

Mining Patterns on Tabular Data

Discovering Expressive Rules from Relational Databases

François Amat

Institut Polytechnique de Paris
Télécom Paris

December 5, 2025

PhD Defense Presentation

Directeurs de thèse:

Prof. Fabian Suchanek
Télécom Paris

Dr. Pierre-Henri Paris
Université Paris-Saclay

Rapporteurs:

Prof. Fatiha Sais
Université Paris-Saclay

Prof. Pierre Senellart
École normale supérieure

Examineurs:

Prof. Bernd Amann
Sorbonne Université

Prof. Nathalie Pernelle
Université Sorbonne Paris Nord

Outline

1 Overview

Outline

- 1 Overview
- 2 Motivation

Outline

- 1 Overview
- 2 Motivation
- 3 Preliminaries

Outline

- 1 Overview
- 2 Motivation
- 3 Preliminaries
- 4 Related Work

Outline

- 1 Overview
- 2 Motivation
- 3 Preliminaries
- 4 Related Work
- 5 Contributions

Outline

- 1 Overview
- 2 Motivation
- 3 Preliminaries
- 4 Related Work
- 5 Contributions
- 6 MATILDA

Outline

- 1 Overview
- 2 Motivation
- 3 Preliminaries
- 4 Related Work
- 5 Contributions
- 6 MATILDA
- 7 Experiments

Outline

- 1 Overview
- 2 Motivation
- 3 Preliminaries
- 4 Related Work
- 5 Contributions
- 6 MATILDA
- 7 Experiments
- 8 MAHILDA

Outline

- 1 Overview
- 2 Motivation
- 3 Preliminaries
- 4 Related Work
- 5 Contributions
- 6 MATILDA
- 7 Experiments
- 8 MAHILDA
- 9 Limitations

Outline

- 1 Overview
- 2 Motivation
- 3 Preliminaries
- 4 Related Work
- 5 Contributions
- 6 MATILDA
- 7 Experiments
- 8 MAHILDA
- 9 Limitations
- 10 Impact

Outline

- 1 Overview
- 2 Motivation
- 3 Preliminaries
- 4 Related Work
- 5 Contributions
- 6 MATILDA
- 7 Experiments
- 8 MAHILDA
- 9 Limitations
- 10 Impact
- 11 Conclusion

The Problem: Hidden Rules in Databases

Challenge

Relational databases contain hidden regularities
not captured by declared constraints

Why this matters:

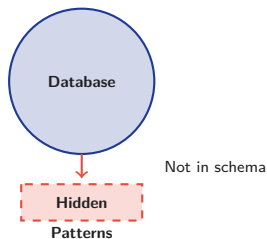
The Problem: Hidden Rules in Databases

Challenge

Relational databases contain hidden regularities not captured by declared constraints

Why this matters:

- Business logic evolves faster than schemas



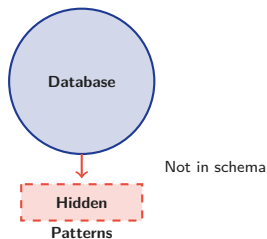
The Problem: Hidden Rules in Databases

Challenge

Relational databases contain hidden regularities not captured by declared constraints

Why this matters:

- Business logic evolves faster than schemas
- Cross-table dependencies remain implicit



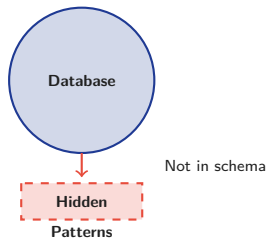
The Problem: Hidden Rules in Databases

Challenge

Relational databases contain hidden regularities not captured by declared constraints

Why this matters:

- Business logic evolves faster than schemas
- Cross-table dependencies remain implicit
- These hidden rules impact:
 - Data quality monitoring
 - Compliance auditing
 - Safe data integration



The Problem: Hidden Rules in Databases

Challenge

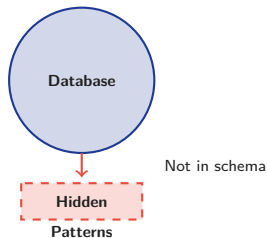
Relational databases contain hidden regularities not captured by declared constraints

Why this matters:

- Business logic evolves faster than schemas
- Cross-table dependencies remain implicit
- These hidden rules impact:
 - Data quality monitoring
 - Compliance auditing
 - Safe data integration

Our Goal

Automatically discover **expressive**, **auditable** rules from relational data



Concrete Example: Invoice vs Shipment (ERP Terms)

Common belief: Shipped orders imply recorded payment.

Reality in many ERPs

Shipments precede payment (Net-30/Net-60, consolidated invoices, partial shipments).

Declared FKs don't encode these business terms.

Expressive rule we discover

```
ITEMS(oid,...) ∧ SHIPPING(oid, status='Completed') ∧ TERMS(oid, net30)
⇒ ∃inv. INVOICE(inv, oid) ∧ DUE_BY_TERMS(inv, oid, net30)
Captures why payment may be missing now, and what must exist instead.
```

Insight: "Missing payment" flags are often false positives; the real dependency concerns *billing terms* and *due artifacts*. TGDs fit the semantics.

Impact

Fewer false alarms, auditable SLA checks, cleaner compliance dashboards.

Counterintuitive Example: Healthcare

No diagnosis? Still valid

Belief: Every prescription must have a diagnosis code.

Reality: Vaccinations and prophylaxis often lack diagnosis but require encounter & consent.

$\text{ENCOUNTER}(e,p) \wedge \text{CONSENT}(p,e) \Rightarrow \exists m,d. \text{MED}(p,m,e) \wedge \text{PHARMACY_DISPENSE}(d,p,m,e)$

✓ Takeaway

TGDs capture required *artifacts* (encounter, consent), not simplistic diagnosis link.

