DELFT UNIVERSITY OF TECHNOLOGY

MASTERS THESIS

Reverse Engineering Relational Data for Entity Type Recognition in Enterprise Solutions at ING

Author:
A.R. BREURKES

Supervisor:
Dr. C. LOFI

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Web Information Systems Group
Software Technology

Student number: 4398033

Thesis committee:

Prof. dr. A. van Deursen,	*TU Delft, Chair*
Dr. C. Lofi,	*TU Delft, Supervisor*
Dr. A. Katsifodimos,	*TU Delft*
Drs. H.A.J. Brons,	*ING*

An electronic version of this thesis is available at
https://repository.tudelft.nl/.

May 20, 2021

DELFT UNIVERSITY OF TECHNOLOGY

Abstract

Electrical Engineering, Mathematics and Computer Science
Software Technology

Master of Science

Reverse Engineering Relational Data for Entity Type Recognition in Enterprise Solutions at ING

By A.R. Breurkes

Database *entity type recognition* is the practice of recognizing conceptual *entity types* for which given data sets contain data. In big data or data lake settings, it is not always known which conceptual entity types are represented in each data set, making it difficult to extract value from the data. Depending on the logical schemas, each conceptual entity type can also be represented in the data instances in multiple different ways. This phenomenon, called *semantic heterogeneity*, poses a challenge when attempting database entity type recognition. Narrowing down the problem space to a specific organization makes it easier to cope with such problems. Organizations know which entity types are used in the organization and require only those to be recognized. And while there is heterogeneity in representation, there is likely a common set of rules each logical schema adheres to which can be exploited to recognize semantic heterogeneity. Furthermore, experts at an organization can provide example data instances for each conceptual entity type of interest, which provide ground truth for the proposed database entity type recognition solution. The proposed solution makes *data profiles* of the example data instances, and then attempts to recognize entity types in previously unseen data instances using a rule-based approach. Rules are used to maximize the ease of explainability of results, as is often desired at a bank, and can easily be added to or removed from the solution to maximize adaptability. Experiments using the proposed solution show promising results, with up to 90 percent of entity types correctly recognized over a total of 170,000 entities.

Acknowledgements

I would like to thank my academic supervisor, Christoph Lofi, for continuously helping me to raise the bar high, having numerous discussions with me on (un)related matter, and for his mentorship throughout my time as his mentee. I am sure that without his array of persistent guidance, I would not have learned as much as I have done now.

Secondly, I would like to thank my industry supervisor, Jerry Brons, for his undivided interest, attention and help. Jerry was virtually always available for discussion and to open doors to resources at ING, but also eagerly provided insightful articles on progress in our field of research.

I would also like to thank Georgios Siachamis for always being available for quick questions, proof reading and discussion. It was nice to have someone to discuss research with without having to be formal.

Finally, I want to thank the thesis committee for their time and energy to evaluate my work, and I want to thank my friends and family for their unconditional support.

Contents

Abstract iii

Acknowledgements v

1 Introduction 1
1.1 Contributions . . . 2
1.2 Thesis Outline . . . 2

2 Problem Description 3
2.1 The origin of the problem . . . 3
2.1.1 The rules leading to special cases . . . 5
1. The table contains data on multiple subtypes from some entity type in its taxonomy. . . . 5
2. The table contains data on a type at the bottom of its taxonomy, and only that type. . . . 6
3. The rows in the table contain data on a combination of multiple entity types, or part thereof, possibly from different taxonomies . . . 6
2.2 The problem . . . 7
2.2.1 Problem one (P1) . . . 8
2.2.2 Problem two (P2) . . . 8
2.3 Formal Problem Definition . . . 9

3 Research Methodology 11
3.1 Decision making process . . . 11
3.2 The research environment . . . 11

4 Background 13
4.1 Similar research . . . 13
4.1.1 Reverse engineering . . . 13
Andersson: reverse engineering to ECR+ . . . 14
Chiang et al.: reverse engineering to EER . . . 14
Malpani et al.: reverse engineering to EDM . . . 15
4.1.2 Entity type recognition . . . 16
Sleeman et al.: fine-grained entity type recognition in knowledge bases . . . 17
Giunchiglia et al.: semantic heterogeneity detection for knowledge bases . . . 17
4.2 Supportive research . . . 18
4.2.1 Cardinalities . . . 18
4.2.2 Patterns and data types . . . 20
4.2.3 Semantic domain classification . . . 20
4.2.4 Inclusion dependencies . . . 21

