompleteness/noise, and how to compute them efficiently.

**Toward RQ1.** As summarized in the competitors overview (see Sec. 3.5), systems such as SPIDER target unary inclusion dependencies on relational data, while AMIE3 and ILP/Prolog learners (e.g., POPPER) focus on single-head Horn rules predominantly over binary predicates. None of these directly searches for *multi-atom existential heads on n-ary schemas*. Our framework organizes the hypothesis space by a *constraint graph* of joinable attributes: this removes non-joinable combinations by construction, collapses isomorphic expansions via canonicalization, and bounds branching to empirically relevant neighborhoods. Candidate generation proceeds by enumerating canonical simple paths (proto-rules) and splitting body/head to obtain TGDs, with normalization ensuring duplicate-free search. On the systems side, tractability is maintained by reusing cached unary/binary projections, performing streaming intersections for multi-way joins, and incrementally updating sufficient statistics across refinements. Together, these design choices keep enumeration predictable on real schemas while preserving expressiveness beyond the fragments handled by prior miners.

**Toward RQ2.** We require that evidence be *interpretable and stable* in the presence of joins and existential witnesses. To this end, we generalize classical *support* (head coverage) and *confidence* to n-ary joins under a *closed-world* assumption, yielding deterministic counts that are auditable and reproducible. To prevent self-witnessing artifacts in recursive or self-join settings, we adopt *disjoint semantics*, which enforce distinct key assignments across repeated relation occurrences. Where knowledge-graph miners often resort to PCA confidence to cope with open-world incompleteness, we retain standard confidence and expose approximate regimes via explicit thresholds (and, when applicable, early-stop bounds), aligning with data-profiling practice while extending it to TGDs. The result is a semantics that remains stable under joins and existential variables and whose statistics can be computed efficiently with our caching and streaming pipeline.

## 3.7 Chapter summary

**Miners at a glance.** We highlight three representative systems. **SPIDER (Metanome)** [56, 24, 17] targets unary inclusion dependencies on relational data. It combines cardinality and range

| System | Data model | Target patterns | Search & pruning | Main limits |
|---|---|---|---|---|
| **SPIDER** [56, 24, 17] | Relational, n-ary | Unary INDs, FK inference | Deduplicate, external sort, merge-scan pairs; cardinality/range prefilters, early stop; violation counts | Narrow expressivity, $O(n^2)$ pairs on wide schemas, brittle under noise |
| **AMIE3** [58] | KB triples (binary) | Horn rules, single head | Level-wise expansion with canonicalization/bounds, multi-index counting; min support, confidence upper bounds, PCA confidence | Binary predicates, open-world assumptions, no multi-head existential heads |
| **POPPER** [48] | Relational facts with labels | Horn rules (optionally recursive) | Generate–test–constrain via ASP/Prolog execution; failure-driven constraint learning; Boolean consistency on examples | Requires labeled data and strong bias; cost grows with bounds |

**Table 3.1:** High-level comparison of representative systems and this work

prefilters with a *merge-scan* operator that admits early stopping, and it reports evidence in the form of inclusion and violation counts, effective for profiling, but limited in expressivity. **AMIE3** [45, 58] learns Horn rules over knowledge-base triples via canonical level-wise expansion with support/confidence bounds. It evaluates *support*, *confidence*, and *PCA confidence*, but remains confined to binary predicates and cannot express multi-atom existential heads. **POPPER (ILP)** [48] follows a supervised *generate–test–constrain* loop implemented in ASP with failure-driven pruning; its "evidence" is consistency with labeled positives/negatives together with parsimony, rather than frequency objectives, powerful when labels and a strong bias are available.

**Other dependency families.** Classical data-profiling dependencies complement these learners. **Functional dependencies (FDs)** [14, 13] capture functional uniqueness and are assessed via counterexamples (exact or approximate), underpinning keys and normalization. **Equality-generating dependencies (EGDs)** [13] enforce derived equalities and are likewise verified by counterexample search. **Unique column combinations (UCCs)/keys** [15, 16] are discovered by uniqueness tests (often via external sort or indexing) and inform schema design. **Inclusion dependencies (INDs)** [17] of the form $R[X] \subseteq S[Y]$ are validated efficiently by *merge-scan*, with approximate variants based on violation rates. **Denial constraints (DCs)** [18] declare forbidden patterns with built-ins and are evaluated by violations rather than frequencies. **Order dependencies (ODs)** express order implications $\vec{X} \rightsquigarrow \vec{Y}$ and are checked via order-violation detection. **Join dependencies (JDs)** characterize lossless decompositions and, beyond special refactoring cases, are rarely mined in practice.

**RQ1/RQ2 take-away.** Together these systems span three evidence paradigms, violation-based, frequency-based, and supervision-based. Yet none directly mines first-order TGDs with multi-atom existential heads while emitting auditable support and confidence. Filling this gap is precisely the objective of **MATILDA**.

# Chapter 4

# Mining TGDs on Tabular Data

**Guiding questions.** This chapter addresses the following research questions: (i) **RQ1** feasibility and tractability of mining expressive first-order TGDs with connected bodies and possibly multi-atom heads on real schemas, together with auditable statistics; (ii) **RQ2** evidence semantics, i.e., which support and confidence notions remain stable under joins and existential variables, how robust they are to incompleteness and noise, and how they can be computed efficiently; (iii) **RQ3** search spaces and pruning principles enabling effective discovery without sacrificing completeness or interpretability, and how duplicates can be characterised and avoided; (iv) **RQ4** when recursive dependencies arise and add material value, and which evaluation/counting semantics prevent trivial or self-justifying rules; and (v) **RQ5** how single-head Horn rules compare to FO-TGDs empirically and in practice, and under which regimes additional expressiveness (multi-head, existential, recursive) is necessary.

First-Order Tuple-Generating Dependencies (TGDs) are general database constraints that subsume Horn rules, foreign keys, and inclusion dependencies. We present MATILDA, a chase-free approach that mines TGDs directly from tabular data using a constraint graph of joinable attributes. Paths in this graph define *proto-rules*; splitting a path yields candidate (non-recursive) TGDs, which we score with closed-world support and confidence (with explicit `NULL` handling) to produce both exact and approximate rules. Tailored pruning and data-guided counts keep the exponential search space tractable, and outputs are de-duplicated up to logical equivalence. Across 54 benchmark databases, MATILDA (i) recovers all inclusion dependencies and Horn rules reported by competing systems, (ii) scales to databases out of reach for them, and (iii) discovers additional multi-head TGDs that other methods cannot express, mining about 50% more joinable patterns overall while keeping runtimes competitive (from sub-second to a few minutes, depending on database size). Our contributions are: (1) a redundancy-free constraint-graph formulation that reduces TGD discovery to path enumeration; (2) a DFS-based enumeration algorithm with early, data-driven pruning and correctness guarantees (completeness and uniqueness up to equivalence); (3) principled evidence metrics (support/confidence) under a closed-world assumption with explicit treatment of `NULL`; and (4) a graph-centric extension toward recursive TGDs that preserves the same design principles.

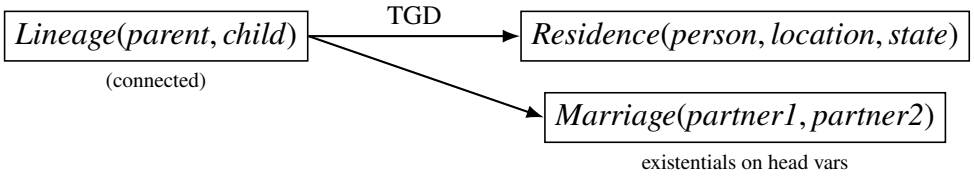


**Figure 4.1:** Illustration of TGD mining: connected body implies multi-atom head with possible existential variables; evidence is given by support and confidence.

## 4.1 TGD mining

We first formalize the mining task before introducing the underlying graph representation and the algorithmic machinery (illustrated in Figure 4.1).

**Definition 4.1.1** (TGD mining)**.** *Given a database, TGD mining is the task of finding all transitively connected TGDs (up to logical equivalence), with their confidence and support.*

**Example 4.1.1.** *In our toy database from Figure 1.1, the goal could be to mine "If someone is a parent then the person is married and has a residence":*

$$\forall x_1, y_1 : Lineage(x_1, y_1) \Rightarrow \exists z_1, z_2, z_3 : \\ Residence(x_1, z_1, z_2) \wedge Marriage(x_1, z_3)$$

## 4.2 TGD mining complexity

Mining TGDs naively faces two core obstacles. **(i) Duplication.** Infinitely many logically equivalent variants arise from variable renamings, atom permutations, and benign duplications; post-hoc isomorphism checks (as in Horn-rule miners) are costly. **(ii) Combinatorics.** Enumerating how attribute positions coalesce into equality classes across joins explodes: for $n$ attributes, the number of set partitions is the Bell number $B_n$ with $B_0$=1 and

$$B_n = \sum_{k=0}^{n-1} \binom{n}{k} B_k .$$

This growth is super-exponential (for intuition: $B_{10}$=115,975 and $B_{15} \approx 1.38 \times 10^9$), so a partition-based search becomes quickly infeasible.

**Example 4.2.1.** *For a database with a total of 75 attributes (i.e., n=75), the number of required permutations, $b_{75}$, exceeds $10^{80}$, surpassing the estimated number of hydrogen atoms in the observable universe.*

These difficulties motivate our graph-based formulation and pruning strategy, introduced next.

## 4.3 Core definitions

We've replaced the difficult partition-based search with a formulation based on graph theory.The *Constraint Graph* (Definition 4.3.3) compactly represents equalities realizable by schema-valid joins: each node denotes a candidate equality between attribute positions (i.e., a binding to a common logical variable), and an edge certifies that two candidates can be jointly realized without contradiction, given type compatibility and declared constraints (keys and inclusion dependencies).

In our approach, finding unique rules (TGDs) is equivalent to navigating unique paths in a graph. Each path gathers a set of consistent logical statements, forming a preliminary *proto-rule*. This *proto-rule* is then finalized into a TGD by systematically partitioning its variables. We guarantee that no duplicate rules are ever created because we use a canonical representation for these paths, which treats structurally identical rules as the same, regardless of how their variables are named.

The graph view affords principled, data-aware pruning. Edges are admitted only when implied by schema-guided joinability, excluding nonsensical combinations a priori. Support is anti-monotone under path extension, so prefixes with insufficient evidence are discarded early. All counts are computed under closed-world semantics with explicit `NULL` treatment, yielding auditable support and confidence.

In sum, we replace partition-based enumeration with a compact, redundancy-free traversal whose branching is governed by the schema and whose depth is curtailed by empirical evidence.

### 4.3.1 Constraint Graph

One may think that the simplest way to represent the space of possible TGDs would be a graph where the nodes are attributes and the edges say whether two attributes are joinable. However, such a graph would allow enumerating attributes and not TGDs. Therefore, we construct a different graph, which we call the *constraint graph*. For this purpose, we assume a complete order of the tables of the database and a complete order of the attributes of each table (e.g., the lexicographical order). We also assume a joinability relation between attributes (Figure 4.2).

> **Definition 4.3.1** (Joinable Attributes)**.** *Two attributes are* joinable *if a join can be semantically justified.*


**Figure 4.2:** Joinable attributes: columns with compatible semantics/domains that can be meaningfully joined.

Joinable attributes can be specified manually by users, they can be derived from database constraints, or they can be mined with schema matching algorithms [59]. The joinability relation is symmetric and reflexive (a column can be joined with itself, see later in Section 4.5). However, joinability is not transitive: given a table $T$ with three attributes $T.a$, $T.b$, and $T.c$, one can have that $T.a$ and $T.b$ are joinable ($T.a \cap T.b \neq \emptyset$) and that $T.b$ and $T.c$ are joinable ($T.b \cap T.c \neq \emptyset$), but this does not mean that $T.a$ and $T.c$ are joinable ($\not\Rightarrow T.a \cap T.c \neq \emptyset$).

Let us first consider non-recursive TGDs (we will extend our definitions to recursive TGDs in Section 4.5). In non-recursive TGDs, a table may not appear in both the body and the head.

However, multiple occurrences of the same table may appear on the same side (either all in the body or all in the head). This allows us to define table-linked attributes:

**Definition 4.3.2** (Table-linked Attributes)**.** *Two attributes $T.a$ and $T'.a'$ are table-linked if they share the same table, i.e., if $T = T'$.*

*Two pairs of joinable attributes, $v_1 = (T_1.a_1, T'_1.a'_1)$, $v_2 = (T_2.a_2, T'_2.a'_2)$ are table-linked if at least one attribute from $v_1$ is table-linked with at least one attribute from $v_2$ (Figure 4.3). If $v_1$ and $v_2$ share an attribute, then the joinable attributes are* perfectly table-linked.

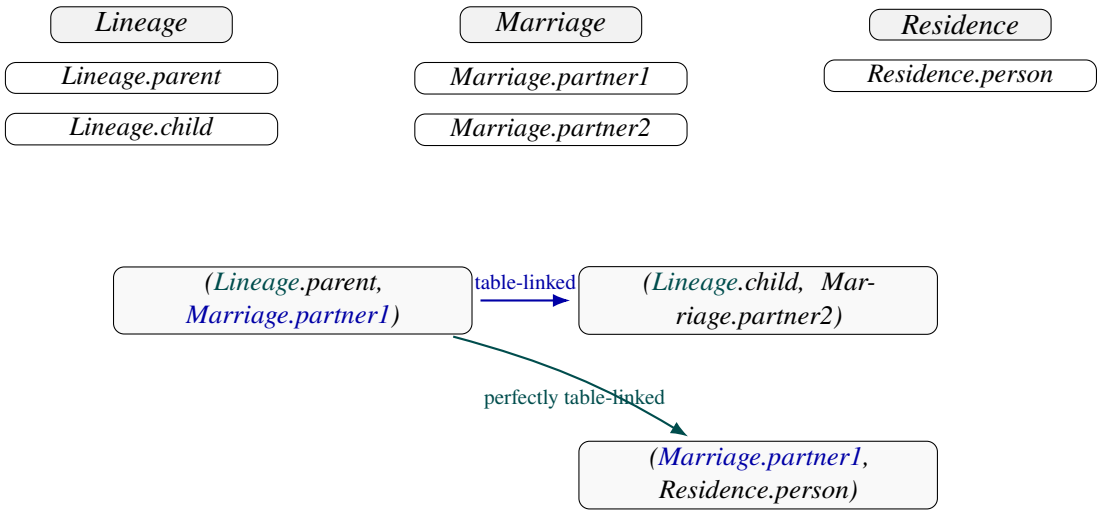


**Figure 4.3:** Table-linked attributes: two attributes are table-linked if they are in the same table. Two pairs of joinable attributes are table-linked if at least one attribute from the first pair is table-linked with one from the second; they are perfectly table-linked if they share an attribute.

**Example 4.3.1.** *Using the toy database of Figure 1.1, several attributes belong to the same table and are therefore table-linked. For instance, Lineage.parent and Lineage.child are table-linked because both belong to the Lineage table.*

*Consider the following pairs of joinable attributes:*

$$v_1 = (\textit{Lineage.parent},\ \textit{Marriage.partner1}) \qquad v_2 = (\textit{Lineage.child},\ \textit{Marriage.partner2})$$

*The two pairs $v_1$ and $v_2$ are table-linked because Lineage.parent and Lineage.child are table-linked (same table). They are not perfectly table-linked since they do not share the same attribute.*

*By contrast, if we take*

$$v_3 = (\textit{Marriage.partner1},\ \textit{Residence.person})$$

*then $v_1$ and $v_3$ are perfectly table-linked because they share the attribute Marriage.partner1.*

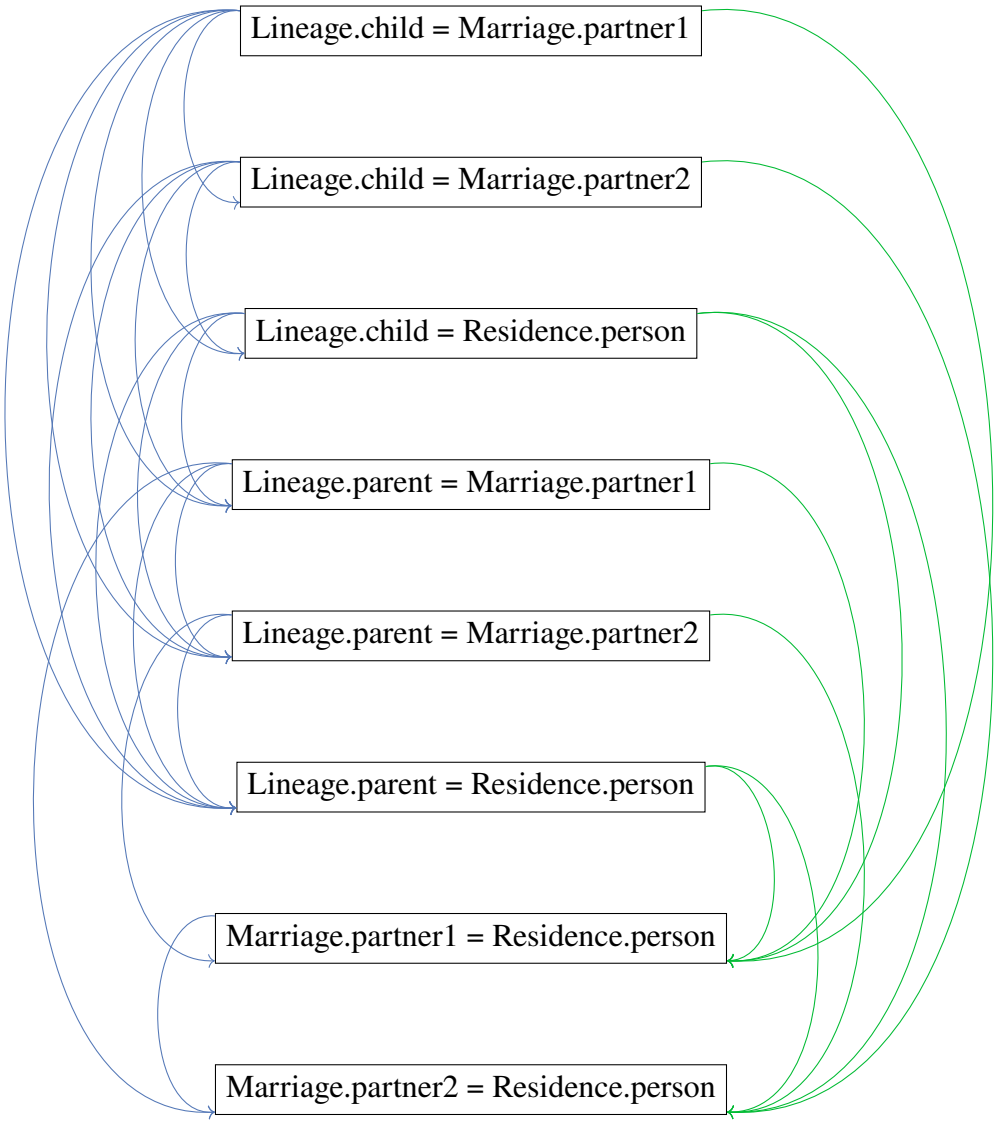


**Figure 4.4:** The constraint graph based on our toy example database: tables Lineage, Residence, and Marriage. The blue arrows represent table-linked edges (nodes share at least one table-bound attribute). The green arrows represent perfectly table-linked edges (nodes share exactly one attribute). Note: all perfectly table-linked edges also represent table-linked relationships.

**Definition 4.3.3** (Constraint Graph)**.** *The constraint graph of a database (illustrated in Figure 4.4) is the directed unlabeled acyclic graph where*

- *A node $(T.a, T'.a')$ is a pair of joinable attributes such that $T.a < T'.a'$. We write the node as $T.a = T'.a'$*[a].

- *An edge* $(v, v')$ *links two table-linked nodes* $v$ *and* $v'$ *with* $v < v'$.

[a] $T.a < T'.a'$ refers to the lexicographic ordering. $T.a = T'.a'$ is the way we write the node $(T.a, T'.a')$, because the node joins two attributes.

**Definition 4.3.4** (Proto-Rule)**.** *Given a path* $P$ *in a constraint graph, a proto-rule is the conjunction of atoms obtained as follows:*

1. *Create a conjunction with one atom for each table mentioned in* $P$*, each with fresh variables.*
2. *Construct the replacement relation* $R$*, which contains* $R(T.i, T'.i')$ *iff* $T.i = T'.i'$ *is in* $P$*.*
3. *For each attribute* $T.i$*, replace its variable by the variable of the attribute* $min\{T'.i' \mid R^*(T'.i', T.i)\}$*.*

Having defined how a path in the constraint graph lifts to a conjunction of atoms (a proto-rule), we next summarise paths by a simple size measure.

**Definition 4.3.5** (Proto-Rule Length)**.** *For a path* $P$ *in the constraint graph, the proto-rule length is the* number of nodes *in* $P$ *(Figure 4.5).*

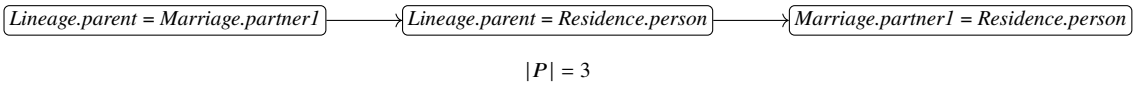


**Figure 4.5:** Proto-rule length: number of nodes on the path in the constraint graph.

The following example instantiates the lifting and replacement process on the running toy graph.

**Example 4.3.2.** *Consider the following path in the constraint graph, which contains two nodes:*

$$Lineage.parent = Marriage.partner1$$

$$Lineage.parent = Residence.person$$

*We obtain the following conjunction:*

$$Lineage(p, c) \wedge Marriage(p_1, p_2) \wedge Residence(p_3, l, s)$$

*We obtain the following replacement relation R:*

$$Lineage.parent \rightarrow_R Marriage.partner1 \rightarrow_R Residence.person$$

*This means that all the variables that are concerned by this replacement relation have to be replaced by the variable of Lineage.parent, i.e. by $p$:*

$$Lineage(p, c) \wedge Marriage(p, p_2) \wedge Residence(p, l, s)$$

To quantify how often a proto-rule holds in the data, we define its *count*.

**Definition 4.3.6** (Proto-Rule Count)**.** *The* count *of a proto-rule is the number of possible bindings of the variables so that the conjunction of the proto-rule holds on the database.*

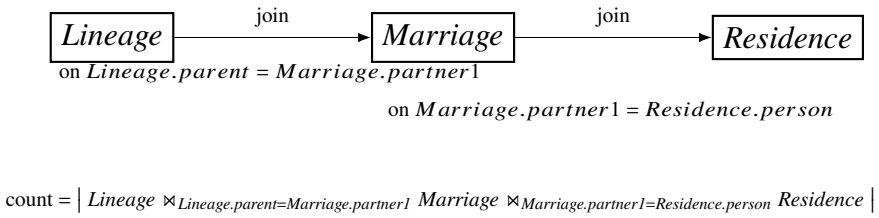


**Figure 4.6:** Proto-rule count: cardinality of the join result with standard conjunctive `ON` conditions (using single equality predicates) as illustrated above.