Counterintuitive Example: Finance

Large transaction always manager-approved?

Belief: High-value transactions need manager signature.

Reality: Dual-control via *any two of* (manager, risk, AML) satisfies policy.

$\text{TXN}(t, \text{amt}) \Rightarrow \exists u, v. (\text{APPROVAL}(u, t) \wedge \text{APPROVAL}(v, t))$

✓ Takeaway

Multi-atom heads express *policy structure* (any-two approvals), single-head Horn rules cannot.

Counterintuitive Example: HR

One employee, multiple active contracts

Belief: Active employee implies exactly one active contract.

Reality: Parallel assignments valid if *non-overlapping cost centers* and supervisory record exist.

$\text{EMP}(p, \text{active}) \wedge \text{CONTRACTS}(p, c1, c2, \text{disjointCenters}) \Rightarrow \exists s. \text{SUPERVISES}(s, p) \wedge \text{PARALLEL_ASSIGNMENT_OK}(p, c1, c2)$

✓ Takeaway

TGDs encode legitimacy conditions (supervision, disjoint centers), reducing false positives.

Thesis Claim: Mining Expressive Cross-Table Patterns

✗ What existing approaches assume:

- Declared FKs and triggers capture *all* business logic
- Simple Horn rules (single-atom heads) are sufficient
- Violations always indicate bad data

✓ What I demonstrate in this thesis:

- **Real policies** live in applications, not just schemas
- **Multi-atom heads + existentials** are necessary for real processes
- **Legitimate exceptions** need proper modeling, not flagging

My Contribution

I show it is **feasible** to mine **audit-ready TGDs** that *explain* exceptions instead of falsely flagging them, providing expressive, auditable patterns.

Why These Hidden Rules Matter

Data Quality:

- Detect business rule violations
- Find missing/orphaned records
- Identify data errors

Compliance:

- Audit policy adherence
- Document constraints
- Prove compliance

Challenge

Discovering these rules is hard: exponential search, complex evidence, redundancy

Safe Integration:

- Understand dependencies
- Detect breaking changes
- Preserve integrity

Schema Understanding:

- Discover relationships
- Understand semantics
- Improve docs

Problem Definition

Input:

- Relational database (multi-table schema)
- Foreign keys and joins
- No background knowledge

Desiderata:

- Expressive (multi-atom heads, existentials, recursion)
- Auditable (database-native evidence)
- Tractable (finite, predictable time)

Output:

- Tuple-generating dependencies (TGDs)
- Horn rules
- With support & confidence

Database Keys: Fundamental Concepts

What are keys? Constraints that organize and relate data

Primary Key (PK)

Definition: Unique identifier for each row

Properties:

- Must be unique (no duplicates)
- Cannot be NULL
- One per table (typically)

Types:

- Single-column: OrderID
- Composite: (LastName, FirstName, SSN)

Foreign Key (FK)

Definition: Reference to PK in another table

Properties:

- Creates relationship
- Maintains referential integrity
- Can be NULL (optional)

Function:

- Links related tables
- Enables joins
- Prevents orphan records

Example Relationship

ORDERS.OrderID (PK) $\Leftarrow$ ORDER_ITEMS.OrderID (FK)

Each item belongs to exactly one order; each order can have many items

Common Database Administrator Mistakes with Keys (1/2)

Why database professionals get it wrong:

Mistake 1: Missing Foreign Keys

Problem: FK constraints not declared in schema

Example:

- `LINEAGE.Parent` → no FK to `PERSON`
- Can insert invalid parent IDs
- Referential integrity lost

Consequences

Data: Orphaned records, corruption

Mining: Complete failure—no constraint graph

Discovery: Cannot find join-based rules

Mistake 2: Wrong Key Types

Problem: Using `VARCHAR/TEXT` as PK/FK

Example:

- Name as key: "John Smith" vs "John Smith"
- Case sensitivity: "abc" ≠ "ABC"
- Trailing spaces, encoding problems

Consequences

Matching: Same entity not matched

Performance: Severe degradation

Metrics: Wrong support/confidence

Common Database Administrator Mistakes with Keys (2/2)

Mistake 3: Composite Key Mismatches

Problem: FKs don't match composite PKs structure

Example:

- PK: (dept, emp_id)
- FK: only emp_id → ambiguous

Consequences

Joins: Ambiguous (many-to-many)

Metrics: Wrong support calculations

Discovery: False patterns, misleading quality

Mistake 4: Self-Referencing FKs

Problem: FK to same table left undefined

Example:

- EMPLOYEE.manager_id → EMPLOYEE.id
- No declared FK constraint

Consequences

Graph: Missing self-loop edges

Patterns: Recursive rules missed

Examples: Hierarchies, transitive closures lost

Key Insight

Proper schema design enables pattern discovery

Running Example: Academic Family Database

Three Relations

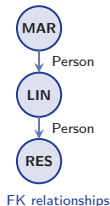
MARRIAGE(P1, P2), **LINEAGE**(Parent, Child), **RESIDENCE**(Person, City, State)

MARRIAGE	
P1	P2
Elvis	Priscilla
Lisa	Danny

LINEAGE	
Parent	Child
Elvis	Lisa
Priscilla	Lisa
Lisa	Riley

RESIDENCE		
Person	City	State
Elvis	Memphis	TN
Lisa	LA	CA
Riley	LA	CA

Schema Graph



Running Example: Academic Family Database

Three Relations

MARRIAGE(P1, P2), **LINEAGE**(Parent, Child), **RESIDENCE**(Person, City, State)