4.3 Conclusion . . . 21

5 Proposed Solution **23**
5.1 The solution in general . . . 23
5.2 Clustering & data profiling . . . 24
5.2.1 Clustering methods for unlabeled example data . . . 25
1. The table contains data on multiple subtypes from some entity type in its taxonomy. . . . 25
2. The table contains data on a type at the bottom of its taxonomy, and only that type. . . . 26
3. The rows in the table contain data on a combination of multiple entity types, or part thereof, possibly from different taxonomies. . . . 27
5.2.2 Specification of the data profiles . . . 27
5.3 Entity type recognition . . . 29
5.3.1 Rule-based classification . . . 29
5.3.2 Output & interpretation . . . 31
5.4 Conclusion . . . 32

6 Results & Evaluation **35**
6.1 Experimental setup . . . 35
6.1.1 The data generator . . . 36
6.2 Clustering results . . . 37
6.2.1 All entity (sub)types from the same taxonomy in the same table 37
6.2.2 One entity type represented in a table . . . 38
6.2.3 Discrepancies with real data . . . 38
6.3 Classification experiments & results . . . 39
6.3.1 Experiment 1: relatively little diversity . . . 40
6.3.2 Experiment 2: many entity types with relatively little diversity 41
6.3.3 Experiment 3: many entity types with relatively high diversity 42
6.3.4 Experiment 4: little entity types with very little diversity . . . 43
6.3.5 Experiment 5: evaluation on real data . . . 43
6.4 Conclusion & Discussion . . . 45

7 Conclusion **47**
7.1 Future Work . . . 48

Bibliography **49**

List of Figures

2.1 An example entity type taxonomy, truncated to compact cars and SUVs. 4

4.1 The database reverse engineering process vs. the database entity type recognition process. 14
4.2 A classification of data profiling tasks. The leaves represent profiles, while the higher levels represent categories. Profiles are not limited to their category, but are grouped based on (Abedjan et al., 2015, Fig. 1) and (Naumann, 2014, Figure 1). 19

5.1 The method flowchart for the proposed solution. 24

6.1 The result of clustering a data instance that adheres to rule one defined in Table 2.3. 37
6.2 The result of clustering a data instance that adheres to rule two defined in Table 2.3. 38

List of Tables

2.1 Definitions of terminology frequently used in this thesis. 3
2.2 An example conceptual representation of a VW Polo. 4
2.3 An overview of the rules for logical schema design. 5
2.4 An example logical schema for the first special case. 5
2.5 An example data instance for the first special case. 6
2.6 An example logical schema for the second special case. 6
2.7 An example data instance for the second special case. 7
2.8 An example logical schema for the third special case. 7
2.9 An example data instance for the third special case. 7
2.10 The notation for sets and corresponding elements used in the formal problem definition. 9

4.1 An overview of how literature was acquired. Cursive phrases indicate search queries used to browse Google Scholar. 22

5.1 An example data instance that adheres to the first rule. 26
5.2 The binary representation of Table 5.1. 26
5.3 Entity type representation clusters for Table 5.1. 26
5.4 Example data instances that adhere to the third rule. 27
5.5 Data profile attributes used in this solution, with their respective descriptions. The thick line separates explicit from implicit attributes. . . 28
5.6 Example data profiles. Rows, columns and values have been truncated for brevity. 29
5.7 An example input row and the result of running it through the entity type recognition algorithm. 32
5.8 Example aggregated results. The data instances each adhere to, in similar order, one rule in Table 2.3. 32

6.1 The metrics used for evaluation of the method. 36
6.2 The data characteristics for the first experiment. 40
6.3 The resulting metrics of the first experiment. 40
6.4 The data characteristics for the second experiment. 41
6.5 The resulting metrics of the second experiment. 42
6.6 The data characteristics for the third experiment. 42
6.7 The resulting metrics of the third experiment. 42
6.8 The data characteristics for the fourth experiment. 43
6.9 The resulting metrics of the fourth experiment. 43
6.10 The data characteristics for the fifth experiment. 44
6.11 The resulting metrics of the first experiment. 44

Chapter 1

Introduction

Nowadays, organizations store vast amounts of relational data in data warehouses and data lakes (and other possible structures). Each set of data within is created and managed by its own team in the organization and contains data on a collection of real world entities, where an *entity* is a specific real world thing with distinct and independent existence. Conceptually, each entity is part of some collection of entities that have the same attributes (e.g. cars, jobs, people, etc.), otherwise known as the *entity type*. But ING (where this thesis is conducted) identified the problem that it is often difficult for teams to work with data managed by another team; it is not always obvious which entity types each of the data sets in the organization contain data on, making it difficult to extract value and hindering data-driven progress. For example, a team working with cars might want to process data on engines, tires and license plates but do not own data on such entity types themselves. A naive solution would be to ask the owner(s) of each available data set for a comprehensive summary of the data, but this would require time of the owner(s) that they would otherwise (rather) spend on their own tasks. As such, an automated solution is desired.