**Example 4.3.3.** *The count of the proto-rule in Example 4.3.2 (illustrated in Figure 4.6) is the number of elements in the result of joining Lineage, Marriage, and Residence on their respective keys, which is three. The corresponding SQL query is:*

```
SELECT
    LINEAGE.Parent,
    LINEAGE.Child,
    MARRIAGE.Partner1,
    MARRIAGE.Partner2,
    RESIDENCE.Person,
    RESIDENCE.Location,
    RESIDENCE.State
FROM
    LINEAGE
JOIN
    MARRIAGE ON LINEAGE.Parent = MARRIAGE.Partner1
JOIN
    RESIDENCE ON MARRIAGE.Partner1 = RESIDENCE.Person;
```

Proto-rules become TGDs by splitting them into body and head (Figure 4.7):

**Definition 4.3.7** (Split)**.** *A split of a proto-rule is a partition of the set of tables of the proto-rule into two sets, the head and the body. A split proto-rule is a pair of a proto-rule and one of its splits* $\langle B, H \rangle$*. A split proto-rule can be written as a TGD:*

$$\forall \vec{x}, \vec{y} : \phi(\vec{x}, \vec{y}) \Rightarrow \exists \vec{z} : \psi(\vec{x}, \vec{z})$$

*where* $\phi, \psi$ *is a partition of the atoms of the proto-rule so that all atoms with table names of* $B$ *appear in* $\phi$*. Each variable* $\alpha$ *of the proto-rule is assigned as follows: If* $\alpha$ *belongs to at least one table in the body and to one table in the head, then* $\alpha \in \vec{x}$*. If* $\alpha$ *belongs only to body tables, then* $\alpha \in \vec{y}$*, otherwise* $\alpha \in \vec{z}$*.*

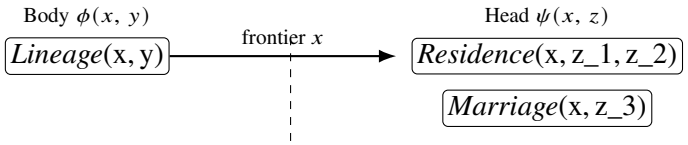


**Figure 4.7:** Split of a proto-rule: atoms partitioned into body $\phi$ and head $\psi$ with shared frontier variables $\vec{x}$ and existentials $\vec{z}$.

**Example 4.3.4.** *Consider again the proto-rule of Example 4.1.1. By splitting this proto-rule into the body {Lineage} and the head {Residence, Marriage}, we obtain:*

$$\begin{aligned} \forall x_1, y_1 : \mathit{Lineage}(x_1, y_1) \Rightarrow \exists z_1, z_2, z_3 : \\ \mathit{Residence}(x_1, z_1, z_2) \wedge \mathit{Marriage}(x_1, z_3) \end{aligned}$$

Having introduced a quantitative notion of frequency (the proto-rule count) that we will later use to prune infrequent candidates, we now connect representation and semantics: splitting a proto-rule yields a TGD, and conversely the next proposition shows that every non-recursive TGD that joins only joinable attributes corresponds to some proto-rule in the constraint graph, independently of its count. This establishes the completeness of the graph-based representation for our class of TGDs.

**Proposition 4.3.1.** *For any non-recursive TGD (i.e., no table appears twice) that joins only joinable attributes, the constraint graph contains a proto-rule that gives rise to that TGD.*

*Proof.* Consider a target TGD of the form

$$\forall \vec{x}, \vec{y} : \psi(\vec{x}, \vec{y}) \Rightarrow \exists \vec{z} : \phi(\vec{x}, \vec{z}).$$

We apply the reverse of the instantiation process of Definition 4.3.7: We first construct the split into head and body by partitioning the set of tables that appear in the TGD, which is possible because no table appears in both body and head (non-recursivity). We can then reconstruct the nodes of the constraint graph: if two atoms $t(x_1, \ldots, x_n)$ and $t'(x'_1, \ldots, x'_{n'})$ of the target

TGD share a common variable $x_i = x'_i$, we derive the constraint graph node $t.i = t'.i'$. This constraint (or $t'.i' = t.i$) must appear in the constraint graph because the graph contains all pairs of joinable attributes, and the TGD connects only joinable attributes. Furthermore, if two pairs of constraints share an attribute (i.e., constraints of the form $t.i = t'.i'$ and $t.i = t''.i''$), these constraint pairs are linked in the constraint graph by construction. Since we consider only connected TGDs, each atom of the TGD is linked to at least one other atom in this manner, ensuring that all atoms are transitively connected within the graph. As a result, the subgraph formed by these linked constraints is itself connected and corresponds to a proto-rule, which is the conjunction of the TGD's atoms. □

This entails that mining all split proto-rules allows mining all non-recursive TGDs up to equivalence.

## 4.4 MATILDA

### 4.4.1 Our TGD Mining

Figure 4.8 illustrates the overall mining pipeline. Mining all TGDs faces a combinatorial explosion (Example 4.2.1). We therefore constrain both *structure* and *search*. Structurally, we restrict to connected patterns, enforce frontier-guardedness, disallow constants/arithmetic in rules, and require joins to follow a schema-level joinability relation. On the search side, we avoid enumerating a global partial order and instead prune candidates by joinability, minimum support and confidence, and a user-specified upper bound $k$ on atoms per rule (in line with prior work [19, 57, 60]).

**Foreign-Key Joinability.** We allow joins only when the columns are connected by a declared foreign key in the database. This makes the join map small and easy to build from the schema. It also matches how systems like AMIE treat relations with clear "points to" links. Because foreign key networks are usually sparse, this choice quickly cuts down how many joins we need to try. Most importantly, foreign keys capture the designer's intent about which rows refer to the same real-world thing (for example, `Order.customer_id` points to `Customer.id`). As a result, our joins are meaningful, and we avoid guesswork based on overlapping values or similar strings.

**Closed-World Semantics and `NULL`s.** We adopt the Closed World Assumption: Any ground atom not stored in the database is treated as *false*. Tuples containing `NULL` are excluded from matching and aggregation; consequently, we neither instantiate TGDs with incomplete bindings nor count partial instances. This ensures that all measured support and confidence rest on fully grounded facts.

### 4.4.2 Algorithm

We can now present our method, MATILDA. We first discuss the method for the case of non-recursive rules, and then extend it to recursive rules in Section 4.5.

```
def dfs(
    graph: ConstraintGraph,
    max_length: int = 3,
    start_node: JoinableAttributes = None,
    visited: set[JoinableAttributes] = set(),
```

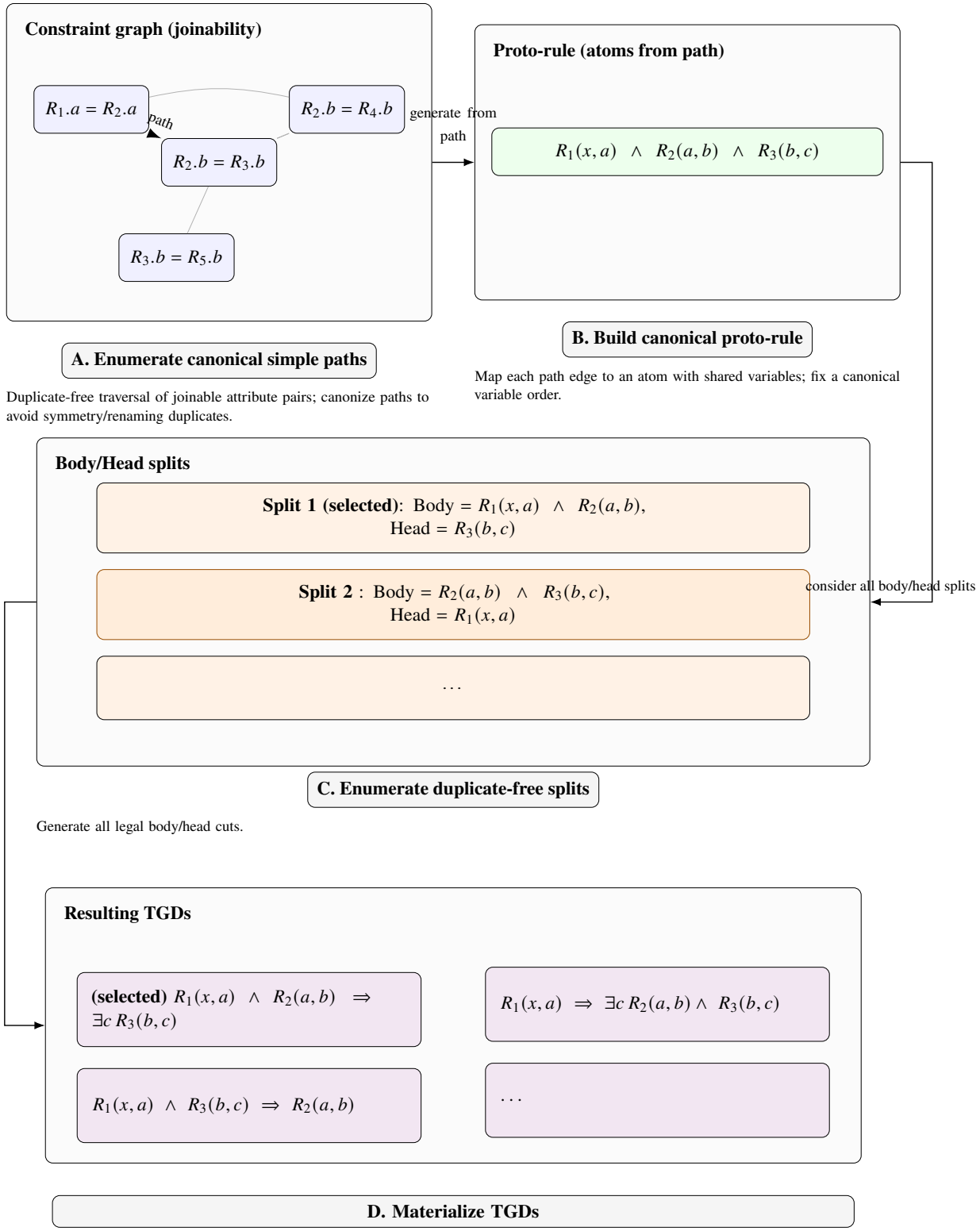


**Figure 4.8:** MATILDA mining pipeline: a canonical simple path in the constraint graph is projected into a proto-rule; duplicate-free body/head splits yield candidate TGDs.

```
    proto_rule: ProtoRule = [],
) -> Iterator[ProtoRule]:
    if start_node is None:
        for next_node in graph.nodes:
            yield from dfs(
                graph,
                max_length,
                next_node,
                visited,
                proto_rule,
            )
        return
    visited.add(start_node)
    proto_rule.append(start_node)
    yield proto_rule
    if len(proto_rule)>=max_length:
        return
    for next_node in graph.neighbors(start_node) - visited:
        if count(proto_rule+next_node)>0:
            yield from dfs(
                graph,
                max_length,
                next_node,
                visited,
                proto_rule,
            )
    visited.remove(start_node)
    proto_rule.pop()
```

**Algorithm 1:** Generate Proto-rules

The heart of MATILDA is Algorithm 1[1]. It takes a constraint graph and the maximum TGD length (the maximum number of constraints in the proto-rule) as input. It then outputs a continuous sequence of proto-rules. Initially, the algorithm iterates over all nodes in the graph, initiating a depth-first search (DFS) from each node. The current starting node is then added to the set of visited nodes, marking it as visited. This node is also appended to the proto-rule. If the proto-rule is smaller than the maximal length, the algorithm calls itself with the new proto-rule and each neighbor of the current node. We exclude neighbors that have been visited.
Upon returning from the recursive call, the neighbor is removed from the visited set and from the proto-rule list. The algorithm backtracks and explores alternative paths.
This approach systematically explores all paths in the graph and filters out proto-rules that are over the maximal length or that have a proto-rule count of zero.

> **Proposition 4.4.1.** *Given a database, Algorithm 1 generates all proto-rules that have a count greater than zero and a length smaller than or equal to the given* max_length.

[1]We show our algorithms in Python instead of pseudo-code to avoid ambiguity, help reproducibility, and ensure correspondence between our system and the paper.

*Proof.* Given a proto-rule $R$ that has a count greater than zero, and a length smaller than or equal to *max_length*, we show how Algorithm 1 finds this rule. Consider the lexicographically smallest node of $R$. Since Algorithm 1 iterates through all nodes of the constraint graph in Line 9, it will eventually consider this node. The algorithm then explores all neighbors of this node recursively. Since $P$ is a path in the graph, the algorithm will also explore $P$ recursively. Since $P$ has a length smaller than or equal to *max_length*, all prefixes of $P$ will also have a length smaller than or equal to *max_length*. Since $P$ has a count greater than zero, all prefixes of $P$ will also fulfill this condition. □

**Proposition 4.4.2.** *Given a database, Algorithm 1 generates proto-rules without duplicates.*

*Proof.* Given a constraint graph $CG$, let us assume that Algorithm 1 generates the same proto-rule twice, i.e., it generates two equivalent proto-rules $PR_1$ and $PR_2$. We further note that Algorithm 1 visits each path in the constraint graph exactly once: each node is added at most once per branch thanks to the `visited` set, and edges are followed only in the forward direction of the current path. Consequently, two different branches cannot traverse the same sequence of nodes in the same order; no duplicate proto-rule is emitted. □

Each proto-rule is then split into body and head (Definition 4.3.7) by Algorithm 2.

```
def split_proto_rule(proto_rule: ProtoRule):
    tables = extract_tables(proto_rule)
    return [ (body, tables-body)
             for body in powerset(tables)
             if len(body)<=len(tables)/2
             and len(body)>0 ]
```

**Algorithm 2:** Generate Splits

This algorithm takes as input a proto-rule and outputs a set of splits. The algorithm starts by extracting the tables involved in the proto-rule (`extract_tables` function) and then iterates over all possible subsets of the tables (`powerset` function). Each subset becomes a body, and its complement becomes a head. We do not return splits where the body is empty. We also do not return splits where the body has more atoms than the head. This is because our metrics are symmetric (Proposition 2.2.1).

**Support and confidence thresholds.** MATILDA uses two user-defined thresholds, *support* and *confidence*, to filter candidate TGDs during enumeration. The *support* threshold discards proto-rules whose underlying join graph occurs fewer times than the specified value;
A *confidence threshold* is used to filter candidate TGDs (Tuple-Generating Dependencies), discarding those whose conditional probability falls below the specified value. The choice of support and confidence thresholds involves a critical trade-off. *Lower thresholds* increase recall (the number of discovered TGDs) but can dramatically expand the search space, making the process more computationally intensive. Conversely, *higher thresholds* risk missing valid TGDs that have lower support or confidence. While a higher threshold yields a smaller, more manageable set of results,

it's important to understand its effect on performance. It primarily acts as a *filter* on the candidate set. It doesn't necessarily speed up the initial discovery algorithm but rather prunes the final output, which can expedite subsequent analysis. A common practice is to start with moderate settings (e.g., a support of 5% and a confidence of 0.8) and then relax these values if the resulting set of TGDs is too sparse.

We use these algorithms in our main algorithm, Algorithm 3. This algorithm takes as input the maximal TGD length threshold and a mapping between joinable attributes. This mapping contains, for each attribute, a set of attributes with which it can be joined (symmetrically). The algorithm outputs pairs of TGDs with their metrics (support and confidence). This works by first creating the constraint graph (Definition 4.3.3). Then, the algorithm runs through all proto-rules and through all splits of the proto-rule. For each split proto-rule, we compute the confidence and support (Definitions 2.2.2 and 2.2.3). Since our metrics are symmetric (Proposition 2.2.1), we can generate two rules for each split, as per Definition 4.3.7.

```
def MATILDA(
    joinable:Dict[Attribute, [Attribute]],
    max_length : int = 3
    ):
    graph = create_constraint_graph(joinable)
    for proto_rule in dfs(graph,max_length):
        if count(proto_rule) > 0:
            for split in split_proto_rule(proto_rule):
                metrics = comp_metrics(proto_rule, split)
                if metrics.support > 0:
                    yield (
                    createTGD(proto_rule, split),
                    metrics
                    )
                if metrics.swap().support > 0:
                    yield (
                    createTGD(proto_rule, split.swap()),
                    metrics.swap()
                    )
```

**Algorithm 3:** MATILDA

### 4.4.3 Theoretical Analysis

**Correctness.** From Proposition 4.3.1, Proposition 4.4.1, and the fact that Algorithm 1 generates all proto-rules that have a count greater than zero and a number of constraints less than *max_length* we directly have:

> **Proposition 4.4.3.** *Algorithm 3 generates all non-recursive TGDs that hold in the given database and that have a proto-rule with a non-zero count and a number of constraints smaller than or equal to* max_length*, without generating duplicate TGDs.*

*Proof.* We show that MATILDA (Algorithms 1–3) enumerates every admissible non-recursive TGD exactly once.
**Completeness** Fix a TGD $R$ has a proto-rule with a non-zero count and a number of constraints smaller than or equal to *max_length*. Proposition 4.3.1 yields a path $P$ in the constraint graph whose proto-rule reduces to $R$. Since $\text{count}(P) > 0$ and $|P| \leq \textit{max\_length}$, Algorithm 1 outputs $P$ (Proposition 4.4.1). Algorithm 2 then considers every admissible body-head split of the tables of $P$; among them is a split $S$ that matches $(\text{body}(R), \text{head}(R))$ up to symmetry. When Algorithm 3 processes $\langle P, S\rangle$, the support test in line 10 succeeds (since $R$ holds in the data), so lines 11-18 emit $R$. Thus $R$ is produced.
**Soundness** Any rule emitted by Algorithm 3 (i) originates from a proto-rule produced by Algorithm 1, and hence obeys the length and positive-count constraints, and (ii) passes the support check in line 10. Therefore, every output rule satisfies the required properties.
**Uniqueness** Assume two identical TGDs were emitted. Then these cannot have distinct proto-rules as Algorithm 1 outputs each proto-rule once (Proposition 4.4.2). The TGDs cannot have different splits either, because Algorithm 2 keeps only one of $\langle B, H\rangle$ and $\langle H, B\rangle$ ($|B| \leq |H|$), and Algorithm 3 outputs at most one orientation per split. Hence, no duplicates can occur. □

**Complexity.** To determine the complexity of Algorithm 3, we focus on the number of *split proto-rules* generated based on the number of joinable attributes $n$. We decompose the complexity into:

1. the number of start nodes,
2. the number of proto-rules generated from each start node,
3. and the number of possible splits for each proto-rule.

There are $n$ start nodes (one per pair of joinable attributes). In the worst case, every pair of joinable attributes is joinable with every other one, making the constraint graph a complete acyclic orientation with $\binom{n}{2}$ edges. From a single start node, the number of proto-rules (simple paths) that can be generated is $2^{n-1} - 1$: Any subset of the remaining $n - 1$ nodes, taken in increasing order, yields exactly one path. Summing over all starts does not multiply this by $n$ (each path has a unique smallest node), so the total number of distinct proto-rules across all starts is $2^n - n - 1$.
For a proto-rule $r$, let $T(r)$ be the number of tables used by $r$. In the worst case, $T(r)$ can be as large as $n$, and each proto-rule can be split in up to $2^{T(r)}$ ways; thus we take the upper bound $2^n$ per rule. Multiplying these factors (Figure 4.9) gives the following worst-case complexity:

$$O\big((2^n - n - 1) \times 2^n\big) = O(2^{2n}).$$

For comparison, the naive approach relies on generating all partitions of $n$ attributes (which has Bell-number complexity) and then enumerating all splits (up to $2^n$). In practice, the naive approach quickly becomes infeasible: Already for $n = 13$, it would require at least $3 \times 4{,}213{,}597$ SQL queries (one query for each of prediction, coverage, and support). With 0.01s per query, it would not finish in a day. By contrast, our method can prune on two fronts:

(1) it considers only *joinable* attributes, and

(2) it discards proto-rules with insufficient count.

These two pruning strategies allow our approach to run in practice.

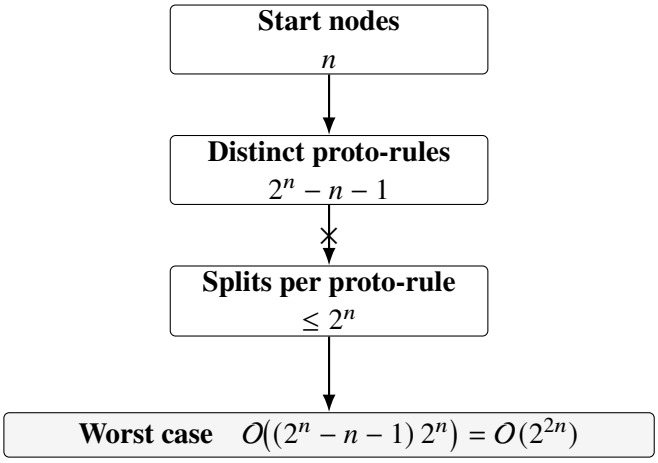


**Figure 4.9:** MATILDA worst-case complexity, split proto-rule enumeration.

## 4.5 MATILDA with recursive TGDs

In this section, we adapt MATILDA to mine recursive TGDs, i.e., TGDs that contain the same table more than once. Mining recursive TGDs requires adapting the previous definitions of attribute, proto-rule, split proto-rule, and TGD metrics.
We start with the definition of qualified tables and attributes (Figure 4.10):

**Definition 4.5.1** (Qualified Table and Attribute)**.** *A* qualified table *is a table with a natural number called the table occurrence i, written $t_i$. A* qualified attribute *is a triple of a table t, a table occurrence i, and one of its attributes a, written $t_i.a$. Lexical order defines a total order on qualified tables and on qualified attributes.*

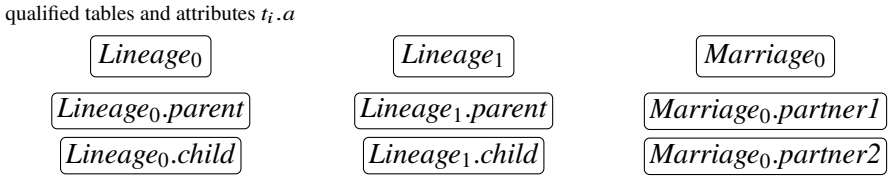


**Figure 4.10:** Qualified tables $t_i$ and qualified attributes $t_i.a$: each occurrence index $i$ denotes a distinct use of table $t$; lexical order applies on $(t, i, a)$.

In what follows, we write TGDs with qualified tables and attributes, so that each atom with the same table name as another atom has a different table occurrence.

**Example 4.5.1** (Qualified tables and attributes.)**.** *Let Lineage(parent, child) and Marriage(partner1, partner2). Two occurrences of Lineage are written $Lineage_0$ and $Lineage_1$. Qualified attributes include $Lineage_0.parent$, $Lineage_1.child$, and $Marriage_0.partner2$. Under lexical order on $(t, i, a)$ we may have, e.g., $Lineage_0.parent < Lineage_0.child < Lineage_1.parent < Marriage_0.partner1$.*

**Definition 4.5.2** (N-Recursive Constraint Graph)**.** *An N-Recursive Constraint Graph (illustrated in Figure 4.11) for a natural number N is a directed unlabeled acyclic graph (similar to Definition 4.3.3), where*

- *A node* $(T_i.a, T_j'.a')$ *is a pair of joinable qualified attributes such that* $T_i.a < T_j'.a'$ *and* $i, j \leq N$
- *An edge* $(v, v')$ *links two table-linked nodes* $v$ *and* $v'$ *with* $v < v'$.

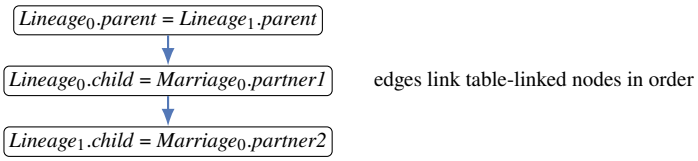


**Figure 4.11:** $N$-recursive constraint graph: nodes are equalities between *qualified* attributes; edges connect table-linked nodes under a global order.