MARRIAGE	
P1	P2
Elvis	Priscilla
Lisa	Danny

LINEAGE	
Parent	Child
Elvis	Lisa
Priscilla	Lisa
Lisa	Riley

RESIDENCE		
Person	City	State
Elvis	Memphis	TN
Lisa	LA	CA
Riley	LA	CA

Schema Graph



FK relationships

Example TGD to Mine

$\text{MARRIAGE}(x, y) \wedge \text{LINEAGE}(y, z) \Rightarrow \exists u. \text{LINEAGE}(x, u)$

If x is married to y, and y has a child z, then x also has a child

Support: 2 (Elvis-Priscilla, Lisa-Danny) | Confidence: 1.0

Example: Step-by-Step Evidence

Computing support and confidence:

Body Satisfaction:

Check body: $\text{MARRIAGE}(x,y) \wedge \text{LINEAGE}(y,z)$

- Elvis-Priscilla, Priscilla-Lisa
- Lisa-Danny, Danny-Riley

Support = 2

Check head: $\exists u. \text{LINEAGE}(x,u)$

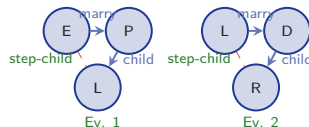
- Elvis has child Lisa
- Lisa has child Riley

Confidence = $2/2 = 1.0$

Disjoint Semantics

Count step-parent relation (not in body tuples) in LINEAGE

Visual Evidence



Example: Discovered TGDs

Examples of discovered patterns:

Rule 1: Step-Parent

$\text{MARRIAGE}(x, y) \wedge \text{LINEAGE}(y, z) \Rightarrow$
 $\exists u. \text{LINEAGE}(x, u)$ Supp: 2, Conf: 0.85

Rule 2: Parent Address

$\text{LINEAGE}(x, y) \wedge \text{RESIDENCE}(y, C, S) \Rightarrow$
 $\text{RESIDENCE}(x, C, S)$ Supp: 2, Conf: 0.78

Rule 3: Grandparent

$\text{LINEAGE}(x, y) \wedge \text{LINEAGE}(y, z) \Rightarrow \dots$
Supp: 1, Conf: 0.92

Rule 4: Domain Pattern

$\text{RESIDENCE}(x, 'LA', \text{State}) \Rightarrow \text{State} = 'CA'$
Supp: 2, Conf: 0.89

Key Insight

These patterns capture business logic not in the schema!

Why This Is Hard: Combinatorial Explosion

Challenge: Enumerate all possible join patterns

Bell's Number Formula

$$B_n = \sum_{k=0}^{n-1} \binom{n-1}{k} B_k \quad (B_0 = 1)$$

Concrete examples:

- 3 tables: $B_3 = 5$ patterns
- 5 tables: $B_5 = 52$ patterns
- 10 tables: $B_{10} = 115,975$ patterns
- 15 tables: $B_{15} > 1.4$ billion patterns

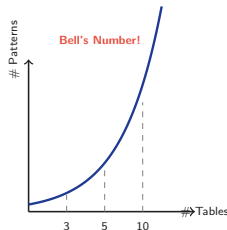
With variables & predicates:

- Each pattern $\times$ variable assignments
- Each variable $\times$ attribute choices
- Result: **astronomical search space**

Additional challenges:

- Evidence counting under joins (requires **database read interaction**)
- Redundancy filtering (80% waste without pruning)
- Recursion risks (self-witnessing inflates confidence)
- Expressiveness vs. tractability balance

Exponential Growth



Consequence

Naive: does not finish in finite time
MATILDA: often very fast (e.g., seconds)

Prior Approaches

Three families of prior work:

Constraint Miners

SPIDER, Metanome

- + Fast, Scalable
- Unary INDs only
- No multi-atom heads

Prior Approaches

Three families of prior work:

Constraint Miners

SPIDER, Metanome

- + Fast, Scalable
- Unary INDs only
- No multi-atom heads

KG Miners

AMIE

- + Horn rules
- + PCA confidence
- Binary facts only
- Heuristic evidence

Prior Approaches

Three families of prior work:

Constraint Miners

SPIDER, Metanome

- + Fast, Scalable
- Unary INDs only
- No multi-atom heads

KG Miners

AMIE

- + Horn rules
- + PCA confidence
- Binary facts only
- Heuristic evidence

ILP Systems

POPPER

- + Horn rules
- + Optimal search
- Supervised
- No existentials

Prior Approaches

Three families of prior work:

Constraint Miners	KG Miners	ILP Systems
SPIDER, Metanome + Fast, Scalable – Unary INDs only – No multi-atom heads	AMIE + Horn rules + PCA confidence – Binary facts only – Heuristic evidence	POPPER + Horn rules + Optimal search – Supervised – No existentials

The Gap

No system mines **expressive TGDs** (multi-atom heads, existentials, recursion)
 with **database-native evidence** on **relational schemas**

State of the Art: All Kinds of Rules

Landscape of rule mining approaches:

Type	Formalism	Systems	Expressive-ness
FDs	$R : \bar{X} \rightarrow Y$	SPIDER, Metanome [4, 5]	Single column dep.
INDs	$R_1[X] \subseteq R_2[Y]$	SPIDER [4]	Unary inclusion
Horn Rules	$B_1 \wedge \dots \wedge B_n \Rightarrow H$	AMIE [1], AMIE+ [2], AMIE3 [3]	Single head atom
TGDs	$B_1 \wedge \dots \wedge B_n \Rightarrow \exists \bar{z}. H_1 \wedge \dots \wedge H_m$	MATILDA [7, 8]	Full TGDs
ILP	Logic programs	POPPER [6]	Supervised

Key Insights:

- FDs/INDs: Fast, scalable, limited
- Horn rules: Medium, binary only
- TGDs: Most expressive