Furthermore, *semantic heterogeneity* arises when multiple data sets contain data on the same entity types but represent them differently, is usually caused by the use of different *logical schemas*, and exacerbates the problem (Giunchiglia et al., 2020); semantically heterogeneous data sets are not immediately compatible but are generally more valuable when combined. For example, two teams (A and B) in the organization manage data on car engines but use different logical schemas to store the data. Team A wishes to compare their engines' performance to those of other teams, but require data on the other engines to do so. Team A does not know that team B also stores data on engines, so they cannot ask team B for the data directly. Instead, they wish to find the data in the available data warehouse or lake, but struggle because there is a large number of data sets stored within that they know nothing about. And above that, team B's engines are represented in the data differently than team A's engines, making it more difficult to recognize team B's engine data as suitable. While humans are relatively good at recognizing conceptual entity types and semantic heterogeneity, human computation does not scale well. Therefore, this thesis proposes an entity type recognition method to help people identify data sets of their interest stored within their organization.

In the future, ING would like to evaluate whether entity type recognition could be a part of data integration. This would be an approach alternative to what precursors to this thesis focused on (Ionescu, 2020; Psarakis, 2020), namely schema matching. Instead of matching data content and schemas directly, entity type recognition could be used to match data instances on semantic similarity and recognize semantic heterogeneity in order to combine data. The goal is to offer a solution that recognizes the entity types present in relational data sets of interest and, in case of semantically

heterogeneous data, points out the cause for semantic heterogeneity. The latter is important because it gives context on how the semantically heterogeneous data could be combined.

For this thesis, the problem space of entity type recognition is narrowed down to a specific organization—ING in this case. When approaching the problem from a specific organization's perspective, the problem space is constrained naturally: it is known which conceptual entity types possibly reside in the data and there are examples of how each of these can be represented; the employed DBMS(s) is/are known; and the handful of general design choices for logical schemas are known. Such design choices specify how entities of a certain entity type are represented in the data instances, and are defined in the logical schema. An example design choice would be to store all data on an entity type in one table, as opposed to dividing parts of the entity type over multiple tables and keeping references (e.g. storing birth dates in another table than the other characteristics of a person). How these constraints will be exploited will be elaborated on in chapter 5.

1.1 Contributions

The goal of this thesis is to offer a solution for database entity type recognition, which also helps to recognize semantic heterogeneity in relational data stored within a specific organization. As such, the contributions of this thesis are as follows:

- A method for entity type recognition based on clustering and data profiling. Data profiles describe metadata on the given relational data, but contain no real data for privacy preservation purposes. In this case, the profiles consist of cardinalities (as described in (Abedjan et al., 2015)).
- An implementation of the method that, based on given data profiles, performs entity type recognition on data sets by matching each row in each data set to the best-fitting data profile. The matching is performed in a rule-based fashion to maximize explainability, as is often desired in a bank.
- A feasibility evaluation of the method, acquired by performing experiments on the implementation.

1.2 Thesis Outline

This thesis continues by giving a detailed and formal problem description in chapter 2, where the constraints of the problem space will also be covered. The research methodology is then described in chapter 3, after which relevant literature will be discussed in chapter 4 in two parts. The first part covers literature that aims to solve similar problems, while the second part covers literature that is (partially) incorporated into or had influence on the proposed solution covered in chapter 5. The proposed solution will be evaluated in chapter 6 using experiments, after which the thesis will be concluded in chapter 7. The conclusion includes recommendations for future work and how to continue research using the proposed solution.

Chapter 2

Problem Description

This chapter will give a detailed description of the problem at hand. First, the terminology that will be used frequently throughout this thesis is defined in Table 2.1. After this overview has been provided, section 2.1 covers the cause of the problem through description and examples, and finally, the problem will be presented formally.

TABLE 2.1: Definitions of terminology frequently used in this thesis.