**Example 4.5.2** ($N$-recursive constraint graph.)**.** *Let* $N = 2$ *and assume person-typed attributes are joinable. For Lineage(parent, child) and Marriage(partner1, partner2),* $G_2$ *contains nodes such as* $(Lineage_0.parent, Lineage_1.parent)$, $(Lineage_0.child, Marriage_0.partner1)$, *and* $(Lineage_1.child, Marriage_0.partner2)$. *Because* $(Lineage_0.parent, Lineage_1.parent) < (Lineage_0.child, Marriage_0.partner1)$ *and both involve* $Lineage_0$, *an edge connects the two.*

The recursive graph can then generate proto-rules (Figure 4.12):
In the recursive setting, a proto-rule is still the pattern induced by a path, but now over *qualified* attributes (table occurrences $t_i$). The index condition (indices start at 0 and are contiguous) collapses equivalent renamings of occurrences, reducing duplicates without removing any expressive rule.

**Definition 4.5.3** (Recursive Proto-Rule)**.** *A recursive proto-rule is a proto-rule (as defined in Definition 4.3.4) that corresponds to a path P in a recursive constraint graph. This path must satisfy the condition that for every qualified table* $T_i$ *in P, if* $i > 0$, *then for every* $j \in [0, i-1]$, *there exists a qualified table* $T_j$ *in P.*

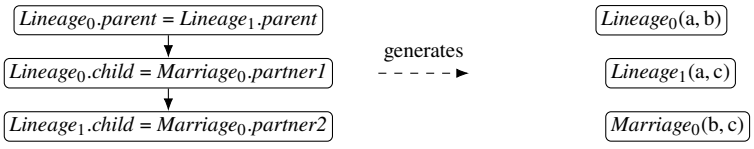


**Figure 4.12:** Recursive proto-rule: a path in the $N$-recursive constraint graph lifts to atoms with identified variables across qualified table occurrences.

The condition on the indexes of qualified attributes ensures that indexes start at 0 and are given consecutively. We can now convert paths in that graph to proto-rules and to TGDs:
With multiple occurrences of the same table, naively splitting can yield symmetric duplicates or trivial redundancies. We therefore discard *redundant* proto-rules (repeating the same table with identical non-dangling variables) and then apply the same body–head split as in the non-recursive case to obtain clean, unique TGDs.

**Definition 4.5.4** (Recursive split proto-rule)**.** *A* recursive proto-rule *is a proto-rule obtained from a recursive constraint graph by Definition 4.3.4. A* dangling variable *of a proto-rule is a variable that appears at most once. A recursive proto-rule is* non-redundant*, if it does not contain two atoms that (1) have the same table name, irrespective of table occurrence, and (2) agree on all non-dangling variables. Non-redundant proto-rules can be split and translated into TGDs according to Definition 4.3.7.*

**Example 4.5.3** (Redundancy and admissible splits.)**.** *Consider variables $a, b, c$ and the proto-rule:*

$$\begin{aligned} &\mathit{Lineage}_0(\mathit{parent} = a, \mathit{child} = b), \quad \mathit{Lineage}_1(\mathit{parent} = a, \mathit{child} = b), \\ &\qquad\qquad \mathit{Marriage}_0(\mathit{partner1} = b, \mathit{partner2} = c). \end{aligned}$$

*Here, $a, b$ are non-dangling (each occurs $\geq 2$ times). The two Lineage atoms have the same table name and agree on all non-dangling variables $(a, b)$, so the proto-rule is* redundant *and must be refined before splitting.*
*By contrast, with*

$$\begin{aligned} &\mathit{Lineage}_0(\mathit{parent} = a, \mathit{child} = b), \quad \mathit{Lineage}_1(\mathit{parent} = a, \mathit{child} = c), \\ &\qquad\qquad \mathit{Marriage}_0(\mathit{partner1} = b, \mathit{partner2} = c), \end{aligned}$$

*the Lineage atoms disagree on the non-dangling variable bound to child ($b \neq c$), so the proto-rule is* non-redundant*. An admissible split yields the TGD:*

$$\underbrace{\mathit{Marriage}_0(\mathit{partner1} = b, \mathit{partner2} = c) \ \wedge \ \mathit{Lineage}_0(\mathit{parent} = a, \mathit{child} = b)}_{\mathit{body}} \Rightarrow \underbrace{\mathit{Lineage}_1(\mathit{parent} = a, \mathit{child} = c)}_{\mathit{head}}.$$

Redundant proto-rules take the form $\mathit{Lineage}_1(\mathit{parent}{=}x) \Rightarrow \mathit{Lineage}_2(\mathit{parent}{=}x)$. Such proto-rules can be further refined in Algorithm 1, but they will not be used to generate TGDs. We thus obtain:

**Definition 4.5.5** (Matilda-N)**.** *Matilda-N (for a natural number $N > 0$) is Algorithm 3, in which the constraint graph is replaced by a recursive constraint graph of degree $N$.*

**Example 4.5.4** (Recursive proto-rule from a path.)**.** *Using the nodes from the previous example, the path*

$$(Lineage_0.parent, Lineage_1.parent) \rightarrow$$
$$(Lineage_0.child, Marriage_0.partner1) \rightarrow$$
$$(Lineage_1.child, Marriage_0.partner2)$$

*induces the proto-rule*

$$\begin{aligned} Lineage_0.parent = Lineage_1.parent \\ \wedge\ Lineage_0.child = Marriage_0.partner1 \\ \wedge\ Lineage_1.child = Marriage_0.partner2. \end{aligned}$$

*Since $Lineage_1$ appears only together with $Lineage_0$ (indexes are consecutive from 0), the index condition holds.*

### 4.5.1 Theoretical Analysis

Having introduced the recursive data structures and the notion of recursive proto-rules, we now outline the formal properties of MATILDA-*N*. We begin with how the procedure controls duplicate TGDs, then state a completeness guarantee under a bounded rule length for non-redundant *N*-recursive TGDs, and finally comment on implementation aspects that matter in practice. This provides a bridge from the construction to its guarantees and practical implications.

**Duplicate control** MATILDA-N can yield the same TGD twice.

**Example 4.5.5** (Duplicates.)**.** *The following proto-rules are equals:*

***Proto-Rule 1***

$$Lineage_0.parent = Lineage_1.parent \quad (4.1)$$
$$Lineage_0.child = Marriage_0.partner1 \quad (4.2)$$
$$Lineage_1.child = Marriage_0.partner2 \quad (4.3)$$

***Proto-Rule 2***

$$Lineage_0.parent = Lineage_1.parent \quad (4.4)$$
$$Lineage_0.child = Marriage_0.partner2 \quad (4.5)$$
$$Lineage_1.child = Marriage_0.partner1 \quad (4.6)$$

*These proto-rules look different but give rise to two logically equivalent TGDs:*

$$Marriage_0(partner1{=}b, partner2{=}c); \wedge; Lineage_1(parent{=}a, child{=}c) \Rightarrow Lineage_0(parent{=}a, child{=}b) \quad (4.7)$$
$$Marriage_0(partner1{=}b, partner2{=}c); \wedge; Lineage_0(parent{=}a, child{=}b) \Rightarrow Lineage_1(parent{=}a, child{=}c) \quad (4.8)$$

This issue appears when the maximal TGD length is greater than two. To avoid yielding the same TGD twice, we cache the results and filter out duplicate TGDs.

**Completeness.** MATILDA-N is complete in the following sense:

**Proposition 4.5.1** (Completeness of MATILDA-$N$)**.** *Fix $N \geq 1$ and a maximal number of atoms $\theta \in \mathbb{N}$. For any schema with primary keys and every* non-redundant *$N$-recursive TGD*

$$\tau \;=\; \forall \bar{x}, \bar{y}\ \phi(\bar{x}, \bar{y}) \;\Rightarrow\; \exists \bar{z}\ \psi(\bar{x}, \bar{z}), \qquad |\phi \cup \psi| \leq \theta,$$

*MATILDA-N outputs a TGD that is logically equivalent to $\tau$.*

*Proof.* **(i) Path enumeration.** Since $\tau$ is $N$-recursive, every atom uses a qualified table $T_i$ with $0 \leq i < N$ (Definition 6.1). Traversing the equalities of $\tau$ in lexicographic order yields a path $p_\tau$ in the $N$-recursive constraint graph $G_N$ (Definition 6.2) that fulfills the index-density condition of Definition 6.3. Algorithm 1 (DFS) enumerates $p_\tau$ as long as $|p_\tau| \leq \theta$.
**(ii) Proto-rule creation.** Line 4 of Algorithm 1 converts $p_\tau$ into a proto-rule $r_\tau$ whose atoms are exactly those of $\tau$; non-redundancy carries over from $\tau$ to $r_\tau$.
**(iii) Splitting.** Algorithm 2 iterates over all admissible body-head partitions of $r_\tau$; one split $S_\tau$ coincides with $(\phi, \psi)$ (or its symmetric counterpart).
**(iv) Emission.** When Algorithm 3 (MATILDA-$N$, Definition 6.5) processes $\langle r_\tau, S_\tau \rangle$, the support test in line 10 succeeds, so the resulting TGD is yielded. Thus at least one rule equivalent to $\tau$ is produced, proving completeness.

□

**Implementation.** MATILDA is implemented in Python, and does not use any external libraries except *SQLAlchemy* for interaction with the databases and *psutil* for monitoring purposes. The choice of Python distinguishes MATILDA from most of the other current rule mining approaches, which are implemented in Java or Prolog [57, 61, 62, 60]. However, data scientists typically work with Python nowadays.

## 4.6 Experiments

The goal of our experimental evaluation of MATILDA is threefold: first, to demonstrate that our algorithm scales to databases that are out of reach for our competitors; second, to show that it mines all rules that competing methods mine; and third, to evaluate the portion of TGDs that go beyond inclusion dependencies and Horn rules. We also examine the scalability and adaptability of MATILDA in various ablation studies.

### 4.6.1 Setup

**Databases.** MATILDA is designed to take a database as input. We chose the relational databases from the relational data project[2] [63], which are well-documented and widely used in academic research [64, 65, 66]. We removed databases that could not be converted from the MariaDB format to SQLite using the tool mysql-to-sqlite3[3], and databases that contained image data. The remaining

[2] https://relational-data.org/
[3] https://github.com/techouse/mysql-to-sqlite3

databases are shown in Table 4.1, covering a wide variety of table arities, sizes, and number of tables. We consider two attributes joinable if one is a foreign key of the other.

| Database | Tables | Rows | Columns | Triples | Score FK/col |
|---|---|---|---|---|---|
| **Biodegradability** | 5 | 21875 | 14 | 48998 | 0.123 |
| **Bupa** | 9 | 2762 | 16 | 2069 | 0.085 |
| **Carcinogenesis** | 6 | 27570 | 23 | 92486 | 0.456 |
| **ccs** | 6 | 422868 | 29 | 473942 | 0.789 |
| **CDESchools** | 3 | 29481 | 90 | 872104 | 0.234 |
| **Chess** | 2 | 2052 | 45 | 28766 | 0.678 |
| **CiteSeer** | 3 | 113760 | 6 | 3311 | 0.345 |
| **classicmodels*** | 8 | 3864 | 59 | 14071 | 0.567 |
| **ConsumerExp.** | 3 | 2241548 | 24 | 15479875 | 0.876 |
| **CORA** | 3 | 57884 | 6 | 2707 | 0.234 |
| **Countries** | 4 | 21118 | 67 | 516839 | 0.456 |
| **CraftBeer** | 2 | 2968 | 11 | 16133 | 0.678 |
| **cs*** | 8 | 563 | 43 | 3279 | 0.789 |
| **Dallas** | 3 | 812 | 27 | 6623 | 0.345 |
| **DCG*** | 2 | 8258 | 5 | 8257 | 0.567 |
| **Dunur** | 17 | 440 | 34 | 275 | 0.876 |
| **Elti** | 11 | 1352 | 22 | 1080 | 0.234 |
| **Hockey** | 22 | 96403 | 300 | 264268 | 0.456 |
| **KRK*** | 1 | 1000 | 8 | 7000 | 0.678 |
| **medical** | 3 | 15899 | 64 | 263023 | 0.789 |
| **Mesh** | 29 | 2700 | 37 | 1704 | 0.345 |
| **MuskSmall** | 2 | 554 | 170 | 79584 | 0.567 |
| **mutagenesis** | 3 | 10324 | 14 | 36241 | 0.876 |
| **nations** | 3 | 11004 | 118 | 33171 | 0.234 |
| **NBA** | 5 | 1221 | 72 | 17446 | 0.456 |
| **NCAA** | 9 | 201555 | 106 | 2390826 | 0.678 |
| **Pima** | 9 | 6912 | 18 | 13056 | 0.789 |
| **PTE** | 38 | 29762 | 76 | 89120 | 0.345 |
| **pubs*** | 11 | 255 | 64 | 945 | 0.567 |
| **Pyrimidine** | 2 | 296 | 13 | 2071 | 0.876 |
| **SAP*** | 5 | 4130447 | 45 | 15366486 | 0.234 |
| **SAT** | 36 | 19867 | 69 | 560 | 0.456 |
| **SFScores** | 3 | 66153 | 25 | 370675 | 0.678 |
| **Shakespeare** | 4 | 35234 | 19 | 149609 | 0.789 |
| **Toxicology** | 4 | 49239 | 11 | 124040 | 0.345 |
| **tpcc*** | 9 | 593933 | 93 | 3893757 | 0.567 |
| **UTube** | 2 | 2735 | 7 | 7110 | 0.876 |
| **Walmart** | 4 | 4628497 | 27 | 4986950 | 0.234 |
| **WebKP** | 3 | 80592 | 6 | 876 | 0.456 |
| **world** | 3 | 5411 | 24 | 21629 | 0.678 |

**Table 4.1:** Database statistics. *Triples* is the number of *distinct* RDF triples obtained by applying the W3C *Direct Mapping* from relational data to RDF (RDB2RDF DM), counting only non-NULL attribute values; we include class assertions (`rdf:type`) and attribute/object-property triples and deduplicate identical triples before counting. *Score FK/col* is, for each database, the average over tables of the ratio of declared foreign-key columns to total columns, i.e., $\frac{1}{|T|}\sum_{t\in T}\frac{\#\mathrm{FK}(t)}{\#\mathrm{cols}(t)}$; higher values indicate more referentially dense schemas. An asterisk (*) marks synthetic data. The *Triples* column reflects the size of the AMIE knowledge base constructed from the direct mapping.

**Choice of DMBS** We use SQLite because it does not require the installation and configuration of a server to replicate our experiments.

**Competitors.** We compare our approach to the following competitors: Spider (Metanome project)[67] is a state-of-the-art inclusion dependency mining system; AMIE 3[57] is the newest incarnation of the AMIE Horn rule mining system; and Popper [60] is, at the time of writing, the fastest ILP system – ILP systems being particularly notable for mining Horn rules. All systems (Popper, Spider, AMIE, and MATILDA) mine both patterns with a 100% confidence and

| Database | Popper | | | Spider | | | AMIE 3 | | | MATILDA-3 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | j'ble | results | time | j'ble | results | time | j'ble | results | time | results | Cov. | time |
| **Biodegradability** | 0 | 2 | 29m:14.10s | 4 | 7 | 1.59s | - | 0 | 0.8s | 4 | 100% | 0.97s |
| **Bupa** | 1 | 24 | 1.15s | 6 | 58 | 2.50s | - | 0 | 0.7s | 9 | 100% | 0.86s |
| **Carcinogenesis** | 0 | 2 | 2.02s | 1 | 25 | 1.97s | 0 | 25 | 7.71s | 2 | 100% | 0.27s |
| **ccs** | - | 0 | 0.00s | - | 233 | 30.44s | 0 | 2 | 53.09s | 0 | - | 0.009s |
| **CDESchools** | - | - | mem-out | 2 | 55 | 18.47s | - | - | mem-out | 4 | 100% | 1.77s |
| **Chess** | - | - | mem-out | 1 | 46 | 1.93s | 0 | 429 | 14.77s | 2 | 100% | 0.47s |
| **CiteSeer** | - | 0 | 42.02s | 2 | 2 | 2.68s | - | 0 | 4.3s | 2 | 100% | 5.48s |
| **classicmodels** | - | - | mem-out | 6 | 132 | 3.31s | 0 | 27 | 2.19s | 14 | 100% | 1.96s |
| **ConsumerExp.** | - | - | mem-out | 0 | 31 | 1m:16.63s | - | - | mem-out | 2 | - | 3m:3.54s |
| **CORA** | 0 | 2 | 20.64s | 4 | 6 | 1.65s | - | 0 | 2.6s | 6 | 100% | 2.64s |
| **Countries** | - | - | mem-out | - | - | mem-out | 0 | 67573 | 4m:30.59s | 0 | - | 8.81s |
| **CraftBeer** | 0 | 2 | 35.06s | 1 | 2 | 1.52s | 0 | 1 | 2.21s | 2 | 100% | 0.58s |
| **cs** | 0 | 4 | mem-out | 8 | 32 | 1.46s | 0 | 33 | 1.41s | 27 | 100% | 3.90s |
| **Dallas** | - | 0 | 0.01s | - | 8 | 1.81s | 0 | 3 | 1.34s | 0 | - | 0.009s |
| **DCG** | - | 0 | 0.59s | - | 4 | 1.34s | - | 0 | 1.04s | 0 | - | 0.0089s |
| **Dunur** | 1 | 35 | 3.91s | 20 | 164 | 1.69s | - | 0 | 0.61s | 40 | 100% | 3.39s |
| **Elti** | 2 | 37 | 1.81s | 12 | 78 | 1.54s | - | 0 | 0.64s | 24 | 100% | 1.82s |
| **Hockey** | - | - | mem-out | 39 | 11172 | 37.16s | - | - | mem-out | - | - | timeout |
| **KRK** | - | 0 | 0.00s | - | 30 | 2.27s | 0 | 58 | 1.23s | 0 | - | 0.0089s |
| **medical** | - | - | mem-out | 1 | 116 | 6.99s | 0 | 123 | 15.26s | 4 | 100% | 2.98s |
| **Mesh** | - | 0 | 0.00s | 0 | 64 | 1.14s | - | 0 | 0.15s | 0 | - | 1.11s |
| **MuskSmall** | - | - | mem-out | 2 | 62 | 3.76s | 0 | 1864 | 29.62s | 2 | 100% | 0.30s |
| **mutagenesis** | 0 | 2 | 20.93s | 2 | 6 | 0.96s | 0 | 6 | 1.81s | 2 | 100% | 0.17s |
| **nations** | - | - | mem-out | 2 | 12444 | 5.66s | 0 | 203243 | 10m:23.24s | 2 | 100% | 0.33s |
| **NBA** | - | - | mem-out | 8 | 392 | 1.91s | 0 | 2422 | 7.09s | 25 | 100% | 0.98s |
| **NCAA** | - | - | mem-out | 10 | 879 | 10.31s | 0 | 903 | 17m:7.14s | 28 | 100% | 1m:1.35s |
| **Pima** | 0 | 12 | 0.79s | 10 | 72 | 1.25s | - | 0 | 1.57s | 10 | 100% | 1.47s |
| **PTE** | 6 | 18 | 35m:22.16s | 22 | 300 | 4.83s | - | 0 | 3.53s | 33 | 100% | 10.45s |
| **pubs** | - | - | timeout | 10 | 213 | 0.74s | 0 | 14 | 0.70s | 22 | 100% | 3.02s |
| **Pyrimidine** | - | 0 | 0.00s | - | 26 | 0.52s | 0 | 286 | 1.67s | 0 | - | 0.014s |
| **SAP** | - | - | mem-out | 1 | 41 | 31.69s | - | - | mem-out | 2 | 100% | 21.48s |
| **SAT** | 4 | 10 | 0.72s | 8 | 1639 | 2.59s | - | 0 | 9.54s | 20 | 100% | 3.19s |
| **SFScores** | - | 0 | 0.23s | - | 3 | 2.91s | 0 | 8 | 12.18s | 0 | - | 0.011s |
| **Shakespeare** | - | 0 | 0.46s | 2 | 13 | 3.48s | 0 | 7 | 6.26s | 2 | 100% | 0.28s |
| **Toxicology** | 0 | 6 | 4.96s | 2 | 10 | 2.55s | 0 | 3 | 8.20s | 4 | 100% | 2.64s |
| **tpcc** | - | - | mem-out | 0 | 639 | 6m:33.56s | 0 | 13 | 10m:5.66s | 0 | 100% | 7m:15.58s |
| **UTube** | - | 0 | 0.00s | 0 | 3 | 1.39s | - | 0 | 0.16s | 0 | - | 0.18s |
| **Walmart** | 1 | 4 | 0.17s | 2 | 88 | 26.79s | - | - | mem-out | 8 | 100% | 0.19s |
| **WebKP** | 0 | 2 | 14.45s | 4 | 6 | 2.22s | 0 | 2 | 3.66s | 6 | 100% | 2.75s |
| **world** | - | - | mem-out | 0 | 4 | 1.72s | 0 | 12 | 1.81s | 2 | - | 0.13s |

**Table 4.2:** Comparison of MATILDA-3 against other methods in terms of the number of results, execution time, and coverage. A value of "-" indicates that the corresponding data cannot be computed. The "j'ble" column gives the number of TGDs that join only joinable columns. The "Cov." column gives the percentage of the joinable TGDs of the competitors that MATILDA mines. As SPIDER is an approximate algorithm, not all inclusion dependencies it reports are actual inclusions.

approximate patterns with a confidence of less than 100%. Spider does not provide confidence or support metrics, Popper reports only support, and AMIE and MATILDA output both.
Each of our competitors required slight adaptations of our dataset. For Spider, we converted the database into CSV files, one for each table. We configured the Metanome framework to be used with the command-line interface extension. For AMIE, all tables had to be converted to binary relations. We used a direct mapping[4] to automatically convert the database into a knowledge base of such binary relations.

| **Database** | **Rule** | **Interpretation** |
|---|---|---|
| CraftBeer | ?a beers.ounces ?b => ?a beers.ibu ?b | The ounces attribute is sometimes equivalent to the ibu attribute (ounces are linked to bitterness). |
| Country | ?a geographic.county ?b => ?a geographic.region ?b | The county attribute is equivalent to the region attribute; some counties share the same name as the region. |
| UW | ?a person.yearsInProgram ?b => ?a person.hasPosition ?b | The number of years in a program often corresponds to a value representing the person's position. |
| Dallas | ?a incidents.subject_count ?b => ?a incidents.officer_count ?b | Subject counts are often the same as the officer count for each incident. |

**Table 4.3:** Example rules from AMIE3

For Popper, each table in the database is translated into three Prolog files:

- *bias.pl*: Encodes the database schema (including table arities) and configuration settings.
- *bk.pl*: Represents the entire database as background knowledge.
- *exs.pl*: Contains examples drawn from the table that has been selected as the target (i.e., the head of the generated Horn rules).

We thank the authors of Popper for their assistance with this transformation process.
To cover all potential rules, we set the maximum number of atoms in a rule body (configured in *bias.pl*) to the total number of tables in the database. In order for Popper to leverage joinability constraints, we remove any table from both *bias.pl* and *exs.pl* unless it shares at least one node in the constraint graph with the selected head table. (In this context, "sharing a node" means that there is at least one attribute linking the two tables in the constraint graph.) For instance, in our toy example, if no node in the constraint graph includes both an attribute from the "Marriage" table and an attribute from the "Lineage" table (the selected head), then all data from the "Marriage" table would be removed.
**Server Settings** All experiments were conducted on servers equipped with Intel® Xeon® CPU E5-2650 v2 @ 2.60GHz or similar processors. The setup comprised 15 servers, each configured with a time limit of two hours and a RAM usage limit of 15 GB per job. Each server was equipped with 512 GB of RAM. The computational runs were managed by a main server, utilizing bash scripts and `tmux`, and the data was curated using log extraction methods. All scripts are available on GitHub.