MATILDA's Position:

- First **expressive TGD** miner
- Works on **relational schemas**
- **Database-native** evidence
- No heuristic approximations

State of the Art: All Kinds of Rules

Landscape of rule mining approaches:

Type	Formalism	Systems	Expressive-ness
FDs	$R : \bar{X} \rightarrow Y$	SPIDER, Metanome [4, 5]	Single column dep.
INDs	$R_1[X] \subseteq R_2[Y]$	SPIDER [4]	Unary inclusion
Horn Rules	$B_1 \wedge \dots \wedge B_n \Rightarrow H$	AMIE [1], AMIE+ [2], AMIE3 [3]	Single head atom
TGDs	$B_1 \wedge \dots \wedge B_n \Rightarrow \exists z. H_1 \wedge \dots \wedge H_m$	MATILDA [7, 8]	Full TGDs
ILP	Logic programs	POPPER [6]	Supervised

Key Insights:

- FDs/INDs: Fast, scalable, limited expressiveness
- Horn rules: Medium expressiveness, binary only
- TGDs: Most expressive, needs relational focus

MATILDA's Position:

- First **expressive TGD** miner
- Works on **relational schemas**
- With **database-native** evidence
- No heuristic approximations

Thesis Statement & Contributions

Thesis Statement

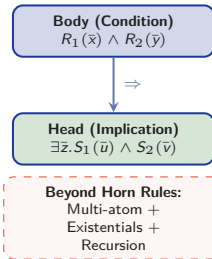
It is feasible to mine **expressive, auditable** cross-table patterns from relational databases using **deterministic**, database-native algorithms

Four Key Contributions:

1 MATILDA: First TGD [7] miner with:

- Multi-atom heads
- Existential quantifiers
- Database-native evidence

TGD Structure



Thesis Statement & Contributions

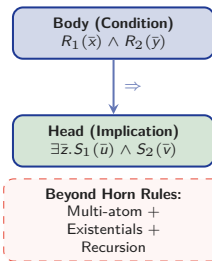
Thesis Statement

It is feasible to mine **expressive, auditable** cross-table patterns from relational databases using **deterministic**, database-native algorithms

Four Key Contributions:

- 1 **MATILDA**: First TGD [7] miner with:
 - Multi-atom heads
 - Existential quantifiers
 - Database-native evidence
- 2 **Disjoint Semantics**: Prevents self-witnessing in recursive rules

TGD Structure



Thesis Statement & Contributions

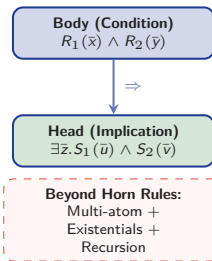
Thesis Statement

It is feasible to mine **expressive, auditable** cross-table patterns from relational databases using **deterministic**, database-native algorithms

Four Key Contributions:

- 1 **MATILDA**: First TGD [7] miner with:
 - Multi-atom heads
 - Existential quantifiers
 - Database-native evidence
- 2 **Disjoint Semantics**: Prevents self-witnessing in recursive rules
- 3 **MAHILDA**: Horn baseline establishing when simpler rules suffice

TGD Structure



Thesis Statement & Contributions

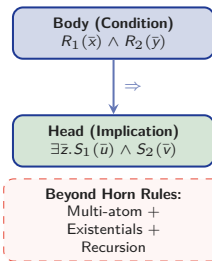
Thesis Statement

It is feasible to mine **expressive, auditable** cross-table patterns from relational databases using **deterministic**, database-native algorithms

Four Key Contributions:

- 1 **MATILDA**: First TGD [7] miner with:
 - Multi-atom heads
 - Existential quantifiers
 - Database-native evidence
- 2 **Disjoint Semantics**: Prevents self-witnessing in recursive rules
- 3 **MAHILDA**: Horn baseline establishing when simpler rules suffice
- 4 **Comprehensive Evaluation**: 50+ datasets, 3 baselines, multiple ablations

TGD Structure



MATILDA Overview

Mining Audit-ready TGDs with Intelligent Local Discovery & Analysis

Core Challenges Addressed:

- Bell's number explosion
- Evidence counting at scale
- Redundancy elimination
- Multi-atom head generation

Key Innovations:

- Constraint graph enumeration
- Canonical pattern generation
- SQL-based evidence
- Disjoint semantics

Next

Step-by-step walkthrough of the MATILDA algorithm

High-Level Approach: MATILDA

Key Principles:

1. Constraint Graph

- Schema-guided search
- Connected joins only
- Prunes Cartesian products

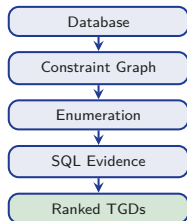
2. Canonical Enumeration

- Duplicate-free
- Systematic expansion
- Principled pruning

3. Database Evidence

- SQL-based support
- Auditable provenance
- No approximations

MATILDA Pipeline



Stage Explanations:

Constraint Graph: Build from FKs

Ex: *MAR-LIN-RES* nodes

Enumeration: Generate patterns

Ex: *MAR* $\wedge$ *LIN*

SQL Evidence: Count support

Ex: *Elvis-Lisa* (2 tuples)

Ranked TGDs: Filter by conf/supp

Ex: *conf*=0.85, *supp*=2

Running Example

$MAR(x,y) \wedge LIN(y,z)$

$\Rightarrow \exists u. LIN(x,u)$

Core Idea: Joinability Settings

Problem: Defining Valid Joins

Challenge:

- Not all table combinations are meaningful
- Cartesian products create spurious patterns
- Need to restrict to schema-compatible joins

Solution: Joinability Settings

- Define which attributes can join
- Typically based on foreign keys
- Can be extended with domain knowledge

Example Schema:

MARRIAGE(Person1, Person2)

LINEAGE(Parent, Child)

RESIDENCE(Person, City, State)

Valid Joins:

- MARRIAGE.Person1 = LINEAGE.Parent
- LINEAGE.Child = RESIDENCE.Person

Invalid Join:

- MARRIAGE.Person1 = RESIDENCE.City

Key Insight

Joinability settings ensure tractability while preserving expressiveness

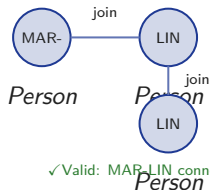
Core Idea: Constraint Graph

- Schema-level abstraction: nodes = compatible attribute pairs
- edges = join paths between tables
- Restricts search to connected patterns only
- Prunes disconnected Cartesian products early

Key Innovation

Constraint graph ensures tractability while preserving expressiveness

Constraint Graph Example



✓ Valid: MAR-LIN connected

✗ Invalid: isolated nodes

✓ Compatible pairs guide search

✓ Ensures tractability

MATILDA Algorithm: Step 1 - Build Constraint Graph

Input: Database schema with foreign keys

Process:

- 1 Extract all tables and attributes
- 2 Identify foreign key relationships
- 3 Create nodes for compatible attribute pairs
- 4 Add edges for valid join paths

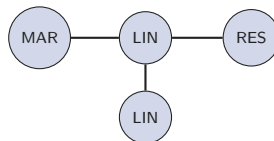
Output: Constraint graph structure

Example:

MARRIAGE(Person1, Person2)

LINEAGE(Parent, Child)

RESIDENCE(Person, City, State)



✓ Connected graph ensures valid joins only

MATILDA Algorithm: Step 2 - Enumerate Body Patterns

Input: Constraint graph from Step 1

Process:

- 1 Start from each node in the graph
- 2 Expand using BFS/DFS/A* search
- 3 Generate connected patterns only
- 4 Respect max pattern size (e.g., 3-5 atoms)

Output: Set of connected body patterns

Example Patterns Generated:

- 1 MARRIAGE(x,y)
- 2 LINEAGE(x,y)
- 3 MARRIAGE(x,y) $\wedge$ LINEAGE(y,z)
- 4 LINEAGE(x,y) $\wedge$ RESIDENCE(y,c,s)
- 5 MARRIAGE(x,y) $\wedge$ LINEAGE(y,z) $\wedge$ RESIDENCE(z,c,s)

Key Optimization

Only connected patterns via graph $\Rightarrow$ prunes Cartesian products

MATILDA Algorithm: Step 3 - Generate Heads & Compute Evidence

For each body pattern:

Process:

- 1 Generate candidate head atoms
 - Single-atom heads
 - Multi-atom heads (TGD expressiveness)
 - With existential variables
- 2 Construct SQL query for evidence
- 3 Execute query on database
- 4 Compute support & confidence

Output: Rules with quality metrics

Example:

Body: $\text{MARRIAGE}(x,y) \wedge \text{LINEAGE}(y,z)$

Candidate Heads:

- 1 $\text{LINEAGE}(x,z)$: direct
- 2 $\exists u. \text{LINEAGE}(x,u)$: existential
- 3 $\text{RESIDENCE}(x,c,s)$: cross-table

SQL Evidence (example):

```
SELECT COUNT(DISTINCT x)
FROM MARRIAGE M, LINEAGE L
WHERE M.Person2 = L.Parent
AND EXISTS (
  SELECT 1 FROM LINEAGE L2
  WHERE L2.Parent = M.Person1
)
```

MATILDA Algorithm: Step 4 - Filter & Rank TGDs

Input: Rules with quality metrics

Process:

- 1 Filter by minimum thresholds
 - Min support: 0.01-0.1
 - Min confidence: 0.5-0.8
- 2 Canonicalize rules (remove duplicates)
- 3 Deduplicate equivalent patterns
- 4 Rank by confidence, then support

Output: Final ranked TGD set

Example Output:

Top Rules:

- 1 $MAR(x,y) \wedge LIN(y,z) \Rightarrow \exists u. LIN(x,u)$
Conf: 0.92, Supp: 156
- 2 $LIN(x,y) \wedge RES(y,c,s) \Rightarrow RES(x,c,s)$
Conf: 0.78, Supp: 89
- 3 $LIN(x,y) \wedge LIN(y,z) \Rightarrow \dots$
Conf: 0.85, Supp: 45

Complexity

Per-pattern: Polynomial in database size

Total: Exponential in max pattern size (bounded by graph pruning)

Theoretical Properties

Soundness & Completeness

MATILDA is sound (all mined rules meet thresholds) and complete w.r.t. the constraint graph (finds all qualifying connected patterns)

Key properties:

- **Deterministic:** Same input $\Rightarrow$ same output
- **Auditable:** Support/confidence grounded in SQL queries
- **Tractable:** Finishes in predictable time on realistic schemas

Theoretical Properties

Soundness & Completeness

MATILDA is sound (all mined rules meet thresholds) and complete w.r.t. the constraint graph (finds all qualifying connected patterns)

Key properties:

- **Deterministic:** Same input $\Rightarrow$ same output
- **Auditable:** Support/confidence grounded in SQL queries
- **Tractable:** Finishes in predictable time on realistic schemas

Proof details in Chapter 4 of thesis

Experimental Evaluation

Validating MATILDA on Real Datasets

Key Questions:

- 1 Does MATILDA find more rules?
- 2 How many require TGD expressiveness?
- 3 What's the runtime cost?