[4] https://www.w3.org/TR/rdb-direct-mapping/

| Database | Rule | Interpretation |
|---|---|---|
| NCAA | $\forall x_0 :$ Team_0(TeamId = $x_0$) $\Rightarrow \exists z_0 :$ Actions_0(GameId = $z_0$) $\wedge$ Game_0(GameId = $z_0$, Team2Id = $x_0$) | Every team $x_0$ must appear as the second team (Team2Id = $x_0$) in at least one game $z_0$, together with an action recorded for that game. |
| Dunur | $\forall x_0 :$ person_0(name = $x_0$) $\Rightarrow \exists z_0 :$ brother_0(name1 = $x_0$, name2 = $z_0$) $\wedge$ person_1(name = $z_0$) | Every person $x_0$ has at least one brother $z_0$ who is also a person (non-perfect rule). |
| University | $\forall x_0 :$ student_0(student_id = $x_0$) $\Rightarrow \exists z_0 :$ RA_0(prof_id = $z_0$, student_id = $x_0$) $\wedge$ prof_0(prof_id = $z_0$) | Every student $x_0$ has at least one professor $z_0$ such that $x_0$ is a research assistant to $z_0$, and $z_0$ is a valid professor. |
| CORA | $\forall x_0 :$ paper_0(paper_id = $x_0$) $\Rightarrow \exists z_0 :$ cites_0(cited_paper_id = $x_0$, citing_paper_id = $z_0$) $\wedge$ paper_1(paper_id = $z_0$) | Every paper $x_0$ is cited by at least one paper $z_0$. |
| PTE | $\forall x_0 :$ pte_drug_0(drug_id = $x_0$) $\Rightarrow \exists z_0 :$ pte_active_0(drug_id = $x_0$, is_active = $z_0$) $\wedge$ pte_number_0(binary = $z_0$) | Each drug $x_0$ has an active flag $z_0$ (in `pte_active_0`), and that same flag appears as a binary value in `pte_number_0`. |
| UW | $\forall x_0 :$ course(course_id = $x_0$) $\Rightarrow \exists z_0 :$ advisedBy(p_id_dummy = $z_0$) $\wedge$ taughtBy(p_id = $z_0$, course_id = $x_0$) | Every course $x_0$ has at least one person $z_0$ who both advises and teaches it. |
| Same-gen | $\forall x_0 :$ same_gen(name2 = $x_0$) $\Rightarrow \exists z_0 :$ parent(name1 = $x_0$, name2 = $z_0$) $\wedge$ person(name = $z_0$) | Entities marked as same generation have a parent–child relationship, revealing generational or hierarchical structures. |

**Table 4.4:** Examples of TGDs that only MATILDA mines. Only the first rule is a "perfect" one, whose constituent Horn rule has a confidence of 100%.

### 4.6.2 Results

**Effectiveness.** Table 4.2 shows the results of all mining systems on all databases. The Popper system fails on half of the databases with a memory overflow. This is not surprising, as ILP systems are typically designed for smaller datasets.
Spider runs on nearly all databases. It finds many inclusion dependencies, and on most databases, the inclusion dependencies it reports are correct (i.e., one column is indeed a subset of the other column, as reported). It finds only inclusion dependencies, not Horn rules (let alone TGDs).
AMIE runs on most databases, but finds only very few rules. This has two reasons: First, decomposing relations into binaries blows up rule length. For example, Married(a,b)$\wedge$Residence(a,l,s)$\Rightarrow$ Residence(b,l,s) has 2 body atoms on the relational schema but inflates to 5–7 binary atoms after triple encoding. Many such rules then exceed AMIE's rule-length budget L; raising L causes a combinatorial explosion in candidates and joins, so these rules remain effectively unreachable in practice. Second, the conversion to the triple format (id, relation, value) hinders AMIE from mining meaningful rules, as it restricts the system to patterns that directly associate an ID with an attribute.

MATILDA runs on all databases except *Hockey*. This is because the induced constraint graph on this dataset is extremely large (due to a dense foreign-key graph and many qualified pairs). On all other datasets, MATILDA reproduces all joinable patterns mined by the other approaches. Since it mines patterns that go beyond inclusion dependencies and the simplistic AMIE rules, MATILDA produces around 50% more patterns overall. Despite this, MATILDA runs roughly in the same time as the other approaches. We thus conclude that MATILDA is the only system that can find patterns beyond inclusion dependencies in larger databases.

**Coverage.** MATILDA mines all joinable correct inclusion dependencies that Spider mines, and all joinable rules that Popper mines. MATILDA does not mine all the rules that AMIE mines. This is because AMIE has no means to take into account joinability (even if we wanted to provide this information to AMIE). Therefore, as shown in Table 4.3, most rules it mines are spurious correlations of the form *relation(id=$x_0$, age=$x_1$)* $\Rightarrow$ *relation(id=$x_0$, count=$x_1$)* (objects of type *relation* have the same *age* as their *count*). Furthermore, manual inspection shows that, due to the decomposition into binary relations, almost all of AMIE's rules (99.91%) revolve around joins across a single table. However, some of them can be considered useful (such as the rule that the number of officers sent to an incident often equals the number of involved people). We show in our ablation study[5] that if we consider all columns that share a value joinable, MATILDA can mine all of AMIE's rules.

**Complex TGDs.** Out of the 1050 rules that MATILDA mined in total on all databases, a small proportion of 8% are TGDs that have more than one atom in the head (linked by an existential variable), i.e., patterns that our competitors cannot mine by design. Some examples are shown in Table 4.4. Of these, 44% are "perfect", i.e., they are of the form $\forall\vec{x},\vec{y} : \phi(\vec{x},\vec{y}) \Rightarrow \exists\vec{z} : a(\vec{y},\vec{z}) \wedge b(\vec{y},\vec{z})$, where $a(\vec{y},\vec{z})$ and $b(\vec{y},\vec{z})$ are atoms and there exists also a rule $\forall\vec{y},\vec{z} : a(\vec{y},\vec{z}) \Rightarrow b(\vec{y},\vec{z})$ with a confidence of 1. Such rules are correct and interesting, because they confirm type constraints (see first example in Table 4.4). However, they could in principle be reconstructed from the individual Horn rules of which they are composed, in the way outlined at the end of Section 4.8. The remaining 56% of non-Horn rules express type constraints that do not hold with a confidence of 100% (see the other examples in Table 4.4), meaning that their confidence cannot be computed from the confidences of the constituent rules (Section 2.2). All of these non-perfect rules are of the same shape: They vary mainly in the relation names (sister instead of brother, etc.).

## 4.7 Optimizations and Ablations

### 4.7.1 Ablation Studies

We present several ablation studies. First, we show the impact of different joinability configurations. We then show how MATILDA can cover 100% of AMIE's rules, even those that connect non-joinable attributes.

**Joinability.** We investigate three joinability configurations:

[5] https://github.com/Fran-cois/MATILDA

**FK Joinability**: Serving as the default in our main experiments, joinability is determined exclusively by foreign key relationships. This method leverages schema-defined constraints to ensure that only semantically meaningful joins are evaluated, thereby reducing the search space.
**Rule-Based Joinability**: Joinability is established using pruning rules focused on attribute overlap and data type compatibility. Two attributes are joinable if:

1. *Overlap*: The values of the attributes exhibit a Jaccard similarity of at least **50%**, or they overlap at least **70%** in both directions.

2. *Type Compatibility*: The attributes are declared to be of the same domain (type). Numerical values are not considered joinable, to avoid accidental overlapping, such as between an ID and a person's age.

**Full Joinability**: Every attribute is considered joinable with all other attributes. Full joinability removes all of MATILDA's pruning efforts and is usually not desirable as it generates or tests rules such as $Lineage(parent = x_0) \Rightarrow Residence(state = x_0)$. We present it here just as an ablation.

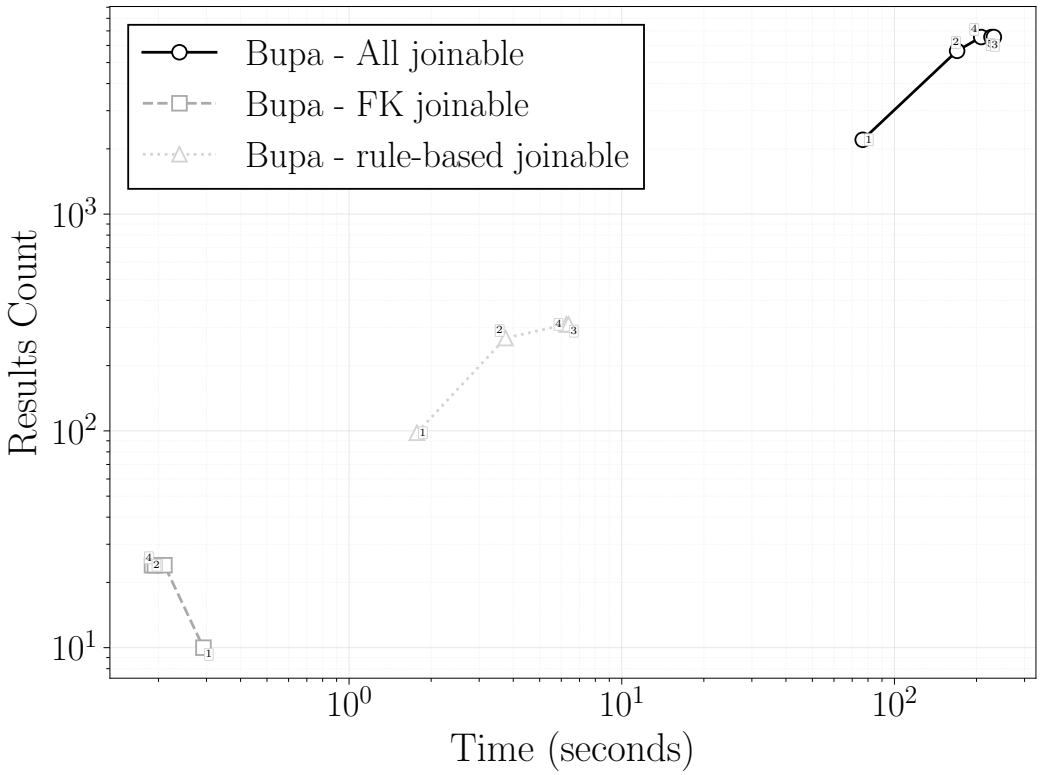


**Figure 4.13:** MATILDA's performance under different joinability settings, with recursion depth varied from 1 to 3 (log-log scale).

For our experiments, we chose the Bupa database, as it was the only database that had more than one table and could be mined with full joinability in less than a day. We evaluate each joinability setting recursion depths of 1, 2, and 3. Figure 4.13 shows the results. The FK-only joinability setting remains relatively unaffected by recursion depth, with negligible time variations (approximately 0.01 seconds) between the lower and higher levels of recursion. Conversely, both the fully joinable and the rule-based setting exhibit a notable increase in computation time and result set size as

| | AMIE 3 | | MATILDA | | |
|---|---|---|---|---|---|
| **Database** | **Results** | **Time** | **Results** | **Time** | **Cover.** |
| CSS | 2 | 53.09s | 202 | 38m52s | 100% |
| CraftBeer | 1 | 2.21s | 62 | 1.56s | 100% |
| Dallas | 3 | 1.34s | 96 | 4.83s | 100% |
| SFScores | 8 | 12.18s | 146 | 4m28s | 100% |
| WebkP | 2 | 3.66s | 26 | 9.3s | 100% |

**Table 4.5:** Comparison of AMIE 3 with MATILDA using the parameters `joinability=True`, `recursion=2`, `max_table=2`. **Coverage** indicates the percentage of AMIE's rules that are covered by MATILDA.

recursion depth increases. This observation confirms our choice of constraining joinability to foreign keys. The relationship between schema complexity and rule discovery is further illustrated in Figures 4.16 and 4.17.

#### Coverage of AMIE Rules

The rules that AMIE mines on our databases usually carry only limited insight, because they often concern a single row in a table and join attributes that are not semantically joinable, as discussed in Section 4.6.2. However, if the user wishes to find these rules also with MATILDA, this is possible: MATILDA has to be run with `joinable=ALL` (which makes all columns joinable), `recursion=2` (which allows the same table to appear at most twice in a TGD), `max_table=2` (which mines at most two tables per TGD), and `max_var=1` (which allows at most one variable per TGD). We identified 5 databases where both AMIE and MATILDA (with this configuration) mine rules, and show the results in Table 4.5. MATILDA is up to 40 times slower than AMIE. This is because, in the absence of joinability constraints, the constraint graph is exponentially large, and, unlike AMIE, MATILDA has to explore it in full in order to find full-fledged TGDs. In return, MATILDA mines not just the (often spurious) rules of AMIE, but up to 100 times more patterns. For normal applications, we recommend the default setting of MATILDA, which finds only those TGDs that join joinable attributes.
More ablation studies, which study the impact of sampling rows and columns, are in Appendix 4.7.2.

### 4.7.2 Sampling Rows and Columns

**Sampling rows.** The complexity of MATILDA depends mainly on the number of attributes, and less on the number of rows. To demonstrate this experimentally, we down-sampled the rows of the Bupa dataset, and measured MATILDA's runtime at different sampling rates. Figure 4.14 shows that a sampling results in a linear decline in the number of results. However, this effect almost vanishes in comparison to the effect of the recursion depths: The log-log scale clearly shows that higher recursion depths (marked as increasing labels along the curves) contribute significantly to the results count, irrespective of the sampling rates. At lower sampling rates (e.g., 10%), the time taken to compute results remains shorter, but fewer rules are discovered. Conversely, higher

| Database | Rule | Interpretation |
|---|---|---|
| Dunur | $\forall x_0 : \text{husband}(\text{name1} = x_0) \Rightarrow \exists z_0 : \text{brother}(\text{name2} = z_0) \wedge \text{uncle}(\text{name2} = z_0, \text{name1} = x_0)$ | Every husband $x_0$ has a brother $z_0$ who is also $x_0$'s uncle, reflecting a complex familial relationship. |
| Mutagenesis | $\forall x_0 : \text{molecule}(\text{molecule\_id} = x_0) \Rightarrow \exists z_0 : \text{atom}(\text{atom\_id} = z_0, \text{molecule\_id} = x_0) \wedge \text{bond}(\text{atom2\_id} = z_0)$ | Every molecule contains at least one atom that participates in a bond, a meaningful chemical structure constraint. |
| Student Loan | $\forall x_0 : \text{disabled}(\text{name} = x_0) \Rightarrow \exists z_0 : \text{bool}(\text{name} = z_0) \wedge \text{no\_payment\_due}(\text{bool} = z_0, \text{name} = x_0)$ | For every disabled individual $x_0$, there exists a boolean indicator ensuring that no payment is due. |
| UW | $\forall x_0 : \text{course}(\text{course\_id} = x_0) \Rightarrow \exists z_0 : \text{advisedBy}(\text{p\_id\_dummy} = z_0) \wedge \text{taughtBy}(\text{p\_id} = z_0, \text{course\_id} = x_0)$ | Every course $x_0$ has at least one person $z_0$ who both advises and teaches it, indicating a structural pattern in course management. |
| Same-gen | $\forall x_0 : \text{same\_gen}(\text{name2} = x_0) \Rightarrow \exists z_0 : \text{parent}(\text{name1} = x_0, \text{name2} = z_0) \wedge \text{person}(\text{name} = z_0)$ | Entities marked as same generation have a parent-child relationship, revealing generational or hierarchical structures. |
| Biodegradability | $\forall x_0 : \text{atom}(\text{molecule\_id} = x_0) \Rightarrow \text{molecule}(\text{molecule\_id} = x_0)$ | Every atom is linked to a molecule. |
| Country | $\forall x_0 : \text{Country}(\text{Code} = x_0) \Rightarrow \text{City}(\text{CountryCode} = x_0)$ | Every Country has a related city |

**Table 4.6:** Examples of TGDs that only MATILDA mines.

sampling rates (e.g., 90%) lead to increased computational overhead due to the larger search space, but they allow for a more exhaustive rule discovery process.
This relationship highlights the tradeoff between computational efficiency and the comprehensiveness of rule mining when adjusting sampling parameters and recursion depths. The figure also suggests that the impact of recursion depth becomes more pronounced as the row sampling rate increases, showing the exponential nature of the search space expansion in rule mining tasks.

**Sampling columns.** Figure 4.15 illustrates the effects of reducing the number of columns on the mining process. Unlike row sampling, reducing the number of available columns significantly impacts performance, both in terms of computational efficiency and the number of extracted rules. Specifically, while row reductions can decrease the number of discovered rules by a factor of approximately $10^2$, column reductions can lower results by an order of magnitude of $10^3$. This suggests that column selection directly influences rule discovery, as reducing the feature space limits the number of possible dependencies that can be inferred.
Furthermore, as recursion depth increases, the impact of column pruning becomes even more evident. A reduced attribute space leads to a significant decline in the number of generated rules. However, due to the randomness in column trimming, certain edge effects may occur, leading to cases where different levels of pruning (e.g., 10% vs. 40%) yield similar results. This irregularity arises from the specific columns removed in each iteration, as some attributes may be more pivotal in rule formation than others.

### 4.7.3 Analysis of Joinable Attributes and Runtime Performance

The number of joinable attributes in a database significantly influences the search space and runtime of pattern mining algorithms. Figure 4.13 shows the relationship between the number of joinable attribute pairs and MATILDA's execution time. We observe that MATILDA scales sublinearly with the number of joinable constraints, which validates the efficiency of our constraint-based search strategy.
Databases with more joinable attributes generally require longer execution times, but the relationship is not strictly linear due to: (1) the pruning effectiveness of our constraint propagation, (2) the distribution of data values affecting pattern frequency, and (3) the complexity of the schema structure beyond simple attribute counts.

### 4.7.4 Complex TGDs

When the joinability is set to Foreign Key Joinability, MATILDA finds only few complex TGDs. When we relax the joinability to Rule-based (Section 4.7.1), more TGDs are unearthed, and we show some examples in Table 4.6.

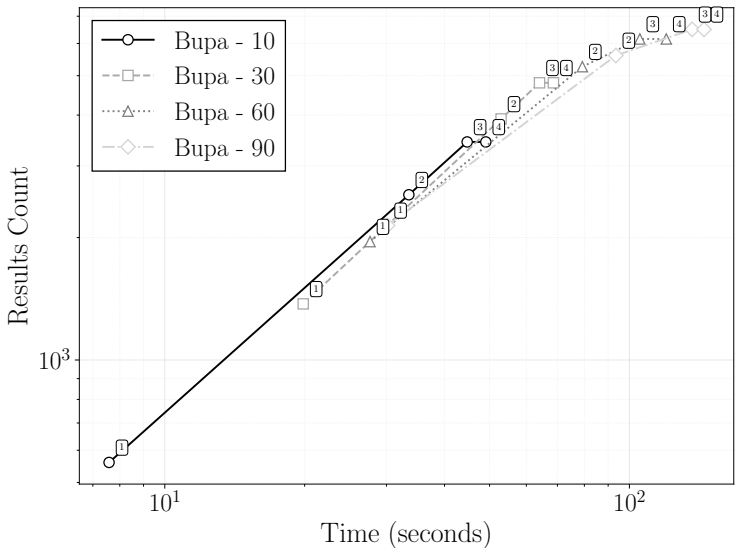


**Figure 4.14:** Performance with varying recursion depth (1,2,3,4) and row sampling rates (log-log-scale).

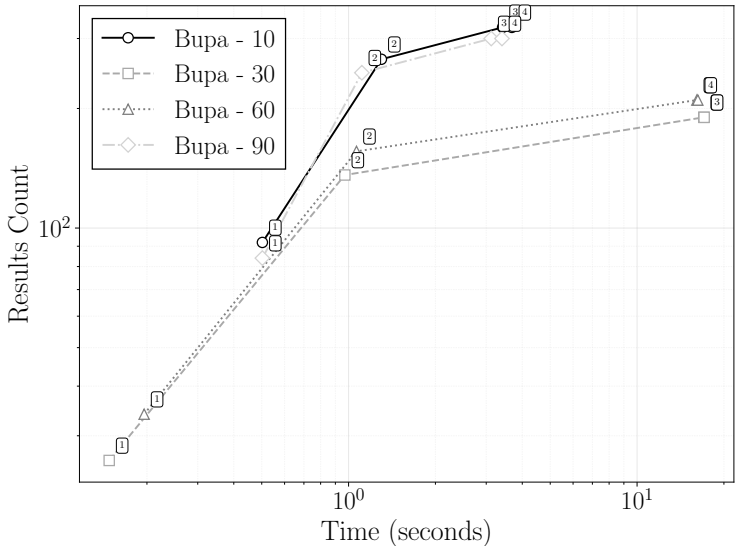


**Figure 4.15:** Performance with varying recursion depth (1,2,3,4) and column sampling rates (log-log-scale).

### 4.7.5 Ablation Study: Impact of Database Complexity on MATILDA Performance

We present a comparative ablation analysis of MATILDA's performance across two critical dimensions: **Execution time** and **rule discovery count**. This study reveals distinct patterns and correlations for each performance aspect, providing insights into the algorithm's computational behavior and effectiveness.

**Methodology**. We evaluated MATILDA with two complementary regressions: a *time analysis* linking execution time to schema and data properties, and a *rule-count analysis* relating the number of discovered rules to the same properties. Five database metrics were considered throughout, table count, column count, row count, triple count, and the foreign-key–to-column ratio.

**Key findings**. Execution time rises chiefly with data volume, showing its strongest correlations with rows ($R = 0.50$) and triples ($R = 0.47$), a modest correlation with columns ($R = 0.33$), and a negligible link to tables. Rule discovery, by contrast, is driven by schema complexity: tables correlate most strongly ($R = 0.56$) and columns moderately ($R = 0.34$), whereas dense foreign-key usage ($R = -0.33$) and larger data volumes ($R \approx -0.25$ to $-0.30$) suppress discovery. These opposing patterns reveal that MATILDA's implicit-relationship heuristics thrive on rich schemas but are impeded by sheer data size.

**Practical implications**. For time-critical scenarios, practitioners should limit rows ( and therefore triple counts); for maximal rule extraction, they should favour table-rich, row-lean designs, accepting some runtime overhead. Balanced deployments pair moderate schema complexity with controlled data volumes. Developers are encouraged to optimise row-level processing and reinforce large-scale pattern recognition, while researchers should model runtime and rule yield separately.

**Conclusion**. MATILDA exhibits a dual performance nature: execution time is governed by data volume, whereas rule discovery hinges on schema structure, necessitating distinct optimisation strategies for each objective.