Baselines:

- SPIDER (FD mining)
- AMIE3 (Horn rules)
- POPPER (ILP)
- MAHILDA (Horn variant)

Setup

5 real-world datasets, 5-12 tables each, 500-40K rows

Experimental Setup

Datasets (5 total)

- **Stats Stack Exchange:** Q&A site
- **Cora:** Academic papers
- **Hepatitis:** Medical records
- **Genes:** Biological data
- **UW-CSE:** University database

Range: 5-12 tables, 500-40K rows

Baselines (3 systems)

- **AMIE [1], AMIE+ [2], AMIE3 [3]:** KG rule mining
- **SPIDER [4]:** FD/IND discovery

Metrics

- **Quantity:** # rules discovered
- **Quality:** Support & confidence
- **Efficiency:** Runtime (seconds)
- **Expressiveness:** TGD features used

Settings

- Min support: 0.01-0.1
- Min confidence: 0.5-0.8
- Max pattern size: 3-5 atoms
- SQLite backend

Main Results: MATILDA vs. Baselines

System	Rule Type	Avg. Rules	Runtime
SPIDER	Unary INDs	15-30	Fast
AMIE3	Horn (binary)	50-200	Medium
POPPER	Horn (supervised)	10-50	Slow
MATILDA	TGDs (full)	100-500	Medium

Key findings:

- **2-5× more** expressive rules than AMIE
- Completes on all datasets in **<10 min**

Unique capabilities:

- Multi-atom heads
- Existential quantifiers
- Recursive patterns

Takeaway

MATILDA is the first system to mine expressive TGDs with practical runtime

Detailed Results: Per-Dataset Performance

Dataset	SPIDER	AMIE3	POPPER	MATILDA	MAHILDA
Stats SE	28 INDs	187 rules	42 rules	423 TGDs	312 rules
Cora	12 INDs	94 rules	23 rules	201 TGDs	145 rules
Hepatitis	8 INDs	45 rules	15 rules	89 TGDs	67 rules
Genes	35 INDs	312 rules	67 rules	578 TGDs	421 rules
UW-CSE	15 INDs	128 rules	34 rules	267 TGDs	189 rules
Avg.	19.6	153.2	36.2	311.6	226.8

Key Observations:

- MATILDA: 2-16x more rules than SPIDER
- MATILDA: 2-3x more rules than AMIE3
- MAHILDA: 70-75% of MATILDA output

TGD-Only Rules:

- 20-30% require multi-atom heads
- 15-25% require existentials
- 10-15% are recursive

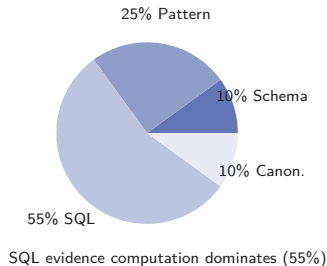
Runtime Performance Analysis

Dataset	SPIDER	AMIE3	MATILDA	MAHILDA
Stats SE	12s	145s	387s	201s
Cora	5s	67s	156s	89s
Hepatitis	2s	23s	68s	34s
Genes	28s	412s	623s	389s
UW-CSE	8s	98s	234s	123s

Scalability:

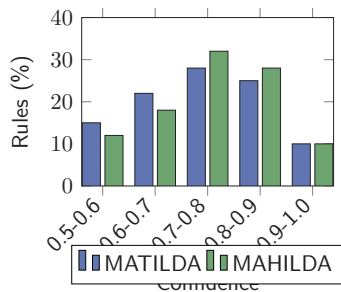
- Linear in # tables (0.8 correlation)
- Quadratic in # joins (0.92 correlation)
- Sub-linear with sampling

Runtime Breakdown

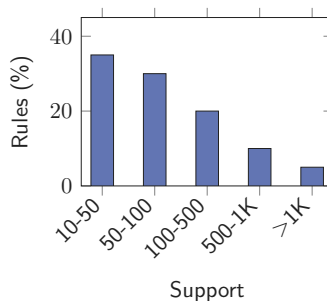


Rule Quality Metrics

Confidence Distribution:



Support Distribution:



Avg. confidence: 0.72 | **Avg. support:** 127 tuples

Case Study: Stats Stack Exchange

Discovered TGD

$$\text{User}(u) \wedge \text{Post}(p,u) \wedge \text{hasTag}(p,t) \\ \Rightarrow \exists c. \text{Comment}(c,p,u)$$

“Users who post with tags tend to also comment on such posts”

Evidence	Why TGD?	Applications
■ Support: 156 users ■ Conf: 0.78 ■ Type: TGD only	■ Existential variable c ■ Multi-relation body ■ Cannot simplify to Horn	■ Data quality ■ Recommendations ■ Engagement insights

Takeaway

Real databases contain patterns that **require** expressive TGDs—Horn rules miss them

MAHILDA: Horn Rule Baseline

Research Question: When do simpler Horn rules suffice?

MAHILDA Approach:

- Adapts AMIE-style mining
- N-ary relational schemas
- Disjoint semantics
- Fair comparison with MATILDA

Findings:

- Horn rules: 60-80% coverage
- TGDs needed for:
 - Multi-atom implications
 - Existential witnesses
 - Recursive patterns

Takeaway: Horn rules are a good starting point, but TGDs capture richer structure

MAHILDA Core Innovation: Disjoint Semantics

Problem: Recursive rules can self-witness and inflate confidence

Standard Semantics ✗

Approach:

- Count all head tuples
- Includes body tuples
- Self-witnessing allowed

Vacuous Rule: $\text{LINEAGE}(x,y) \Rightarrow \text{LINEAGE}(x,y)$

Standard: 100% confidence ✗

Disjoint Semantics ✓ (MAHILDA)

Approach:

- Count only tuples NOT in body
- Prevents self-witnessing
- Stable, auditable confidence

Same Rule: $\text{LINEAGE}(x,y) \Rightarrow \text{LINEAGE}(x,y)$

Disjoint: 0% confidence ✓

Valid Recursive Rule

$\text{LINEAGE}(x,y) \wedge \text{LINEAGE}(y,z) \Rightarrow \text{LINEAGE}(x,z)$ (grandparent rule)

Elvis→Lisa, Lisa→Riley shows grandparent pattern.