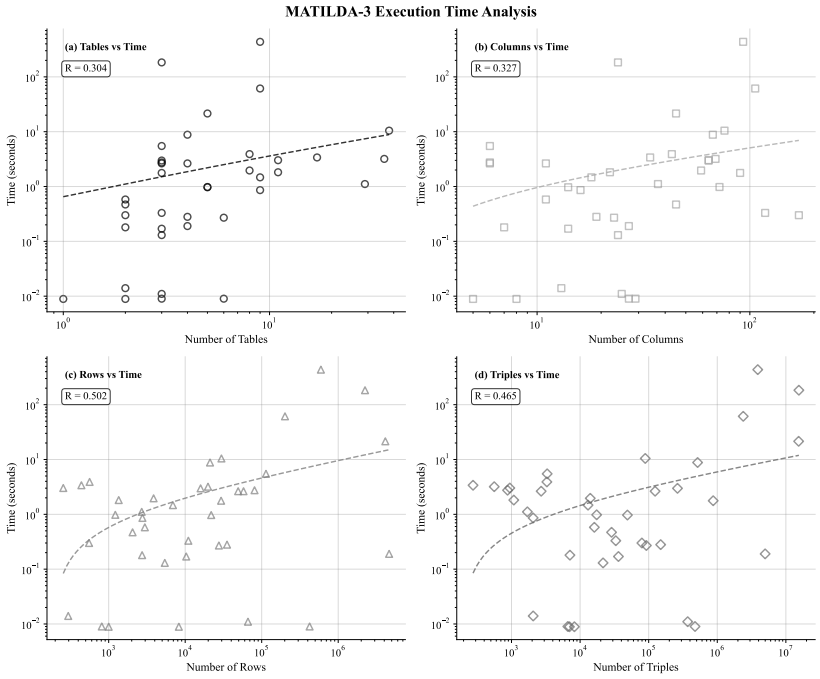


**Figure 4.16:** Correlation between number of columns and number of discovered rules.

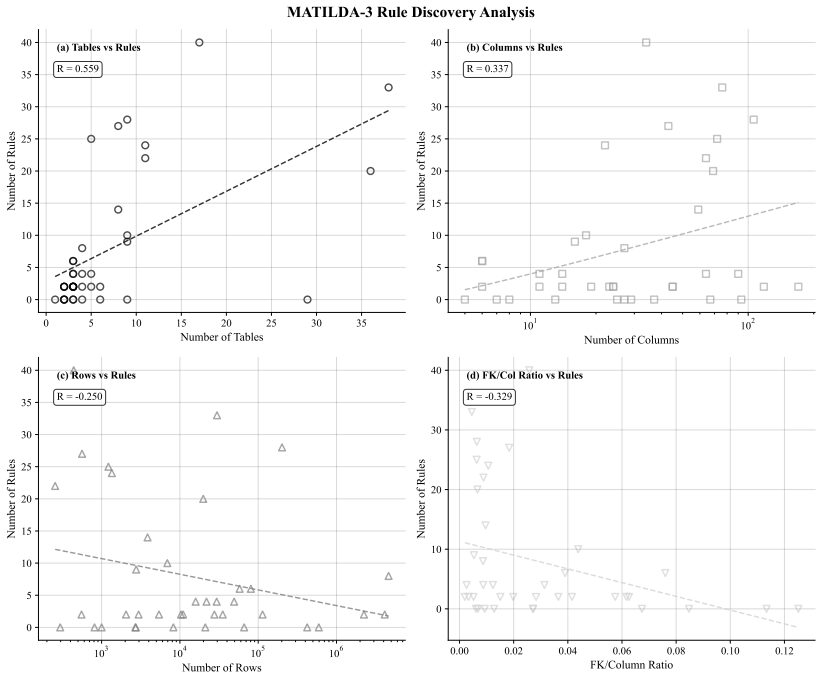


**Figure 4.17:** Correlation between number of tables and number of discovered rules.

## 4.8 Discovery Algorithm Comparison: DFS, BFS, and A*

We retain three planners that cover complementary trade-offs for MATILDA: *DFS* (baseline), *BFS* (level-wise exploration), and *A** (best-first with a global priority queue). DFS has minimal control overhead and a small memory footprint; it turns early pruning signals into saved work but explores narrowly without global guidance. BFS expands the search space level by level, which can surface short, high-quality rules early and lends itself to set-oriented or batched evaluation; the cost is a larger frontier and higher per-iteration overhead on wide branching factors. A* relies on a priority function $f=g+h$; when $h$ aligns with the objective and memory is ample, it can reach strong rules quickly at the cost of maintaining a heuristic-ordered frontier.

**Empirical observation on BUPA.** On *BUPA*, the three traces are nearly superposed (Figure 4.18). A* and BFS surface the first rules slightly earlier (about 0.024–0.025 s) than DFS (about 0.047 s), and their cumulative-query curves advance with comparable slopes; by roughly 0.32 s all planners reach ∼40–45 rules and then progress in step (Figure 4.19). End-of-run totals are close in our run (DFS/BFS ≈ 753 queries, A* ≈ 696). The plots thus mainly show early traversal-order effects; beyond the first instants, the three planners proceed at a matching pace on this dataset.

On the *BUPA* benchmark, DFS, BFS, and A* exhibit near-identical runtimes, query counts, and time-to-first rules; observed gaps fall within normal run-to-run variance and are not statistically meaningful. In this setting, the planner mainly affects exploration order and transient frontier size rather than total cost or result quality. We therefore use *DFS* as the default for its simplicity and footprint. Orthogonally, another way to improve performance is to change the database driver (next section). Larger schemas or different pruning regimes may accentuate differences, but on *BUPA* the choice of planner is an engineering preference rather than a performance lever.

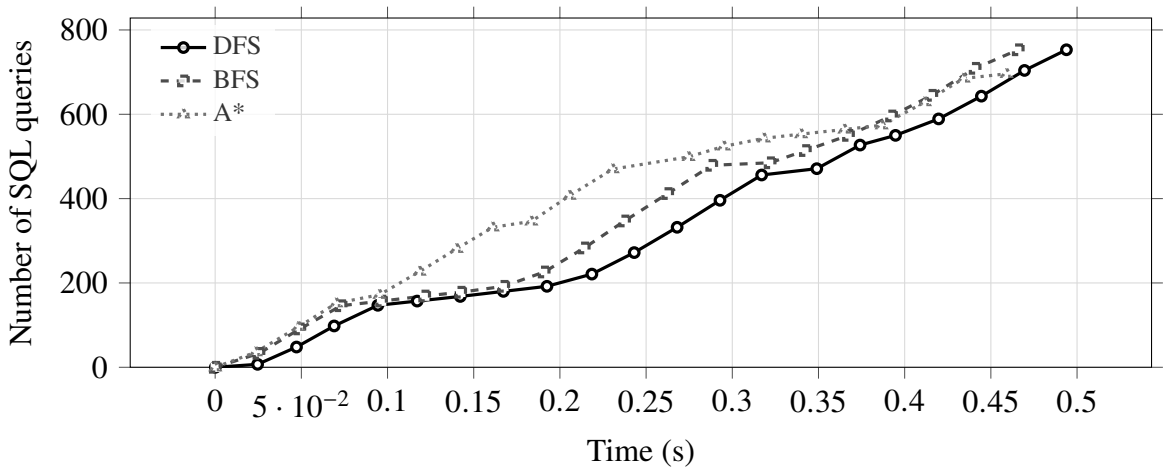


**Figure 4.18:** Runtime vs. number of SQL queries on the *BUPA* database. Curves nearly coincide; differences are not statistically significant under our settings.

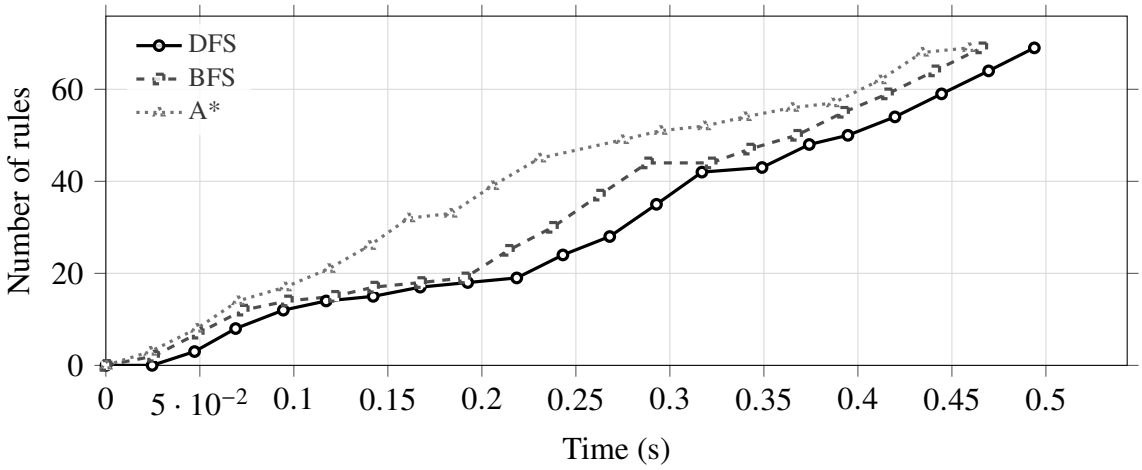


**Figure 4.19:** Cumulative number of discovered rules over time on the *BUPA* database. All planners reach the final set at a similar pace; small gaps fall within run-to-run variance.

## 4.9 Indexes and Join Performance Across Database Engines

We cap recursion depth (that is, the number of repeated self-joins of the same relation) at three because this already surfaces the real scalability limits of mainstream engines on realistic, string-heavy schemas, while deeper recursion offers diminishing returns relative to the engineering it demands. In engines dominated by nested loops such as SQLite, a three-way self-join on non-key string attributes is structurally expensive: any missing inner index devolves into repeated full scans, and even when secondary indexes exist, low selectivity, skew, or collation and expression predicates force wide range probes. In practice, the *self-join*×3 pattern can stretch from minutes to hours on multi-million-row inputs, whereas ×1 or ×2 remain tractable.

By contrast, Oracle's cost-based optimizer can pivot to hash or sort-merge joins, exploit parallelism, and use richer index families, notably bitmap and function-based indexes, to tame the same triple self-join. With the right indexes and fresh statistics in place, wall-clock times often drop from hours to minutes, sometimes to near-instant. The price is operational: sustained performance depends on a DBA, i.e., the database administrator responsible for indexing strategy, statistics management (including histograms and extended column groups), partitioning and parallelism settings, and plan stability across releases.

To gauge portability, we built a converter, `sqlite3-to-oracle`[6], that automates schema and data transfer and attempts to carry over join-key indexes where feasible. Import itself is difficult because of type mapping, collation and timezone issues, and large-object handling. Automatic indexing can help once the data are in place, but it is also non-trivial to adopt and supervise; in practice it is more manageable than hand-crafting every index, yet it still requires monitoring to validate candidate indexes, control storage growth, and ensure plan stability. The converter removes much of the mechanical toil, but it cannot erase the need for DBA work: partial-to-function-index mismatches and extended statistics still require attention before Oracle reliably selects hash or

[6] https://github.com/Fran-cois/sqlite3-to-oracle

merge or bitmap-assisted plans. End-to-end automation succeeded on roughly two-thirds of our tables; the remainder needed manual Data Definition Language (DDL) or statistics fixes.
MATILDA is deliberately lightweight and repeatable, running out of the box on plain SQLite (no vendor drivers, no Docker[7]. setup, minimal dependencies), which keeps experiments portable across machines and makes depth 3 a consistent stress point rather than a tuning exercise. We therefore keep SQLite as the primary baseline for portability and reproducibility and treat Oracle as a confirmatory environment: it demonstrates that advanced indexing, for example bitmap, can compress the triple self-join dramatically, but pushing beyond depth 3 adds little scientific insight relative to the extra engineering it entails.

## 4.10 Chapter summary

We presented MATILDA, a scalable, duplicate-free miner of Tuple-Generating Dependencies (TGDs) from tabular data. The method builds a redundancy-free *Constraint Graph* of schema-valid equalities; canonical paths define *proto-rules*, and a systematic frontier/existential split yields TGDs. A depth-first traversal with schema-guided joinability (e.g., foreign keys) and anti-monotone pruning on data counts keeps exploration tractable, with evidence computed under closed-world semantics and explicit `NULL` handling. The framework guarantees completeness for positive-count, bounded-length proto-rules and uniqueness up to $\alpha$-renaming. In practice, moderate support/confidence thresholds and a short maximum path length suffice; open directions include non–foreign-key joinability heuristics, richer treatment of incomplete data, tighter DBMS integration for very large joins, and the recursive extension outlined in Section 4.5.

[7]Docker is an open platform for developing, shipping, and running applications. For more information, visit the official website: https://www.docker.com/

# CHAPTER 5

# Mining Horn Rules with Disjoint Semantics

This chapter introduces MAHILDA, a high-performance system purpose-built for mining Horn rules from relational databases (**RQ5**). Although MATILDA can also mine Horn rules, focusing on this fragment unlocks substantial simplifications and optimizations. Horn rules constitute a practically important subset of FO-TGDs, on our corpora, at least 30% of the rules mined by MATILDA are Horn, and, thanks to their single-head structure and the absence of existential quantifiers, they can typically be mined more efficiently. This chapter therefore complements Chapter 4 by concentrating exclusively on the Horn fragment and assessing when it suffices versus when the additional expressiveness of FO-TGDs is warranted.
MAHILDA exploits the Horn restriction to streamline MATILDA's core search. The single-head form eliminates the multi-head/existential "splitting" step, shrinking the hypothesis space and enabling more aggressive pruning. In particular, MAHILDA leverages the standard anti-monotonicity of support under body expansion together with tighter confidence upper bounds to abandon unpromising branches early. Finally, we enforce *disjoint semantics* as an evaluation-time guard to rule out vacuous self-matches, which is especially important for recursive patterns.

## 5.1 Background and Problem

For this work, we adopt the formal setting established in prior sections. Specifically, we use the closed-world assumption and set semantics with explicit `NULL`s (Section 2.2). We model relationships via the constraint-graph abstraction (Section 4.3.3), restricting joins to be guided by foreign keys. The primary new element is the disjoint-semantics guard for Horn rules (Section 5.2). This approach to handling recursion mirrors that for general TGDs (Section 4.5): both aim to prevent trivial self-matches when a relation appears multiple times, but employ mechanisms tailored to their rule classes. For general TGDs, the mechanism is *structural*; during graph construction, we introduce *qualified attributes* (e.g., $t_i.a$) and build an N-RECURSIVE CONSTRAINT GRAPH that distinguishes table occurrences from the outset. For the single-head Horn fragment, the mechanism is *semantic*; during evaluation, we enforce *disjoint semantics*, i.e., different occurrences of the same table must bind to tuples with distinct primary keys.
Finally, we assume all rules are bounded by a total length L, representing the total number of atoms in the body and head.

## 5.2 Disjoint Semantics

While recursion dramatically enlarges the search space, discovering meaningful rules requires excluding self-evident results. For instance, consider a rule intended to find a person's children by leveraging lineage and marriage information:

$$\texttt{LINEAGE}(p, c_1) \ \wedge \ \texttt{MARRIAGE}(p, s) \ \Rightarrow \ \texttt{LINEAGE}(p, c_2)$$

A naive evaluation of this rule is flawed because it would always discover the trivial case where $c_2 = c_1$. This conveys no new knowledge. We call such self-evident results *vacuous rules*. To prune these rules, we impose *disjoint semantics* across the rule's body and head. Whenever two occurrences of the same relation appear in the rule and agree on the bindings of their shared variables, they must be witnessed by different tuples.
This preserves shared-variable bindings while excluding vacuous instantiations.
We denote this with an index such constraints, for example:

$$\texttt{LINEAGE}_0(p, c_1) \ \wedge \ \texttt{MARRIAGE}(p, s) \ \Rightarrow \ \texttt{LINEAGE}_1(p, c_2)$$

This revised rule now correctly searches for a child $c_2$ who is different from $c_1$, effectively identifying siblings.

**Definition 5.2.1** (Disjoint Semantics)**.** *Let*

$$\psi(\vec{x}) = \bigwedge_{t \in T} \bigwedge_{i=1}^{n_t} t_i(\vec{x})$$

*be a conjunction of atoms, where $T$ is a set of tables in the TGD and for each table $t$ the index $i$ runs over the $n_t$ occurrences of $t$ in the TGD. Assume that each table $t$ has an associated key, denoted by $Key(t)$. The* disjoint semantics *of $\psi$ is defined as:*

$$DS(\psi(\vec{x})) \equiv \bigwedge_{t \in T} \ \bigwedge_{1 \le i < j \le n_t} \ \bigvee_{a \in Key(t)} t_i.a(\vec{x}) \neq t_j.a(\vec{x}).$$

In other words, for every table $t$ that appears multiple times in the Horn rule, for each pair of distinct occurrences $t_i$ and $t_j$ (with $i \neq j$), there must be at least one key attribute $a$ for which the value assigned in occurrence $i$ differs from that in occurrence $j$. This guarantees that the different occurrences point to different rows in the database.

**Example 5.2.1.** *The Horn rule*

$$Lineage_0(Parent = x) \Rightarrow Lineage_1(Parent = x)$$

*under standard semantics would allow a parent to satisfy the rule using the same tuple in both occurrences. Under disjoint semantics, the prediction is refined to*

$$Lineage_0(Parent = x, Child = c) \wedge Lineage_1(Parent = x, Child = c') \wedge c \neq c',$$

*so that the two occurrences of Lineage must refer to different rows (as differentiated by the* `Child` *attribute, which is assumed to be part of the key).*

We now adapt the metrics accordingly.

**Definition 5.2.2** (Disjoint Horn rule Prediction)**.** *Given a Horn rule of the form*

$$\phi(\vec{x}, \vec{y}) \Rightarrow \psi(\vec{x}),$$

*its* disjoint prediction *is defined as the set of tuples:*

$$\{\vec{x} \mid \exists \vec{y}, \vec{z} : \ \phi(\vec{x}, \vec{y}) \wedge \psi(\vec{x}) \wedge DS(\phi(\vec{x}, \vec{y}) \wedge \psi(\vec{x})\}.$$

By replacing the standard prediction with the disjoint prediction in the definitions of support and confidence, we obtain the disjoint variants of these metrics.
Finally, we adjust the proto-rule count to account for the disjoint semantics.

**Definition 5.2.3** (Proto-Rule Count with Disjoint Semantics)**.** *Let $PR$ be a proto-rule involving qualified atoms from one or more tables. The* proto-rule count with disjoint semantics*, denoted by $count_D(PR)$, is defined as the number of bindings for the variables in $PR$ that satisfy both the proto-rule and the disjoint constraints; that is, the number of assignments such that:*

$$PR(\vec{x}) \wedge DS(PR(\vec{x}))$$

*holds in the database. In effect, only bindings where, for every table t appearing multiple times in PR, the corresponding key attributes in different occurrences are assigned to distinct values are counted.*

**Example 5.2.2** (Children via marriage: self vs. disjoint semantics)**.** *Consider the rule*

$$\mathit{LINEAGE}(p, c_1) \ \wedge \ \mathit{MARRIAGE}(p, s) \ \Rightarrow \ \mathit{LINEAGE}(p, c_2).$$

***Naive (self) evaluation.*** *Any body match witnesses the head with the* same *child ($c_2 = c_1$), so the rule is trivially satisfied. On the instance of Fig. 1.1, there are three body matches (for $p \in \{Priscilla, Elvis, Lisa\}$), hence support = 3 and confidence = 1.*
***Disjoint semantics evaluation.*** *By Def. 5.2.1, the two occurrences of* `LINEAGE` *must bind* distinct *tuples. With primary key (Parent, Child), this enforces $(p, c_1) \neq (p, c_2)$, i.e. $c_2 \neq c_1$. Thus only matches that exhibit a* different *child for the same parent are counted. In Fig. 1.1 no parent has two distinct children, so support = 0 and confidence = 0.*
Note. *disjoint semantics does not alter the syntactic form of the rule; it acts solely as a runtime constraint during evaluation. For clarity, one may annotate it as*

$$\mathit{LINEAGE}(p, c_1) \wedge \mathit{MARRIAGE}(p, s) \Rightarrow \mathit{LINEAGE}(p, c_2) \ \wedge \ (c_2 \neq c_1),$$

*which serves merely as notational shorthand for the same semantics.*

**Why we use primary keys for disjoint semantics.** Let $R(A_1, \ldots, A_k)$ be a relation (table) with primary key attributes $\mathrm{pk}(R) \subseteq \{A_1, \ldots, A_k\}$. Write $\pi_S(\bar{t})$ for the projection of a tuple of terms $\bar{t}$ onto an attribute set $S$, and define the *key projection* $\kappa_R(\bar{t}) \coloneqq \pi_{\mathrm{pk}(R)}(\bar{t})$. Under set semantics and a declared primary key, $\kappa_R$ is injective: two atoms $R(\bar{t})$ and $R(\bar{t}')$ denote the same database row iff $\kappa_R(\bar{t}) = \kappa_R(\bar{t}')$. Hence, to enforce that two occurrences of $R$ refer to *different* rows, it suffices (and is correct) to require $\kappa_R(\bar{t}) \neq \kappa_R(\bar{t}')$.
When two occurrences of the same relation $R$ (in the body or in the head) must be pairwise disjoint while preserving the bindings of a set of shared attributes $B$, we require that they agree on $B$ but differ on the primary key: $\exists \bar{t}_i, \bar{t}_k \left( R(\bar{t}_i) \ \wedge \ R(\bar{t}_k) \ \wedge \ \pi_B(\bar{t}_i) = \pi_B(\bar{t}_k) \ \wedge \ \pi_{\mathrm{pk}(R)}(\bar{t}_i) \neq \pi_{\mathrm{pk}(R)}(\bar{t}_k) \right)$.
Intuitively: "find another row of $R$ that agrees on the shared attributes $B$ but has a different primary key." The key-based inequality packs the condition into a single, index-friendly predicate and avoids a wide disjunction over attributes.
*Other instantiations.* For **MARRIAGE**(*partner1*, *partner2*) with key *partner1*, disjointness for the same *partner1* reduces to a single inequality on *partner2*:

$$\exists\, p_2' \ \big(\mathtt{MARRIAGE}(p_1, p_2') \wedge p_2' \neq p_2\big).$$

For **LINEAGE**(*Parent*,*Child*) with composite key (*Parent*, *Child*), disjointness for a fixed parent $p$ reduces to finding $c' \neq c$:

$$\exists\, c' \ \big(\mathtt{LINEAGE}(p, c') \wedge c' \neq c\big).$$

In SQL, if row-value (tuple) comparisons are supported, disjointness reduces to the single predicate `(r2.Parent, r2.Child) <> (r1.Parent, r1.Child)`; otherwise use an equivalent key, `r2.Parent IS DISTINCT FROM r1.Parent OR r2.Child IS DISTINCT FROM r1.Child`.

**Example 5.2.3** (SQL realization on the running rule)**.**

$$\mathit{LINEAGE}(p, c_1) \ \wedge \ \mathit{MARRIAGE}(p, s) \ \Rightarrow \ \mathit{LINEAGE}(p, c_2).$$

*Under disjoint semantics, the head* `LINEAGE` *must witness a tuple different from the body's* `LINEAGE` *for the* same *parent* $p$*. If* $\mathrm{pk}(\mathit{LINEAGE}) = (\mathit{Child})$*, this becomes* $c_2 \neq c_1$*.*
*SQL (key-based).*

```
SELECT DISTINCT r1.parent AS p, r2.child AS c2
FROM LINEAGE AS r1
JOIN MARRIAGE AS m ON m.partner1 = r1.parent
JOIN LINEAGE AS r2
  ON r2.parent = r1.parent                -- same shared vars {Parent}
 AND ( r2.child) <> ( r1.child)  -- disjoint by key
```

*Because* `r2.parent = r1.parent`*, the inequality reduces to* `r2.child <> r1.child`*, which the optimizer can check via indexes on the key.*
*SQL (no declared key: fallback on tuple disequality).*

```
... JOIN LINEAGE AS r2
  ON r2.parent = r1.parent
 AND (r2.parent IS DISTINCT FROM r1.parent
      OR r2.child  IS DISTINCT FROM r1.child
/* or, in general, any attribute
  in Attr(LINEAGE) to ensure a
  different row under bag tables */)
```

*Without a key, ensuring "different row" requires an attribute-wise disjunction (often with NULL-safe comparisons like* `IS DISTINCT FROM`*), which is wider and harder to optimize than a single key inequality.*

## 5.3 MAHILDA

MAHILDA can be viewed as a Horn-specialized instance of MATILDA. It reuses the same constraint-graph and schema-guided joinability regime, but restricts the rule language to *Horn rules* (universally quantified, no existential witnesses). Search states are connected *proto-rules* that fix the body side of a candidate rule, and the head consists of exactly one relational atom. This specialization yields a leaner search, centered on two targeted modifications:
**Single-Atom Head Constraint.** Evaluate only rules with a single relational atom in the head. This removes the need for complex "split" procedures designed for multi-atom heads, since each constructed proto-rule is tested against just one potential head.
**Anti-monotonic pruning (fixed head).** With the head fixed, any refinement of the body (adding literals) can only shrink the match set and cannot enlarge the subset compatible with the head. Consequently, support is anti-monotone and the optimistic (prediction) upper bound on confidence

is monotone non-increasing under body extension. Therefore, extend a *proto-rule* only while some quality metric can still improve; if the measured confidence reaches its bound, or the bound falls below the thresholds, stop recursing under this *proto-rule*.