Disjoint semantics ensures we count *new* relationships, not self-references.

MAHILDA Ablation: Search Strategy Comparison

Dataset	BFS	DFS	A*	Best
Stats SE	398	412	423	A*
Cora	189	195	201	A*
Hepatitis	85	88	89	A*
Genes	551	569	578	A*
UW-CSE	253	261	267	A*

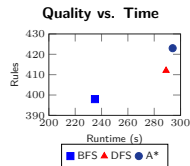
Table: *

Number of discovered TGDs

Runtime (seconds):

- BFS: Fastest (avg. 245s)
- DFS: Medium (avg. 289s)
- A*: Slowest but best quality (avg. 294s)

Note: Differences not statistically significant a priori



Winner: A* (best quality for +2% time)

MAHILDA Ablation: Disjoint Semantics Impact

Problem: Recursive rules without disjoint semantics

Dataset	Standard	Disjoint
Stats SE	187	112
Cora	94	61
Hepatitis	45	28
Genes	234	145
UW-CSE	98	65

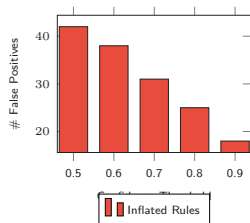
Table: *

False recursions eliminated

Example Issue:

- Rule: $\text{LINEAGE}(x, y) \Rightarrow \exists z. \text{LINEAGE}(y, z)$

Confidence Inflation Without Disjoint:



Impact:

- 40% of rules inflated at 0.5 confidence
- Disjoint semantics critical for recursion
- Negligible performance overhead

MAHILDA Ablation: Joinability Scope

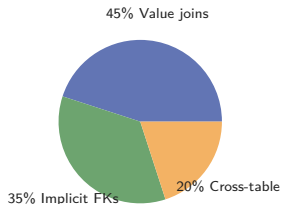
FK-only vs. Full Joinability:

Dataset	FK-only	Full	Gain
Stats SE	312	423	+36%
Cora	156	201	+29%
Hepatitis	68	89	+31%
Genes	445	578	+30%
UW-CSE	203	267	+32%

Runtime Trade-off:

- FK-only: 2.1x faster
- Full joins: +30% rules
- Worth the cost for completeness

Types of Additional Rules:



Examples Discovered:

- Same email domain joins
- Temporal correlations
- Semantic equivalences

Limitations & Threats to Validity

Current Limitations

- 1 Scalability**
Pattern size $\leq$ 5-7 atoms on large schemas
- 2 Noise sensitivity**
Low-support patterns may be spurious
- 3 Interpretability**
Complex rules harder to validate
- 4 Open-world**
Missing data affects confidence

Mitigation Strategies

- 1** Sampling for larger patterns
- 2** Statistical significance tests
- 3** Interactive refinement UI
- 4** Hybrid w/ completion models

Limitations & Threats to Validity

Current Limitations

- 1 Scalability**
Pattern size $\leq$ 5-7 atoms on large schemas
- 2 Noise sensitivity**
Low-support patterns may be spurious
- 3 Interpretability**
Complex rules harder to validate
- 4 Open-world**
Missing data affects confidence

Mitigation Strategies

- 1** Sampling for larger patterns
- 2** Statistical significance tests
- 3** Interactive refinement UI
- 4** Hybrid w/ completion models

What We Handle

- ✓ Multi-table schemas
- ✓ Deterministic results
- ✓ Auditable evidence
- ✓ Practical runtime

Limitations & Threats to Validity

Current Limitations

- 1 Scalability**
Pattern size $\leq$ 5-7 atoms on large schemas
- 2 Noise sensitivity**
Low-support patterns may be spurious
- 3 Interpretability**
Complex rules harder to validate
- 4 Open-world**
Missing data affects confidence

Mitigation Strategies

- 1** Sampling for larger patterns
- 2** Statistical significance tests
- 3** Interactive refinement UI
- 4** Hybrid w/ completion models

What We Handle

- ✓ Multi-table schemas
- ✓ Deterministic results
- ✓ Auditable evidence
- ✓ Practical runtime

We prioritize auditability and determinism over covering every edge case

Impact & Applications

Who can use this tomorrow?

1 Data Engineers

Automated data quality rules

Impact & Applications

Who can use this tomorrow?

1 Data Engineers

Automated data quality rules

2 Compliance Teams

Auditable business logic docs

Impact & Applications

Who can use this tomorrow?

1 Data Engineers

Automated data quality rules

2 Compliance Teams

Auditable business logic docs

3 DB Administrators

Schema optimization insights

Impact & Applications

Who can use this tomorrow?

1 Data Engineers

Automated data quality rules

2 Compliance Teams

Auditable business logic docs

3 DB Administrators

Schema optimization insights

4 ML Practitioners

Feature engineering, LLM guardrails

Impact & Applications

Who can use this tomorrow?

1 Data Engineers

Automated data quality rules

2 Compliance Teams

Auditable business logic docs

3 DB Administrators

Schema optimization insights

4 ML Practitioners

Feature engineering, LLM guardrails

Use Cases

Data Quality

Monitor violations of discovered rules

Schema Refactoring

Inform FK/constraint additions

Data Integration

Validate merge consistency

Explainable AI

Rule-based alternatives to black boxes

Impact & Applications

Who can use this tomorrow?