### 5.3.1 Algorithm Walkthrough

**Main algorithm.** Algorithm 5.1 (MAHILDA) takes as input a database instance $D$, thresholds $(\kappa_s, \kappa_c)$ for minimum support and confidence, and a length bound $\ell_{\max}$. Its output is a set of mined Horn rules, each represented as a record $(proto\_rule, head, support, confidence, signature)$. The search proceeds depth–first over *proto-rules*, conjunctions of atoms with equality constraints, which serve as structured partial hypotheses. As formalized in Definition 4.3.4, a single proto-rule determines a family of derived Horn rules by pairing the body it encodes with each admissible relation symbol as a candidate head under the schema's joinability regime.

**Control flow.** MAHILDA begins by enumerating root proto-rules and, for each, calls a recursive helper DFS. At any node, DFS first checks global stopping conditions: if the current proto-rule exceeds the bound $\ell_{\max}$ or the prediction oracle reports PRED = 0 (no remaining evidence worth exploring), recursion halts immediately. Otherwise, the routine evaluates all admissible heads at this node via EVAL$(D, proto_rule, h)$, obtaining support and confidence. Candidates meeting the thresholds $(\kappa_s, \kappa_c)$ and deemed canonical by MINIMAL are mapped to a canonical *signature* SIGN$(proto_rule, h)$, which identifies equivalence classes of rules modulo isomorphisms.

**Cache and pruning.** A branch-local cache tracks, for each signature, the best confidence seen so far. A candidate is accepted only if it strictly improves the cache entry for its signature; upon acceptance, the record is emitted and the cache is updated. This cache also drives pruning: if no head at the current node improves any cached signature, the branch is cut, no extension can redeem a head already dominated at its signature.

**Expansion.** When at least one head improves the cache, DFS extends the proto-rule along each admissible edge returned by EDGES$(proto_rule)$, forming $next$ = EXTEND$(proto_rule, e)$, and recurses on *next* with the *same* cache. Disjoint semantics is enforced by the oracle in EDGES (and consistently reflected in EVAL), ensuring that repeated occurrences of the same relation are witnessed by distinct tuples and preventing vacuous self-joins. The recursion aggregates all accepted records across the explored subtree, and the outer MAHILDA loop unions these results over all roots.

**Outcome.** The result is a concise, non-redundant set of Horn rules that (i) satisfy $(\kappa_s, \kappa_c)$, (ii) are canonical under the chosen signature, and (iii) have survived cache-guided branch-and-bound pruning while respecting the disjoint semantics guard and the length limit $\ell_{\max}$.

```
1
2  def CacheBest(sig, cache):
3      """Return best confidence seen for signature (0.0 if unseen)."""
4      return cache.get(sig, 0.0)
5
6  def MAHILDA(DB, ks, kc, Lmax):
7      """
8      Mine Horn rules with DFS + prediction-guided pruning.
9      Returns: list of (proto_rule, head, support, confidence, signature).
```

```
    """
    results = []
    for root in INIT_PROTO_RULES(DB):
        cache = {}  # signature -> best confidence
        results += _dfs(DB, root, cache, ks, kc, Lmax)
    return results

def _dfs(DB, proto_rule, cache, ks, kc, Lmax):
    out = []
    # Global stops: length bound or zero prediction mass
    if PROTO_LEN(proto_rule) > Lmax or PRED(proto_rule) == 0:
        return out
    cache_improved = False
    # Score admissible heads at this node
    for head in HEADS(proto_rule):
        supp, conf = EVAL(DB, proto_rule, head)
        if supp >= ks and conf >= kc and MINIMAL(proto_rule, head, ks, kc):
            sig = SIGN(proto_rule, head)
            if CacheBest(sig, cache) < conf:
                cache[sig] = conf
                out.append((proto_rule, head, supp, conf, sig))
                cache_improved = True
    # If no head improved the cache, stop exploring this branch
    if not cache_improved:
        return out
    # Otherwise, extend along admissible edges (disjointness enforced by EDGES)
    for e in EDGES(proto_rule):
        nxt = EXTEND(proto_rule, e)
        out += _dfs(DB, nxt, cache, ks, kc, Lmax)
    return out
```

**Listing 5.1:** MAHILDA: DFS miner with cache-guided pruning

### 5.3.2 Complexity

To determine the complexity of MAHILDA, we focus on the number of *split proto-rules* generated based on the number of joinable attributes $n$. We decompose the complexity into:

1. the number of start nodes,
2. the number of proto-rules generated from each start node,
3. and the number of possible splits for each proto-rule.

There are $n$ start nodes (one per pair of joinable attributes). In the worst case, every pair of joinable attributes is joinable with every other one, making the constraint graph a complete acyclic orientation with $\binom{n}{2}$ edges. From a single start node, the number of proto-rules (simple paths) that can be generated is $2^{n-1} - 1$: Any subset of the remaining $n - 1$ nodes, taken in increasing order, yields exactly one path. Summing over all starts does not multiply this by $n$ (each path has a unique smallest node), so the total number of distinct proto-rules across all starts is $2^n - n - 1$.

For a proto-rule $r$, let $T(r)$ be the number of tables used by $r$. In the worst case, $T(r)$ can be as large as $n$, and to have all horn rules, each proto-rule can be split in up to $T(r)$ ways; thus we take the upper bound $n$ per rule. Multiplying these factors (Figure 5.1) gives the following worst-case complexity:

$$O\big((2^n - n - 1) \times n\big) = O(n2^n).$$

Which is nicer than MATILDA's complexity.

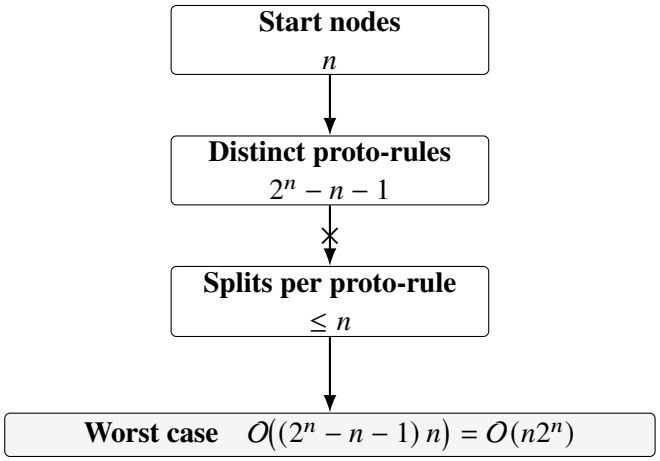


**Figure 5.1:** MAHILDA worst-case complexity, for split proto-rule enumeration.

## 5.4 Evaluation

### 5.4.1 Experimental Setup

**Environment and Datasets.**

We evaluate MAHILDA on **54** multi-table datasets from `relational-data.org`, migrated to `SQLite 3.46`. All experiments run on a server with an Intel Xeon E5-2650 v2 @ 2.60 GHz processor, with a strict two-hour time limit and a 15 GB memory cap per job. We report on a representative subset of ten databases in this paper (Table 5.1); the full scripts, logs, and results for all 54 datasets are available in our public repository.[1]

**Metrics.**

We measure wall-clock time and mean resident-set size (RSS) for memory footprint. A lightweight sampler records the RSS, which includes the Python interpreter, MAHILDA, and SQLite buffers, providing a close approximation of the total in-RAM usage.

**Mining Configuration.**

Unless stated otherwise, we use the following default parameters:

[1] `http://github.com/Fran-cois/MAHILDA`

- **Graph Construction.** The search is constrained to attributes connected by declared FOREIGN-KEY edges. This focuses the search on semantically meaningful join paths and is crucial for sub-second performance.

- **Support & Confidence.** We set the minimum support threshold $\kappa_s = 0$ and enforce no minimum confidence, enumerating all potential rules. Rules matching only a single tuple are filtered out post-discovery.

- **Maximum Rule Length.** We cap the total number of atoms in the rule body and head to $L = 3$.

**Baselines.**

We benchmark MAHILDA against three established systems to situate its performance and rule output, using the same set of miners from the MATILDA study for direct comparability:

- **POPPER:** An Inductive Logic Programming (ILP) system that searches for Horn rules.

- **AMIE 3:** A prominent Horn-rule miner designed for knowledge bases, which requires converting the relational schema to RDF triples.

- **SPIDER:** A state-of-the-art discovery tool for unary inclusion dependencies (INDs).

**Table 5.1:** A representative sample of the 54 benchmark datasets used in our evaluation.

| Database | #Tables | #Rows | Type |
|---|---|---|---|
| Biodegradability | 5 | 21,875 | Synthetic |
| CDESchools | 3 | 29,481 | Real |
| Chess | 2 | 2,052 | Synthetic |
| Cora | 3 | 57,884 | Real |
| CraftBeer | 2 | 2,968 | Real |
| Countries | 4 | 21,118 | Real |
| Dunur | 17 | 440 | Synthetic |
| Nations | 3 | 11,004 | Synthetic |
| PTE | 38 | 29,762 | Real |
| SAT | 36 | 19,867 | Synthetic |

### 5.4.2 Results and Analysis

MAHILDA consistently discovers its target class of rules several orders of magnitude faster than all baselines. As shown in Table 5.2, it completes most tasks in under a second, achieving speedups of $10^3$–$10^4\times$ over the slowest competing method. This performance gap is most pronounced on complex schemas where baselines time out (e.g., POPPER on CHESS) or run out of memory (e.g., AMIE 3 on CDESCHOOLS).

**Table 5.2:** Runtime and rule counts for MAHILDA versus baselines. Time is in seconds (lower is better). A dash (—) indicates a timeout (>2h), while OOM indicates an out-of-memory error. The fastest runtime for each dataset is in **bold**.

| | Popper | | SPIDER | | AMIE 3 | | MAHILDA | |
|---|---|---|---|---|---|---|---|---|
| Database | #Rules | Time(s) | #Rules | Time(s) | #Rules | Time(s) | #Rules | Time(s) |
| Biodegradability | 0 | 1754.1 | 4 | 1.6 | 0 | 0.8 | 4 | **0.22** |
| CDESchools | OOM | OOM | 2 | 18.5 | OOM | OOM | 3 | **1.79** |
| Chess | OOM | OOM | 1 | 1.9 | 0 | 14.8 | 2 | **0.08** |
| Cora | 0 | 20.6 | 4 | 1.7 | 0 | 2.6 | 4 | **0.43** |
| CraftBeer | 0 | 35.1 | 1 | 1.5 | 0 | 2.2 | 2 | **0.06** |
| Countries | OOM | OOM | OOM | OOM | 0 | 270.6 | 0 | **1.21** |
| Dunur | 1 | 3.9 | 20 | 1.7 | 0 | 0.6 | 12 | **0.19** |
| Nations | OOM | OOM | 2 | 5.7 | 0 | 623.2 | 2 | **0.18** |
| PTE | 6 | 2122.2 | 22 | 4.8 | 0 | 3.5 | 16 | **0.48** |
| SAT | 4 | 0.7 | 8 | 2.6 | 0 | 9.5 | 7 | **0.41** |

**Analysis of Rule Coverage: Quality over Quantity.** The differences in the number of discovered rules stem from fundamental differences in system design and objectives. MAHILDA is not designed to find every possible statistical regularity; instead, it targets a specific class of high-value rules: **non-vacuous, joinable Horn rules that are semantically grounded in the database's foreign-key structure**.
This focused approach is a deliberate trade-off. By design, MAHILDA excludes:

1. *Most unary inclusion dependencies (INDs).* MAHILDA is stricter than SPIDER and aggressively prunes them under disjoint semantics.

2. *Vacuous or redundant rules.* It prunes self-joins and other trivial patterns that inflate rule counts without providing actionable insight.

3. *Rules without foreign-key paths.* Its schema-guided search avoids the combinatorial explosion faced by systems like Popper and AMIE 3 that must consider a much larger space of potential attribute relationships.

The result is a smaller, more relevant set of rules that capture functional dependencies across multiple tables. For data practitioners, a concise set of validated, cross-table rules is often more valuable than an exhaustive list of thousands of low-level patterns that require significant manual filtering. In summary, MAHILDA achieves its remarkable speed by intelligently constraining its search space to the most semantically significant rules, demonstrating that prioritizing rule quality can yield orders-of-magnitude gains in performance.

## 5.5 Ablations

### 5.5.1 Impact of Disjoint Semantics on Scalability

This subsection evaluates the computational impact of MAHILDA's disjoint semantics (Definition 5.2.1) by comparing runtime performance and rule discovery behavior against the traditional non-disjoint baseline as recursion depth increases.

**Experimental Setup.**

We use the **Biodegradability** dataset from the RELATIONAL-DATA.ORG repository, containing 1,055 tuples across 13 tables with 47 attributes and 12 foreign-key relationships. This schema exhibits sufficient complexity to stress-test recursive rule discovery while remaining tractable for exhaustive exploration under both semantic variants.

**Configuration.**

We run MAHILDA with recursion depth $N \in \{1, 2, \ldots, 10\}$ under two modes:

- **Disjoint semantics:** Self-joins are constrained by primary key inequalities (Definition 5.2.1)
- **Non-disjoint semantics:** Traditional semantics allowing arbitrary self-joins without constraints

All other parameters remain constant: support threshold $\kappa_s = 1$, confidence threshold $\kappa_c = 0.1$, maximum walk length $L = 3$, and a timeout of 600 seconds.

**Metrics.**

We measure (i) end-to-end runtime including SQL query execution, (ii) total number of discovered rules meeting the thresholds, and (iii) memory consumption during peak search activity.

**Results and Analysis**

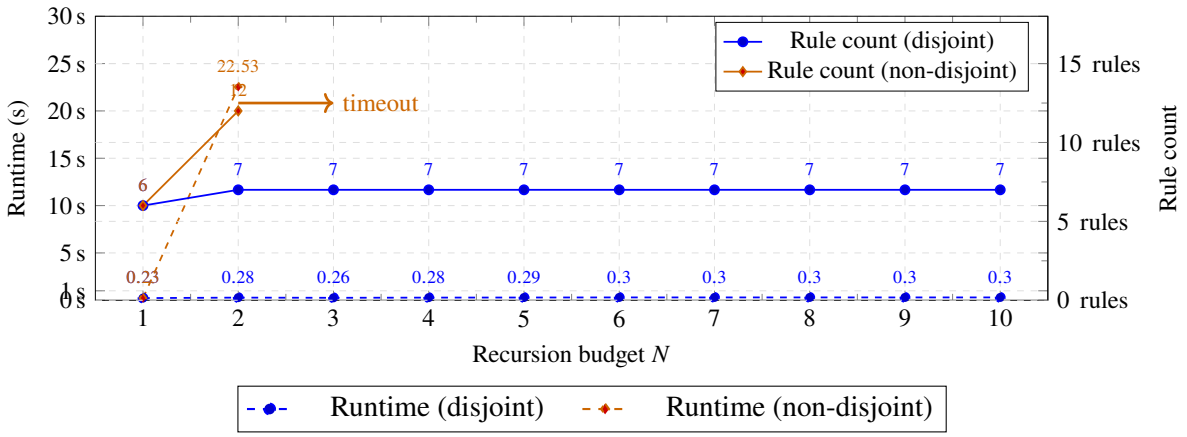


**Figure 5.2:** Runtime performance and rule discovery behavior under disjoint vs. non-disjoint semantics on the **Biodegradability** dataset. Left axis shows execution time (seconds); right axis shows number of discovered rules. disjoint semantics achieve stable sub-second performance with early convergence, while non-disjoint semantics exhibit exponential growth and timeout beyond $N = 2$.

The results (Figure 5.2) demonstrate a stark performance dichotomy between the two semantic approaches.

**Disjoint semantics: Stable performance with early convergence.**

Under disjoint semantics, MAHILDA exhibits remarkable stability across all recursion depths. Runtime increases modestly from 0.23s at $N = 1$ to 0.28s at $N = 2$, then plateaus at approximately 0.30s for $N \geq 3$. This flat performance profile reflects the aggressive pruning effect of disjoint constraints: once MAHILDA explores the meaningful cross-table dependencies, additional recursion depth yields only redundant self-join variants that are immediately rejected.
The rule count converges to 7 rules at $N = 2$ and remains constant thereafter. These rules represent genuine functional dependencies spanning multiple tables through foreign-key relationships, precisely the patterns most valuable for data quality assessment and query optimization.

**Non-disjoint semantics: Combinatorial explosion.**

In contrast, non-disjoint semantics trigger exponential growth in both runtime and rule count. Runtime jumps from 0.23s at $N = 1$ to 22.53s at $N = 2$, a 98× increase, before hitting the 600-second timeout for $N \geq 3$. The rule count similarly explodes from 6 to 12 rules between $N = 1$ and $N = 2$. This behavior stems from the proliferation of self-join variants. Without disjoint constraints, the algorithm generates multiple logically equivalent rules that differ only in their self-join patterns. For example, a single meaningful dependency like `Molecule`(logp) $\Rightarrow$ `Molecule`(class) spawns numerous variants involving the same molecule binding to multiple atoms within the same rule body.
In practice, we recommend using disjoint semantics as the default for production. This choice yields predictable performance (runtime stays stable regardless of the recursion budget), improves rule quality by emphasizing cross-table dependencies over self-join variants, keeps memory usage steady by controlling complexity, and makes conservative recursion limits unnecessary.
Non-disjoint semantics should be reserved for specialized analytical scenarios that explicitly require reasoning about multiple instances of the same entity within a single rule, a rare requirement in typical data discovery workflows.

**Broader Impact on Rule Mining Approaches**

This ablation study highlights a fundamental trade-off in association rule mining for relational databases. Traditional approaches borrowed from frequent itemset mining prioritize exhaustiveness, often at the cost of combinatorial explosion when applied to rich relational schemas. Our disjoint semantics demonstrate that *semantic constraints can dramatically improve both efficiency and result quality* by aligning the search space with the underlying data model.
The 100+ fold runtime improvement achieved by disjoint semantics suggests that future rule mining systems should incorporate schema awareness as a first-class design principle, rather than treating relational structure as an afterthought.

### 5.5.2 Ablation Study: Joinability Scope (FK-only vs full joinability)

We compare **FK-only** joinability (declared foreign keys only) to a **full joinability** variant that augments the constraint graph with a small number of high-confidence attribute-equality edges per relation.

**Definition (full joinability).** Let $G = (V, E)$ be the FK-only constraint graph. The *full-joinability* (pairwise-joinability) variant $G^+ = (V^+, E^+)$ assumes that *every attribute is joinable with every other attribute in the database*.

Thus every attribute is considered joinable to every other attribute of the same type domain; no distinct-value overlap tests and no degree cap are applied. By construction, $V \subseteq V^+$ and $E \subseteq E^+$.

**Setup.** Same thresholds as the main evaluation; Metrics: runtime, memory, and rules discovered. We keep the recursion depth fixed across conditions to isolate the impact of the joinability scope.

**Findings.** At a fixed recursion depth $\ell_{\max}$, switching from FK-only joinability to full joinability leaves the number of discovered rules essentially unchanged. The principal effect of the broader join set is an increase in branching, which enlarges the path frontier and in turn raises runtime and peak memory. This trend is consistent with Table 5.2: rule counts remain stable at fixed depth, and even computational time and cost did not change.

**Why disjoint semantics makes FK-only and full joinability equivalent in practice.** Under *disjoint semantics*, any repeated occurrence of the same relation must bind to distinct primary-key tuples. At a fixed recursion depth, this requirement neutralizes the additional equalities introduced by full joinability, so the set of non-isomorphic rules coincides with the FK-only setting. In our ablations, the operational profile also remains essentially unchanged: time and peak memory are statistically indistinguishable between FK-only and full joinability. This indicates a saturation effect in MAHILDA. The search already explores the join skeletons that matter under FK edges, and the extra same-type equalities rarely produce new admissible bodies before pruning.

Formally, for most datasets we observe

$$\text{JoinableMax}(\mathcal{S}) \approx \text{FK}(\mathcal{S}) \quad \text{(up to canonical isomorphism and under depth } \ell_{\max}),$$

which explains the alignment in both rule counts and resources.

**Ablation conclusion.** Across all datasets, and for a fixed maximum walk length $\ell_{\max}$, restricting joinability to foreign-key (FK) edges or allowing full attribute-equality joinability yields the *same set of non-isomorphic rules* with *indistinguishable runtimes and peak memory*. This suggests that, once the search is stabilized by (i)disjoint semantics (distinct occurrences of a relation must bind to different primary-key tuples), (ii) canonicalization , and (iii) empty-join prediction (early detection of zero-support joins), the additional equalities permitted by full joinability do not produce new canonical candidates. Operationally, the candidate space is already saturated within the FK neighborhood.

By contrast, *enabling TGDs* (existential variables and/or multi-atom heads) genuinely enlarges the hypothesis class: it yields derivations that are *unreachable in the Horn-only setting*, increasing both the number of discovered rules and the computational work. In short, with the above stabilizers in place, the choice of *expressivity* (Horn vs. TGDs) has far greater impact than the choice of *joinability regime* (FK-only vs. full).

## 5.6 Chapter summary

MAHILDA is presented as a practical and efficient system for mining single-head Horn rules directly over multi-table relational schemas. A cornerstone of its design is the adoption of **disjoint semantics**, a principled approach that suppresses vacuous self-joins and ensures evidence is measured consistently under closed-world assumptions. This methodology enhances the quality and relevance of the discovered rules.

The system's architecture compiles the database schema into a constraint graph of equi-joinable attributes. Candidate rules are then enumerated via a memory-streaming Depth-First Search (DFS), which is made highly efficient through canonicalization to eliminate isomorphic duplicates, alongside aggressive pruning and memoization. By default, the hypothesis space is constrained to bounded-length Horn rules, which can be recursive, and are restricted to joins reachable through foreign-key paths. This conservative configuration, combining shallow recursion limits with disjoint semantics, results in precise rules and remarkably stable runtimes.

Our empirical evaluation validates this design. The disjoint semantics were demonstrated to significantly reduce redundant rule variants and stabilize runtime performance across different recursion budgets. The DFS strategy proved to synergize effectively with branch-and-bound pruning techniques. While expanding the joinability scope beyond foreign keys can increase recall, this comes at the direct expense of higher latency.

For practical applications, we recommend a default configuration that enables disjointness, utilizes foreign-key-only joinability with conservative depth limits, and employs a DFS search. The principal limitations of MAHILDA are its sensitivity to the quality of declared foreign keys and the deliberate omission of patterns that are not reachable through these established schema paths.