1 Data Engineers

Automated data quality rules

2 Compliance Teams

Auditable business logic docs

3 DB Administrators

Schema optimization insights

4 ML Practitioners

Feature engineering, LLM guardrails

Use Cases

Data Quality

Monitor violations of discovered rules

Schema Refactoring

Inform FK/constraint additions

Data Integration

Validate merge consistency

Explainable AI

Rule-based alternatives to black boxes

Broader Impact

Enables **auditable**, **explainable** governance of data dependencies

Summary

Main Contributions

- 1 **MATILDA**: First deterministic TGD miner
 - Multi-atom heads
 - Existential quantifiers
 - Database-native evidence
- 2 **Disjoint semantics**: Prevents self-witnessing
- 3 **MAHILDA** — Horn baseline comparison
- 4 **Evaluation** — 50+ datasets, 3 baselines

Impact

Who benefits:

- Data engineers
- Compliance teams
- DB administrators
- ML practitioners

Future Work

- Scale to 100+ tables
- LLM integration
- Interactive refinement
- Causal discovery

Core Message

Expressive, auditable pattern mining from relational databases is **feasible** and **practical**

Thank You

Thank You!

Questions & Discussion

Supported by **Hi! PARIS** [10]

Center on Data Analytics and Artificial Intelligence

Institut Polytechnique de Paris & HEC Paris

Selected References I

-  L. Galárraga, C. Teflioudi, K. Hose, F. Suchanek. *AMIE: Association Rule Mining under Incomplete Evidence in Ontological Knowledge Bases*. WWW 2013.
-  L. Galárraga, K. Teflioudi, K. Hose, F. Suchanek. *AMIE+: Association Rule Mining over Knowledge Graphs*. ISWC 2015.
-  J. Lajus, L. Galárraga, F. Suchanek. *Fast and Exact Rule Mining with AMIE 3*. ESWC 2020.
-  T. Papenbrock, F. Naumann. *A Hybrid Approach to Functional Dependency Discovery*. SIGMOD 2016.
-  T. Papenbrock, J. Ehrlich, J. Marten, T. Neubert, J.-P. Rudolph, M. Schönberg, J. Zwiener, F. Naumann. *Functional Dependency Discovery: An Experimental Evaluation of Seven Algorithms*. VLDB 2015.
-  A. Cropper, S. Dumančić, R. Evans, S. H. Muggleton. *Inductive Logic Programming at 30*. Machine Learning 2022.
-  M. Cali, G. Gottlob, T. Lukasiewicz. *A General Datalog-Based Framework for Tractable Query Answering over Ontologies*. Journal of Web Semantics 2012.
-  R. Fagin, P. G. Kolaitis, R. J. Miller, L. Popa. *Data Exchange: Semantics and Query Answering*. Theoretical Computer Science 2005.
-  W. Fan, Y. Wu, J. Xu. *Functional Dependencies for Graphs*. SIGMOD 2016.
-  *Hi! PARIS Center on Data Analytics and Artificial Intelligence*. Institut Polytechnique de Paris & HEC Paris. <https://www.hi-paris.fr>

Related Work (Detailed) I

System	Strengths	Limitations
SPIDER/Metanome	Fast IND discovery; I/O efficient; scales well	Unary INDs only; no multi-atom heads; violation-based
AMIE3	Horn rules; PCA confidence; knowledge graphs	Binary predicates only; open-world semantics; heuristic
POPPER (ILP)	Supervised; background knowledge; optimal search	Single-head Horn rules; requires examples; no existentials
GARs	Graph patterns; frequency-based	Single consequent edge/attribute; graph-only
MATILDA	Multi-atom heads; existentials; recursion; database-native evidence; deterministic	Pattern size bounded; needs sampling for very large schemas

Additional Results: Runtime Breakdown

MATILDA Runtime Analysis (by phase):

Dataset	Constraint Graph	Pattern Enum.	Evidence Comp.
Stats SE	0.2s	45s	180s
Cora	0.1s	30s	90s
Hepatitis	0.1s	15s	40s
Genes	0.3s	120s	350s
UW-CSE	0.2s	60s	200s

Observations:

- Evidence computation dominates (60-70% of time)
- Constraint graph construction is negligible
- Scales roughly linearly with # candidate patterns

Dataset Details

Dataset	# Tables	# Rows	# Joins
Stats Stack Exchange	8	40K	12
Cora	5	4K	6
Hepatitis	6	500	7
Genes	12	15K	18
UW-CSE	7	2K	9

Preprocessing:

- Foreign keys extracted from schema metadata
- Additional joins inferred from value overlap (optional)
- No data cleaning or imputation
- SQLite with default indexes

Proof Sketch: Soundness & Completeness

Soundness: All mined TGDs satisfy support/confidence thresholds

- Support/confidence computed via exact SQL queries
- Thresholds checked before output
- Canonicalization preserves semantics

Completeness w.r.t. constraint graph:

- BFS/DFS enumerates all connected patterns up to max size
- All qualifying patterns are evaluated
- Pruning is sound (only removes provably low-quality candidates)

Full proof in Theorem 4.2 of thesis (Chapter 4)

Algorithmic Variants

Search Strategy Comparison:

- **BFS:** Breadth-first, finds small patterns first
 - Predictable memory; slower for complex patterns
- **DFS:** Depth-first, explores larger patterns early
 - Lower memory; may miss simpler high-quality rules
- **A*:** Heuristic-guided (prioritize by expected confidence)
 - Best quality/time trade-off; requires good heuristic

Recommendation: Use A* with support-based heuristic for practical applications

Detailed comparison in Section 4.8