# Chapter 6

# General Conclusion

This thesis examined how to surface multi-relation regularities that remain implicit in real relational databases. We cast discovery as mining *tuple-generating dependencies* (TGDs) directly over tabular data and defined *support* and *confidence* using explicit, auditable counts that are computed inside the database system. Unlike pipelines that outsource semantics to black-box learners, our approach reasons in the schema's vocabulary, reports deterministic statistics, and yields compact and interpretable rule sets that are suitable for logic-based reasoning and governance.

### Addressing the Research Questions

This thesis was organized around five research questions, each of which receives a concrete answer:

- **RQ1: Can expressive TGDs be mined directly from relational schemas at scale?** Yes. MATILDA achieves sub-second discovery on complex schemas by exploiting the constraint graph and duplicate-free enumeration. Experiments on large, real-world databases and synthetic benchmarks demonstrate that mined rules remain interpretable and cover meaningful cross-table patterns.

- **RQ2: How can we define auditable and stable evidence metrics for TGD mining in the presence of joins and recursion?** By enforcing disjoint semantics and computing support as head coverage and confidence under the closed-world assumption. These metrics are deterministic, stable under key-preserving rewrites, and portable across database engines.

- **RQ3: What role does the constraint graph play in taming combinatorial explosion?** The constraint graph bounds branching by restricting candidate construction to semantically justified joins. This reduces the hypothesis space by orders of magnitude without sacrificing coverage, provided joinability is carefully curated.

- **RQ4: In what settings do recursive dependencies arise and add material value, and what evaluation/counting semantics prevent trivial or self-justifying rules?**

  Recursive dependencies often arise in the presence of self–foreign keys. The pruning mechanisms introduced in Section 4.5 (*structural*, via qualified occurrences and the $N$-recursive constraint graph) and Section 5.2.1 (*semantic*, via disjoint semantics for Horn rules) eliminate vacuous, self-witnessing rules.

- **RQ5: How do recursion controls, joinability regimes, and sampling strategies influence accuracy and latency?** Ablation studies quantify these effects: shallow recursion (depth 1–2) suffices for most workloads, foreign-key paths outweigh cross-domain joins, and row sampling accelerates early exploration with minimal accuracy loss.

### Summary of Contributions

We introduced MATILDA, a unified search framework that enumerates candidate rules over a *constraint graph* formed by relations and joinable attributes. Canonical generation and sound pruning keep enumeration duplicate-free and well-behaved as schemas grow, while preserving expressiveness and result quality. We extended mining to *recursive* TGDs and stabilized counting with *disjoint semantics*, which prevent self-witnessing when a relation reoccurs. Joinability is operationalized by restricting construction to attributes connected by declared foreign-key paths or a curated join graph, aligning the hypothesis space with practitioner-intended joins and reducing combinatorics. We also instantiated MAHILDA, a practical miner for single-head Horn rules under the same disjointness assumptions, to provide a clean baseline against ILP systems and enable straightforward deployment in pipelines that consume Horn clauses. Finally, we released an open, SQL-native evaluation harness with reproducible protocols and ablations that relate runtime and coverage to joinability regimes, recursion depth, indexing, and sampling strategies.

### Discussion and Positioning

The proposed framework bridges three lines of work that are often treated separately. First, data-profiling tools excel at exact or approximate single-family constraints but have narrow expressivity. Second, ILP and knowledge-graph miners discover Horn-style rules with learned or probabilistic semantics that complicate auditability. Third, classical database dependencies give strong logical guarantees but are rarely mined directly from operational schemas at scale. By aligning search with schema structure through the constraint graph and by enforcing disjoint counting for recursion, the thesis offers a synthesis: expressive rules with deterministic evidence and a systems pathway that remains faithful to SQL execution.

### Algorithmic and Theoretical Insights

Two ideas proved central. The first is *duplicate-free candidate generation* over simple paths in the constraint graph. This reduces the search to a canonical basis and avoids redundant revisits of syntactic variants. The second is *joinability-guarded exploration*, which bounds branching as a function of key-path degree and attribute fan-out rather than raw schema size. Under a fixed recursion depth and a bounded out-degree in the join graph, enumeration admits practical worst-case guarantees that match our empirical behavior. Disjoint semantics remove vacuous recursive patterns and yield evidence that is stable under renaming and key-preserving rewrites. These properties make the statistics portable across engines and runs.

### Empirical Findings

Across representative benchmarks that include inclusion-dependency discovery, knowledge-graph rule mining, and ILP baselines, constraining exploration to foreign-key or curated paths combined with shallow body and head walks delivers consistent speedups while preserving or improving

coverage within the space of *semantically joinable* hypotheses. Duplicate-free generation reduces wasted evaluations and improves time-to-first rule. Ablations show how recursion controls, pruning policies, indexing, and caching shape the accuracy–cost trade-off. Results indicate that expressive cross-table regularities can be mined in place, without export to external learning stacks, and can be translated into actionable and auditable constraints.

### Future Directions

One promising direction is to strengthen the robustness of join discovery by augmenting the joinability graph with mined dependencies and calibrated confidence bounds,and by recording complete provenance for each mined join edge, covering data and code versions, miner parameters, evidence tuples, and confidence bounds, so that the edge can be traced from raw source rows to every downstream use. A second avenue is to develop a unified dependency-mining framework that jointly discovers TGDs, denial constraints, and numeric rules; one could remedy the current fragmentation by learning these constraints together, thereby enabling stronger, cross-type reasoning for data cleaning and value imputation. To meet the demands of modern, evolving datasets, the algorithms should be adapted into *anytime* and *streaming* variants that maintain rule catalogs under dynamic updates with explicit guarantees. Efficiency can be further improved through reasoning-aware pruning: lightweight chase tests and other inexpensive consequence checks would prioritize candidates with the highest marginal impact on downstream inference and discard dominated branches early. On the systems side, significant performance gains are likely from acceleration techniques such as parallel multi-join execution, compressed columnar layouts with vectorized operators, and selective GPU offloading for batched joins and statistics maintenance. For real-world adoption, operational integration is essential: rule catalogs should be linked to governance workflows, versioning, and impact analysis so that discovery, review, deployment, and enforcement form a closed loop. This loop can be enriched by *human-in-the-loop* refinement, where interactive feedback from domain experts steers exploration, calibrates thresholds, and resolves borderline cases to improve precision. Finally, a particularly powerful direction lies at the interface with representation learning: symbolic rules can guide, regularize, and explain machine-learning models, while embeddings can suggest promising regions of the hypothesis space for rule discovery, always with auditable evidence retained within the database.

### Closing Remarks

Mining expressive TGDs directly over relational data is feasible and useful at scale. By aligning the hypothesis space with schema semantics, enforcing principled counting for recursive patterns, and validating behavior on diverse SQL-native workloads, this work outlines a practical path from implicit cross-relation structure to explicit and testable knowledge. We hope that constraint-graph exploration, disjoint semantics, the Horn-rule baseline, and the open evaluation harness will help unify dependency discovery with scalable data systems and governance-oriented analytics, and that they will serve as a foundation for the next generation of auditable, data-centric AI.

# Chapter A

# Database Examples

The CTU Prague Relational Learning Repository hosts a diverse collection of relational databases. Below are a few notable examples along with UML-like diagrams that illustrate their schemas (columns and relationships).

## A.1 Stats Stack Exchange Dataset

- **Domain:** Education
- **Size:** 658.4 MB
- **Number of Tables:** 8
- **Task:** Regression
- **Description:** An anonymized dump of all user-contributed content on the Stats Stack Exchange network. This dataset includes user interactions, questions, answers, and associated metadata, making it suitable for analyzing user behavior and content trends.

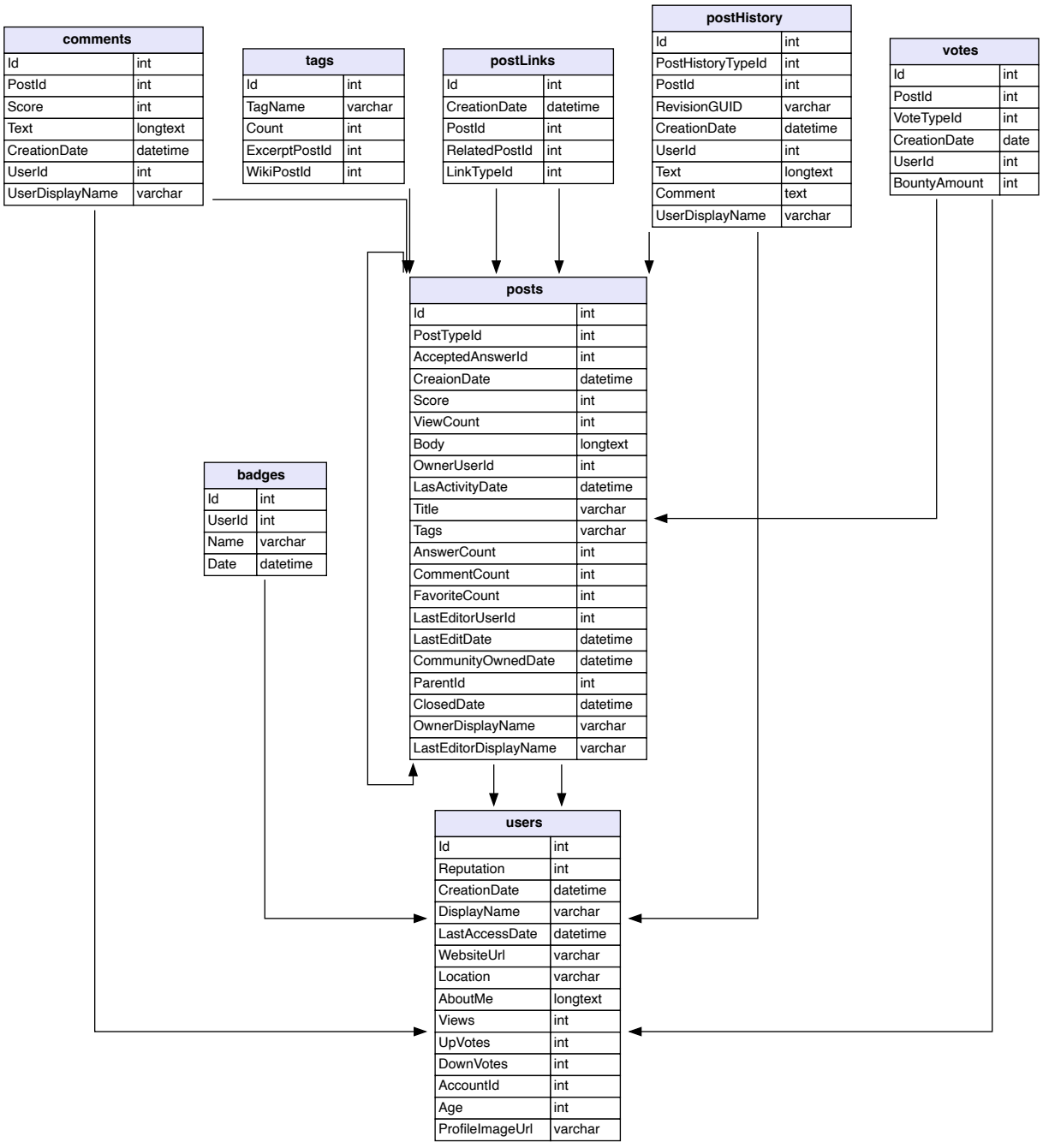


**Figure A.1:** UML-style diagram of the Stats Stack Exchange dataset schema.

## A.2 Cora Dataset

- **Domain:** Education
- **Size:** 4.5 MB
- **Number of Tables:** 3
- **Task:** Classification
- **Description:** The Cora dataset consists of 2,708 scientific publications classified into one of seven classes. The citation network includes 5,429 links, and each publication is described by a binary word vector indicating the presence of specific words.

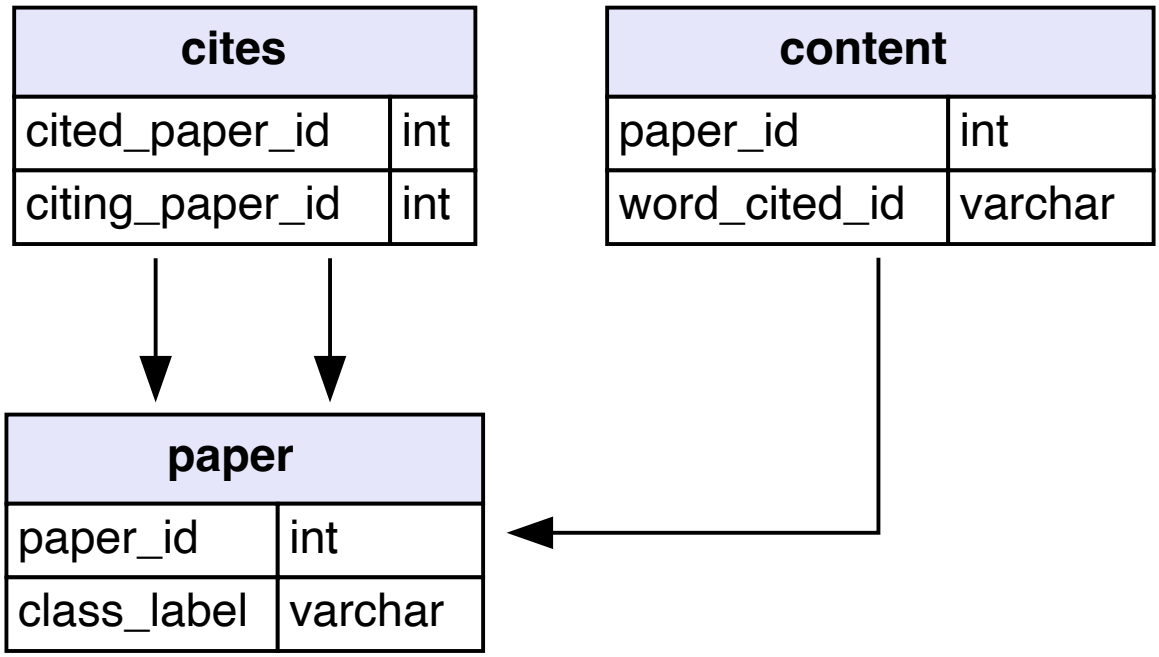


**Figure A.2:** UML-style diagram of the Cora dataset schema.

## A.3 Hepatitis Dataset

- **Domain:** Medicine
- **Size:** 2.2 MB
- **Number of Tables:** 7
- **Task:** Classification
- **Description:** The PKDD'02 Hepatitis dataset describes 206 instances of Hepatitis B and 484 cases of Hepatitis C. The data includes patient information, laboratory results, and other medical attributes, facilitating research in medical diagnosis and disease classification.

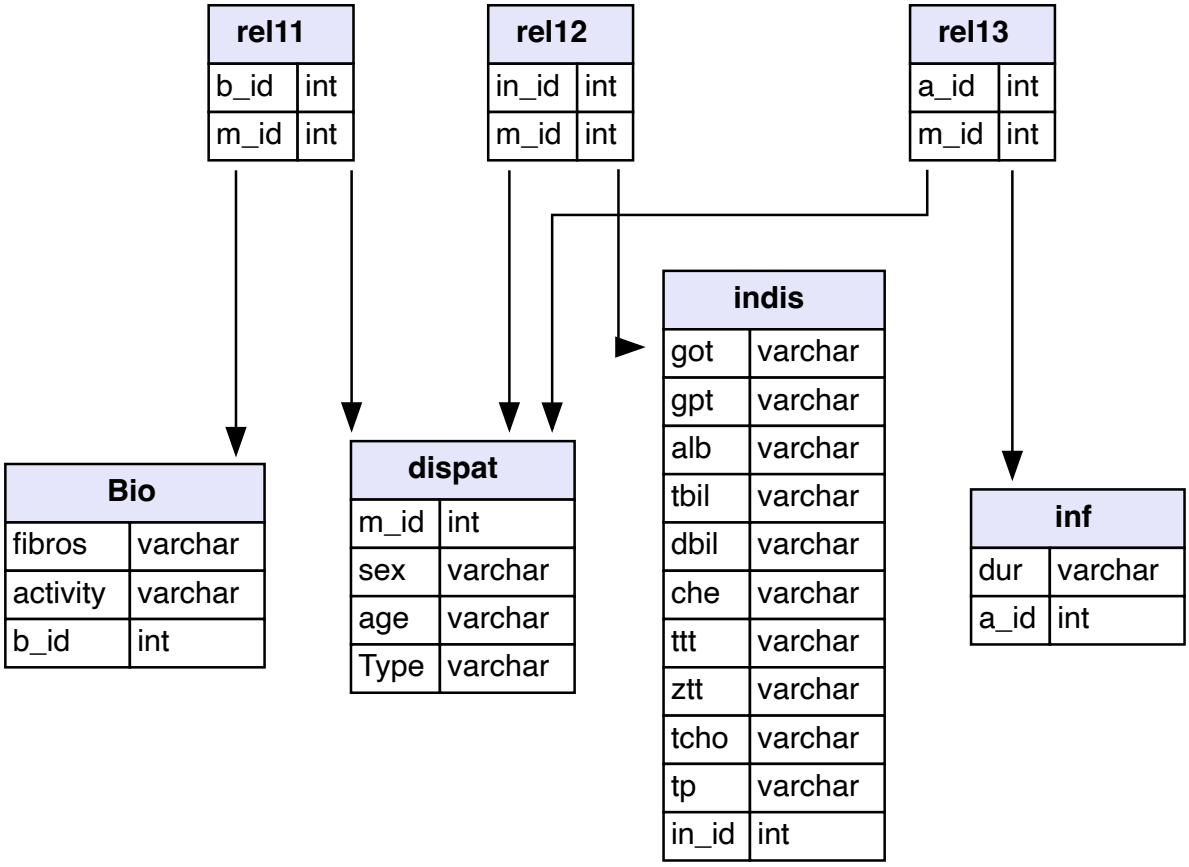


**Figure A.3:** UML-style diagram of the Hepatitis dataset schema.

## A.4 Genes Dataset

- **Domain:** Medicine
- **Size:** 1.8 MB
- **Number of Tables:** 3
- **Task:** Classification
- **Description:** The KDD Cup 2001 Genes dataset focuses on the prediction of gene/protein function and localization. It includes data on gene expression levels, functional annotations, and other biological attributes, supporting research in genomics and bioinformatics.

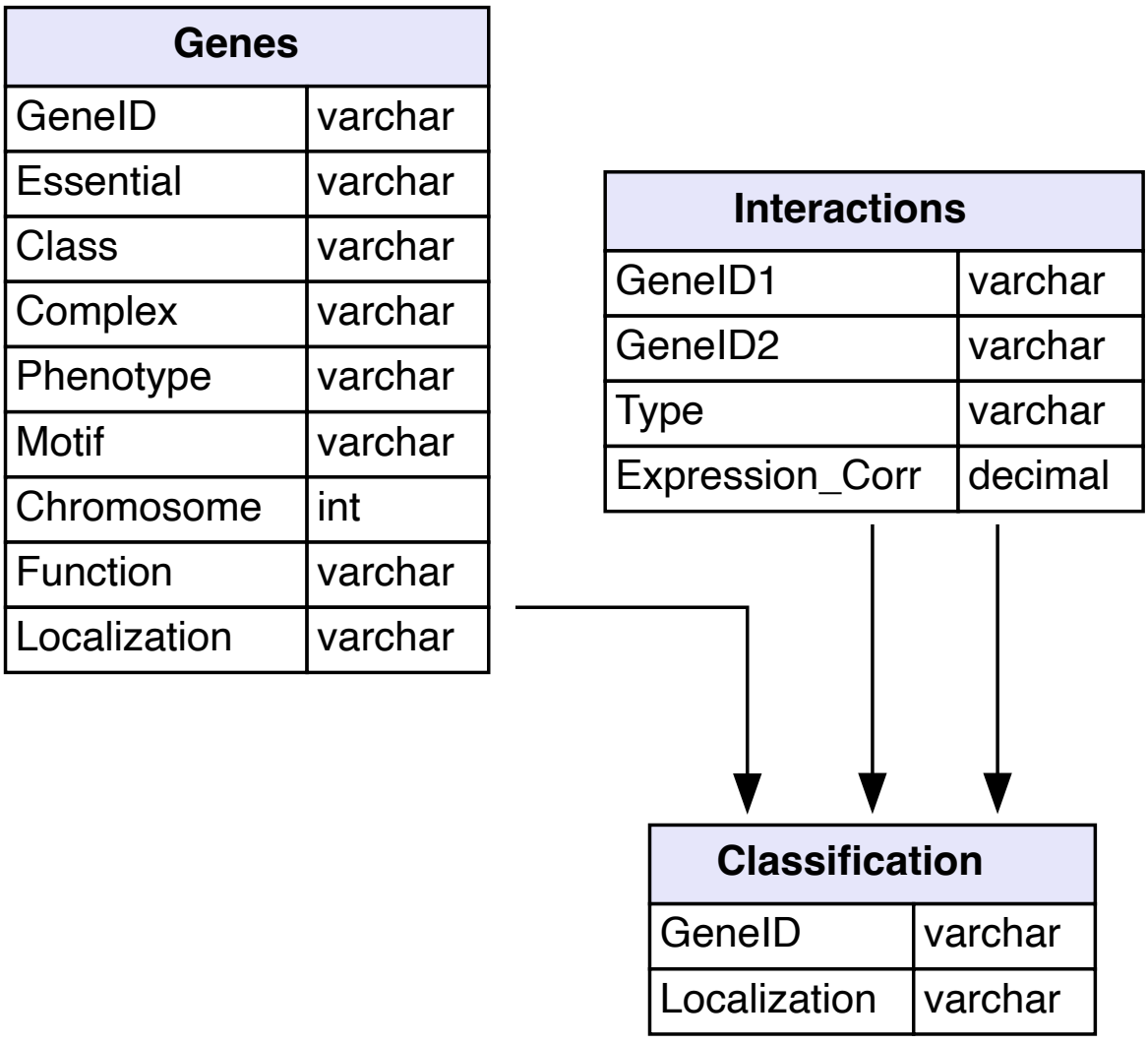


**Figure A.4:** UML-style diagram of the Genes dataset schema.

## A.5 UW-CSE Dataset

- **Domain:** Education
- **Size:** 200 KB
- **Number of Tables:** 4
- **Task:** Classification
- **Description:** This dataset lists facts about the Department of Computer Science and Engineering at the University of Washington (UW-CSE), including entities such as students and professors, and their relationships like advisorships and publications.

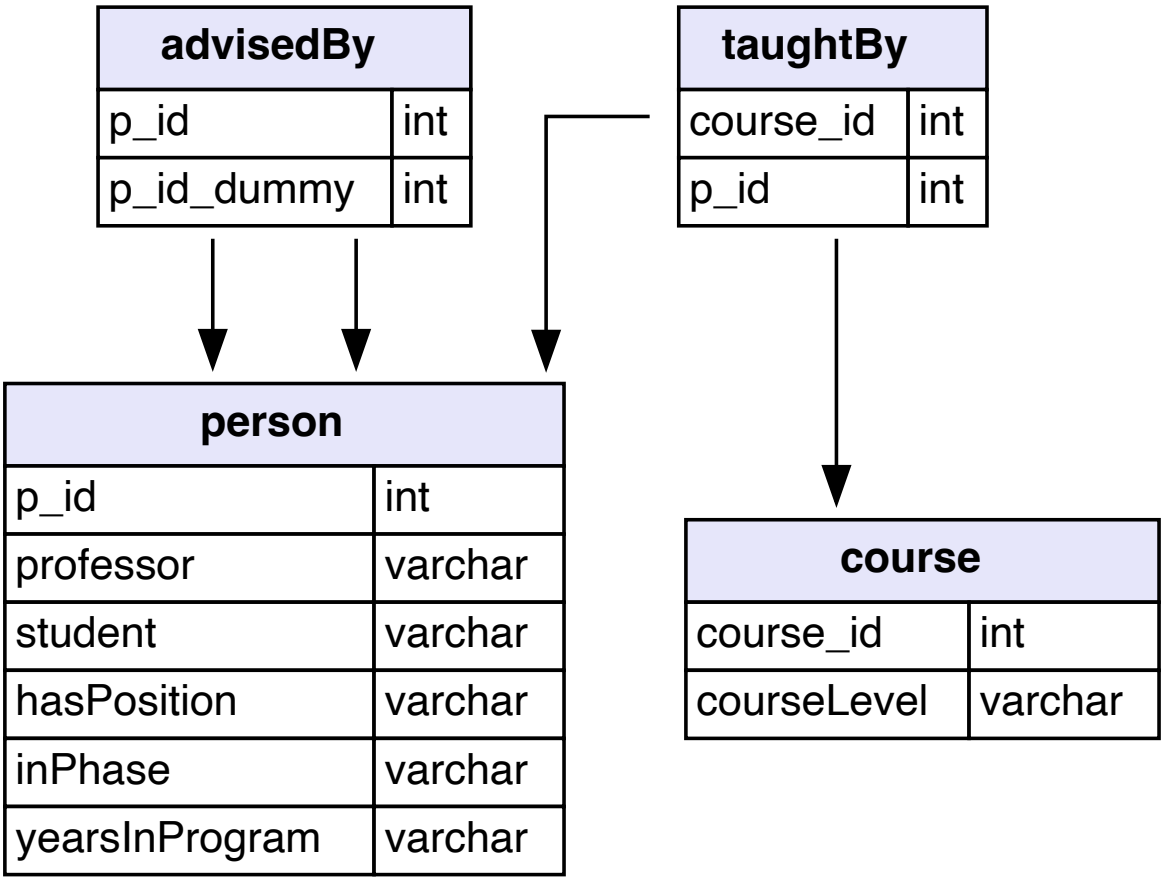


**Figure A.5:** UML-style diagram of the UW-CSE dataset schema.

# Annexes B

# Résumé en Français

Cette thèse s'attaque à un problème central de l'ingénierie des données et de la fouille de connaissances : découvrir automatiquement des dépendances logiques expressives et auditables directement au-dessus de schémas relationnels réels. Alors que les bases de données encodent des contraintes structurelles explicites, nombre de règles métier critiques restent implicites, engendrant des défauts de qualité, des incohérences applicatives et une conformité réglementaire fragilisée. La littérature exploite principalement des règles de Horn ou des graphes de connaissances, souvent déconnectés des schémas multi-tables et de leurs contraintes de jointure. Or, de nombreux phénomènes métiers requièrent des règles plus riches, telles que les dépendances génératrices de tuples (TGDs) à têtes multi-atomiques avec témoins existentiels partagés, formalisées par $\phi(\vec{x}, \vec{y}) \rightarrow \exists \vec{z}(\psi_1(\vec{x}, \vec{z}) \wedge \cdots \wedge \psi_k(\vec{x}, \vec{z}))$. La découverte de telles règles soulève des défis majeurs : l'explosion combinatoire de l'espace de recherche, l'évaluation fiable de leur support et confiance en contexte réel, et leur opérationnalisation industrielle. Pour relever ces défis, nous proposons un cadre méthodologique *in situ* qui articule une formalisation unifiée des TGDs connectées, des métriques robustes (support, confiance) respectant une symétrie corps/tête, des stratégies de génération et d'élagage adaptées aux contraintes de joignabilité (*joinability*), et des options sémantiques comme la récursivité contrôlée pour explorer différentes familles de règles.
La contribution centrale est un cadre de fouille déterministe, opérant sur toute base de données relationnelle, capable de découvrir à la fois des TGDs d'ordre supérieur avec l'algorithme *MATILDA*, et des règles de Horn multi-relations avec *MAHILDA*. Contrairement aux approches heuristiques ou de type boîte noire, nos méthodes garantissent des résultats reproductibles en fondant l'évidence, le support et la confiance directement dans la sémantique de la base de données. Elles généralisent les contraintes déclarées pour capturer des implications inter-tables complexes, tout en restant calculables grâce à une recherche guidée par le schéma et un élagage raisonné. Le résultat est un mécanisme unifié et auditable pour révéler des dépendances cachées, suffisamment expressives pour modéliser une logique métier réaliste et structurées pour s'intégrer à des pipelines de raisonnement formel et de conformité.
L'évaluation expérimentale sur une large collection de bases de données démontre la capacité de MATILDA à découvrir des TGDs complexes dans un temps prévisible, surpassant les approches concurrentes en termes de performance et d'explicabilité. Les expériences avec MAHILDA confirment que l'introduction d'une sémantique disjointe est cruciale pour maîtriser la complexité, éviter le surapprentissage et garantir une mise à l'échelle efficace lors de la fouille de règles récursives. L'impact industriel de ces travaux est direct, avec des cas d'usage allant de la

validation et l'imputation de données à la documentation prête pour l'audit et à l'amélioration de modèles d'apprentissage automatique. En conclusion, cette thèse démontre que la fouille de TGDs expressives sur des schémas relationnels est un levier robuste pour révéler des connaissances implicites, renforcer la qualité des données et outiller la gouvernance des systèmes d'IA. Le cadre proposé, alliant expressivité et performance, offre des réglages par défaut réalistes et des chemins d'extension contrôlés. Les perspectives de recherche incluent l'intégration de classes de contraintes supplémentaires (EGDs, DCs, ODs), des optimisations système (parallélisation, pré-filtrage), le couplage avec les pipelines MLOps pour la détection de dérive, l'industrialisation de la gouvernance continue des règles, et l'extension des bancs d'essai à des lacs de données hétérogènes.

# List of Figures

# List of Tables

# References


[1] E. F. Codd. “A Relational Model of Data for Large Shared Data Banks”. In: *Communications of the ACM* 13.6 (1970), pp. 377–387. DOI: 10.1145/362384.362685.

[2] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*. Vol. 8. France, Rocquencourt: Addison-Wesley Reading, 1995.

[3] C. J. Date. *SQL and Relational Theory*. 2nd. O’Reilly, 2012.

[4] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume I*. Computer Science Press, 1988.

[5] DB-Engines. *Method of calculating the scores of the DB-Engines Ranking*. https://db-engines.com/en/ranking_definition. 2025.

[6] DB-Engines. *DB-Engines Ranking*. https://db-engines.com/en/ranking. 2025.

[7] DB-Engines. *DB-Engines Ranking by Database Model Category*. https://db-engines.com/en/ranking_categories. 2025.

[8] *TPC Benchmark™ H (TPC-H) Standard Specification*. Tech. rep. Revision 2.18.0. Transaction Processing Performance Council (TPC), 2018. URL: https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.18.0.pdf.

[9] *TPC Benchmark™ DS (TPC-DS) Standard Specification*. Tech. rep. Version 3.2.0. Transaction Processing Performance Council (TPC), 2021. URL: https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v3.2.0.pdf.

[10] *CTU Prague Relational Learning Repository*. https://relational.fit.cvut.cz/.

[11] François Amat, Pierre-Henri Paris, Paolo Papotti, and Fabian M. Suchanek. “MATILDA: Mining Approximate Tuple-Generating Dependencies in Large Databases”. Manuscript submitted to ACM TIST. 2025. URL: https://github.com/fran-cois/matilda.

[12] François Amat, Pierre-Henri Paris, Paolo Papotti, and Fabian M. Suchanek. “MAHILDA: Mining Approximate Horn Rules in Large Databases”. Manuscript in preparation; to be submitted. 2025. URL: https://github.com/fran-cois/mahilda.

[13] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Reading, MA: Addison-Wesley, 1995. ISBN: 0-201-53771-0.

[14] David Maier. *The Theory of Relational Databases*. Rockville, MD: Computer Science Press, 1983. ISBN: 0-914894-42-0.

[15] Konstantin Haller and Wolf-Tilo Balke. “Efficient Discovery of Inclusion Dependencies”. In: *Proceedings of the 15th International Conference on Extending Database Technology (EDBT) Workshops*. 2012, pp. 342–349.

[16] Thorsten Papenbrock, Jörg Zwiener, and Felix Naumann. “HyUCC: Hybrid Unique Column Combination Discovery”. In: *Proceedings of the 2015 ACM CIKM International Conference on Information and Knowledge Management*. 2015, pp. 131–140.

[17] Thorsten Papenbrock and Felix Naumann. “A Hybrid Approach to Functional Dependency Discovery”. In: *Proc. ACM SIGMOD International Conference on Management of Data*. 2015, pp. 821–833. DOI: 10.1145/2723372.2747644.

[18] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. “Holistic Data Cleaning: Putting Violations into Context”. In: *Proc. IEEE 29th International Conference on Data Engineering (ICDE)*. 2013, pp. 458–469. DOI: 10.1109/ICDE.2013.6544830.

[19] Thorsten Papenbrock, Arvid Heise, and Felix Naumann. “Progressive FD Discovery with HyFD”. In: *Proc. VLDB Endow.* 8.11 (2015), pp. 1058–1069. DOI: 10.14778/2794367.2794377.

[20] Jaroslaw Szlichta, Parke Godfrey, and Jarek Gryz. “Fundamentals of Order Dependencies”. In: *Proceedings of the VLDB Endowment* 5.11 (2012), pp. 1220–1231. DOI: 10.14778/2350229.2350241.

[21] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. “TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies”. In: *The Computer Journal* 42.2 (1999), pp. 100–111.

[22] Ronald Fagin, Laura M. Haas, Mauricio Hernández, Renée J. Miller, Lucian Popa, and Yannis Velegrakis. “Clio: Schema Mapping Creation and Data Exchange”. In: *Conceptual Modeling: Foundations and Applications*. Vol. 5600. Lecture Notes in Computer Science. Springer, 2009, pp. 198–236.

[23] Arvid Heise, Jorge-Arnulfo Quiané-Ruiz, Ziawasch Abedjan, Anja Jentzsch, and Felix Naumann. “Scalable Discovery of Unique Column Combinations”. In: *Proc. VLDB Endow.* 7.4 (2013), pp. 301–312.

[24] Jana Bauckmann, Ulf Leser, and Felix Naumann. *Efficient and Exact Computation of Inclusion Dependencies for Data Integration*. HPI Technical Report 34. Potsdam, Germany: Hasso-Plattner-Institut, 2010. URL: https://www.informatik.hu-berlin.de/de/forschung/gebiete/wbi/research/publications/2010/hpi_34_efficient_and_exact_computation_of_inclusion_dependencies_for_data_integration.pdf.

[25] J. Szlichta, P. Godfrey, L. Golab, M. Kargar, and D. Srivastava. “Effective and Complete Discovery of Order Dependencies via Set-Based Axiomatization”. In: *Proc. VLDB Endow.* 10.7 (2017), pp. 721–732.

[26] Edgar F. Codd. “Extending the Database Relational Model to Capture More Meaning”. In: *ACM Trans. Database Syst.* 4.4 (1979), pp. 397–434.

[27] Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. "Conditional Functional Dependencies for Data Cleaning". In: *Proceedings of the 23rd International Conference on Data Engineering (ICDE)*. 2007, pp. 746–755.

[28] Wenfei Fan, Floris Geerts, and Xibei Jia. "A Revival of Integrity Constraints for Data Cleaning". In: *Proceedings of the VLDB Endowment* 1.2 (2008), pp. 1522–1523.

[29] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. "Mining Association Rules between Sets of Items in Large Databases". In: *Communications of the ACM* 36.6 (1993), pp. 67–80.

[30] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. 2nd. Morgan Kaufmann, 2004.

[31] Ronald Fagin. "A normal form for relational databases that is based on domains and keys". In: *ACM Transactions on Database Systems (TODS)*. Vol. 6. 3. 1981, pp. 387–415.

[32] Phokion G. Kolaitis and Moshe Y. Vardi. "Conjunctive-Query Containment and Constraint Satisfaction". In: *Journal of Computer and System Sciences (JCSS)* 61.2 (2000), pp. 302–332.

[33] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang-Chiew Tan. "Composing Schema Mappings: Second-Order Dependencies to the Rescue". In: *ACM Transactions on Database Systems* 30.4 (2005), pp. 994–1055.

[34] Andrea Calì, Georg Gottlob, and Andreas Pieris. "Towards More Expressive Ontology Languages: The Query Answering Problem". In: *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning (KR)*. 2012, pp. 217–227.

[35] Andrea Calì, Georg Gottlob, and Michael Kifer. "Taming the Infinite Chase: Query Answering under Expressive Relational Constraints". In: *Journal of Artificial Intelligence Research* 48 (2013), pp. 115–174.

[36] Matthew Richardson and Pedro Domingos. "Markov Logic Networks". In: *Machine Learning* 62.1-2 (2006), pp. 107–136.

[37] Johanna Völker and Mathias Niepert. "Statistical Schema Induction". In: *Proceedings of the 8th Extended Semantic Web Conference (ESWC)*. 2011, pp. 124–138.

[38] Peter Spirtes and Clark Glymour. "An Algorithm for Fast Recovery of Sparse Causal Graphs". In: *Social Science Computer Review* 9.1 (Apr. 1991), pp. 62–72. DOI: 10.1177/089443939100900106.

[39] Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2000.

[40] Xun Zheng, Bryan Aragam, Pradeep Ravikumar, and Eric P. Xing. "DAGs with NO TEARS: Continuous Optimization for Structure Learning". In: *Advances in Neural Information Processing Systems*. Vol. 31. 2018, pp. 9472–9483.

[41] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Durán, Jason Weston, and Oksana Yakhnenko. "Translating Embeddings for Modeling Multi-relational Data". In: *Advances in Neural Information Processing Systems (NIPS)*. 2013, pp. 2787–2795.

[42] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. *Embedding Entities and Relations for Learning and Inference in Knowledge Bases*. arXiv:1412.6575 [cs.CL]. 2015.

[43] N'Dah Jean Kouagou, Arif Yilmaz, Michel Dumontier, and Axel-Cyrille Ngonga. *Discovering the unknown: Improving rule mining via embedding-based link prediction*. arXiv:2406.10144 [cs.AI]. 2024.

[44] Johanna Jøsang, Ricardo Guimarães, and Atsushi Ozaki. *On the Effectiveness of Knowledge Graph Embeddings: a Rule Mining Approach*. arXiv:2206.00983 [cs.LG]. 2022.

[45] Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. "AMIE: Association Rule Mining under Incomplete Evidence in Ontological Knowledge Bases". In: *Proceedings of the 22nd International World Wide Web Conference (WWW)*. 2013, pp. 413–422.

[46] J. R. Quinlan. "Learning Logical Definitions from Relations". In: *Machine Learning* 5 (1990), pp. 239–266.

[47] Stephen Muggleton. "Inverse Entailment and Progol". In: *New Generation Computing* 13.3-4 (1995), pp. 245–286.

[48] Andrew Cropper and Rolf Morel. "Learning Programs by Learning from Failures". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04. 2020, pp. 3678–3686. DOI: 10.1609/aaai.v34i04.5778.

[49] Larissa C. Shimomura, Nikolay Yakovets, and George Fletcher. "Discovering Graph Generating Dependencies for Property Graph Profiling". In: *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM)*. 2024. DOI: 10.1145/3627673.3679764.

[50] Wenfei Fan, Wenzhi Fu, Ruochun Jin, Ping Lu, and Chao Tian. "Discovering Association Rules from Big Graphs". In: *Proceedings of the VLDB Endowment* 15.7 (2022), pp. 1479–1492. DOI: 10.14778/3523210.3523224.

[51] Céline Hocquette, Andreas Niskanen, Rolf Morel, Matti Järvisalo, and Andrew Cropper. "Learning Big Logical Rules by Joining Small Rules". In: *Proceedings of the 33rd International Joint Conference on Artificial Intelligence (IJCAI-24)*. 2024, pp. 3430–3438.

[52] Xiaoou Ding, Yixing Lu, Hongzhi Wang, Chen Wang, Yida Liu, and Jianmin Wang. "DAFDiscover: Robust Mining Algorithm for Dynamic Approximate Functional Dependencies on Dirty Data". In: *Proceedings of the VLDB Endowment* 17.11 (2024), pp. 3484–3496. DOI: 10.14778/3681954.3682015.

[53] Simon Ott, Christian Meilicke, and Matthias Samwald. "SAFRAN: An interpretable, rule-based link prediction method outperforming embedding models". In: *Proceedings of the 3rd Conference on Automated Knowledge Base Construction (AKBC)*. 2021. URL: https://openreview.net/forum?id=fOom-6sN3.

[54] Pouya Ghiasnezhad Omran, Kewen Wang, and Zhe Wang. "Scalable Rule Learning via Learning Representation". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization. 2018, pp. 2149–2155. DOI: 10.24963/ijcai.2018/297.

[55] Ziyue Xu, Long-Kai Huang, Yizhou Sun, and Wei Wang. "Noether Embedding: A Data-Efficient and Time-Efficient Event Learner". In: *Thirty-seventh Conference on Neural Information Processing Systems*. 2023. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/96c6f409a374b5c81d2efa4bc5526f27-Paper-Conference.pdf.

[56] Jana Bauckmann, Ulf Leser, Felix Naumann, and Véronique Tietz. "Efficiently Detecting Inclusion Dependencies". In: *Proceedings of the 23rd International Conference on Data Engineering (ICDE)*. IEEE Computer Society, Apr. 2007, pp. 1448–1450. DOI: 10.1109/ICDE.2007.369032.

[57] Jonathan Lajus, Luis Galárraga, and Fabian M. Suchanek. "Fast and Exact Rule Mining with AMIE 3". In: *The Semantic Web – 17th International Conference, ESWC 2020, Heraklion, Crete, Greece, May 31–June 4, 2020, Proceedings*. Vol. 12123. Lecture Notes in Computer Science. Springer, 2020, pp. 36–52. DOI: 10.1007/978-3-030-49461-2_3.

[58] Jonathan Lajus, Luis Galárraga, and Fabian M. Suchanek. "Fast and Exact Rule Mining with AMIE 3". In: *The Semantic Web – 17th International Conference, ESWC 2020, Heraklion, Crete, Greece, May 31–June 4, 2020, Proceedings*. Vol. 12123. Lecture Notes in Computer Science. Springer, 2020, pp. 36–52. DOI: 10.1007/978-3-030-49461-2_3.

[59] Philip A. Bernstein, Jayant Madhavan, and Erhard Rahm. "Generic Schema Matching, Ten Years Later". In: *Proc. VLDB Endow.* 4.11 (2011), pp. 695–701. URL: http://www.vldb.org/pvldb/vol4/p695-bernstein%5C_madhavan%5C_rahm.pdf.

[60] Andrew Cropper and Rolf Morel. "Learning Programs by Learning from Failures". In: *Machine Learning* 110.4 (2021). First version: May 2020 (arXiv:2005.02259), pp. 801–856. DOI: 10.1007/s10994-020-05934-z. URL: https://doi.org/10.1007/s10994-020-05934-z.

[61] Stefano Ortona, Venkata Vamsikrishna Meduri, and Paolo Papotti. "RuDiK: Rule Discovery in Knowledge Bases". In: *Proceedings of the VLDB Endowment* 11.12 (2018), pp. 1946–1949. DOI: 10.14778/3229863.3236231.

[62] Thorsten Papenbrock, Tanja Bergmann, Moritz Finke, Jakob Zwiener, and Felix Naumann. "Data Profiling with Metanome". In: *Proceedings of the VLDB Endowment (PVLDB)* 8.12 (2015), pp. 1860–1863.

[63] Vojtěch Motl and Oliver Schulte. "The CTU Prague Relational Learning Repository". In: *Proceedings of the Meta-Learning and Algorithm Selection Workshop*. 2015.

[64] Boris Levin, Abraham Meidan, Alex Cheskis, Ohad Gefen, and Ilya Vorobyov. "PKDD'99 Discovery Challenge". In: *Workshop Notes on Discovery Challenge (PKDD'99)*. Ed. by Petr Berka. Workshop at the 3rd European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'99). University of Economics, Prague. Prague, Czech Republic, 1999.

[65] Prithviraj Sen, Gal Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. "Collective Classification in Network Data". In: *AI Magazine* (2008).

[66] Yizhar Regev, Michal Finkelstein-Landau, and Ronen Feldman. “Rule-based Extraction of Experimental Evidence in the Biomedical Domain, the KDD Cup 2002 (Task 1)”. In: *ACM SIGKDD Explorations Newsletter* 4.2 (2002), pp. 90–92. DOI: 10.1145/772862.772874.

[67] Thorsten Papenbrock, Tanja Bergmann, Moritz Finke, Jakob Zwiener, and Felix Naumann. “Data Profiling with Metanome”. In: *Proc. VLDB Endow.* 8.12 (Aug. 2015), pp. 1860–1863. ISSN: 2150-8097. DOI: 10.14778/2824032.2824086. URL: http://dx.doi.org/10.14778/2824032.2824086.

ECOLE
DOCTORALE

**Titre :** Fouille de motifs dans les données tabulaires

**Mots clés :** dépendances génératrices de tuples (TGD), fouille de données relationnelles, règles de Horn, contraintes d'intégrité, découverte de schémas, sémantique disjointe, minage de règles, évaluation empirique, chase, sémantique du monde fermé

**Résumé :** Cette thèse étudie la découverte automatique de motifs expressifs dans les bases de données relationnelles au moyen des dépendances génératrices de tuples (TGDs). Nous formalisons des mesures de qualité adaptées au cadre multi-relationnel (support et confiance), et présentons MATILDA, un algorithme de fouille qui énumère des règles connectées tout en appliquant des bornes de recherche et des stratégies de pruning efficaces. Nous étendons l'approche à la récursivité et à une sémantique disjointe des relations, puis discutons l'articulation avec les EGDs. Une évaluation expérimentale sur des jeux de données réels et synthétiques compare notre méthode aux systèmes de l'état de l'art (par ex. AMIE3, POPPER, SPIDER) en termes de couverture, précision et temps de calcul. Les résultats montrent que l'approche proposée découvre des règles pertinentes à l'échelle, éclaire la conception de schémas et complète les procédures de contrôle de qualité des données. Nous analysons enfin les implications théoriques (classes de TGDs, garanties de complétude et de correction) et esquissons des perspectives vers d'autres familles de contraintes.

**Title :** Mining Patterns on Tabular Data

**Keywords :** tuple-generating dependencies (TGDs), relational pattern mining, Horn rules, integrity constraints, schema discovery, disjoint semantics, rule mining, empirical evaluation, chase, closed-world semantics

**Abstract :** This dissertation investigates the automatic discovery of expressive patterns in relational databases through tuple-generating dependencies (TGDs). We define multi-relational quality measures (support and confidence) and introduce MATILDA, a mining algorithm that enumerates connected rules while applying effective search bounds and pruning strategies. We extend the approach to recursive rules and to a disjoint semantics over relations, and discuss its interplay with EGDs. A comprehensive experimental study on real and synthetic datasets compares our method against state-of-the-art systems (e.g., AMIE3, POPPER, SPIDER) along coverage, precision, and runtime. The results show that our approach scales to large schemas while uncovering meaningful rules that inform schema design and complement data-quality workflows. We further analyze theoretical aspects (TGD subclasses, completeness and soundness guarantees) and outline directions toward additional dependency families.

**Institut Polytechnique de Paris**
91120 Palaiseau, France