Entity	A thing in the real world with independent existence.
Entity Type	A collection of entities that have the same attributes and adheres to a taxonomy.
Conceptual Schema	A concise description of data requirements, including properties, entity types, relationships and constraints. Does not include implementation details (Elmasri et al., 2000).
Conceptual Instance	A collection of entities, with each entity represented as specified in the conceptual schema.
Logical Schema	A mapping of the conceptual schema to the DBMS-specific data model, translating the conceptual schema into terms of tables, columns, column data types, foreign keys, etc. (Elmasri et al., 2000).
Data Instance	The stored state of entities in the DBMS, as a result of applying the logical schema to the conceptual instance. Data instances are a part of data sets.
Semantic Heterogeneity	The problem that arises when multiple data sets contain data on the same conceptual entity type, but represent it differently due to the application of different logical schemas (Giunchiglia et al., 2020).
Data Profile	An informative summary of a data instance.
Data Integration	Combining data from several sources, with the unified view as the result.

2.1 The origin of the problem

To get a good understanding of the problem, it should be clear how the problem came to be. In other words: it should be clear how an entity is transformed into its representation in the data instance and why semantic heterogeneity occurs. Therefore, assume the following as a running example.

Assume the task to store a specific Volkswagen (VW) Polo in a relational database. The VW Polo is the *entity*; it is a real world thing with distinct and independent existence. The entity itself can obviously not be stored in a database, but a corresponding representation can. The first step to getting to that representation is defining the *conceptual schema*, which concisely describes the data requirements (entity type, properties, relationships and constraints) but no implementation details (Elmasri et al., 2000).

Rather than defining a conceptual schema for every entity, it is done for collections of entities that share the same attributes. These collections are called *entity types*, and each entity type has its own definition in the conceptual schema. The result of applying the conceptual schema to entities of its corresponding type is the collection of data requirements for the given entities; the *conceptual instance*. The VW Polo from the example would look something like in Table 2.2.

TABLE 2.2: An example conceptual representation of a VW Polo.

entity type	vehicle → motorized → car → compact
relationships:	has engine, has tires, has license plate, owned by company, etc.
make	Volkswagen
model	Polo
color	Dark Green
year built	2014
mileage	254,429
etc.	

In the next step, the *logical schema* maps the conceptual schema to the DBMS's data model in terms of tables, columns, column data types, foreign key constraints, etc. (Elmasri et al., 2000). It is now important to note that entity types generally adhere to a certain taxonomy, as depicted in Figure 2.1. For example, the Volkswagen Polo is a compact car, which is a type of car, which is a type of motorized vehicle, which is a type of vehicle. The presence of entity type taxonomies depends on the definition of the conceptual schema within the organization, as does their level of detail; some entity types might be part of a fine-grained taxonomy, while other entity types might be alone in theirs. However, this thesis assumes that entity type taxonomies are defined as simple as possible, meaning that taxonomies contain no entity types that have only one subtype. For example, if compact cars in Figure 2.1

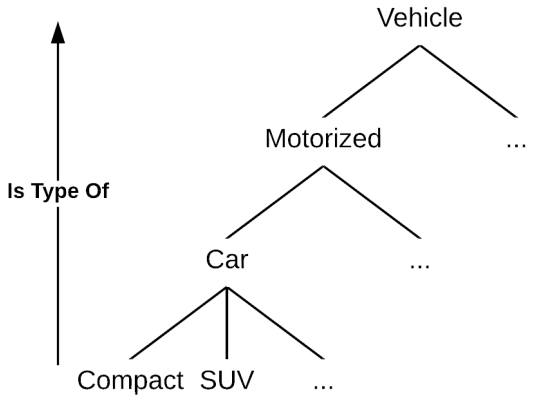


FIGURE 2.1: An example entity type taxonomy, truncated to compact cars and SUVs.

were the only type of cars, there would be no Car subtype. The definition of each taxonomy partially dictates the design of logical schemas, resulting in one of three special cases that will be considered in this thesis.

2.1.1 The rules leading to special cases

Each table in a logical schema adheres to one of the three rules with regard to entity type taxonomy listed in Table 2.3, resulting in three special cases. Each of these cases will be described using the running example of the VW Polo. In reality, there are many more possible rules for logical schema design, as well as the special cases that result from applying those rules. However, this thesis will consider only these three rules to maintain focus on the overall method, leaving other special cases subject to future work.

TABLE 2.3: An overview of the rules for logical schema design.

#	Rule
1	The table contains data on multiple subtypes from some entity type in its taxonomy.
2	The table contains data on a type at the bottom of its taxonomy, and only that type.
3	The rows in the table contain data on a combination of multiple entity types, or part thereof, possibly from different taxonomies.

1. The table contains data on multiple subtypes from some entity type in its taxonomy.

In this case, the logical schema is designed such that a resulting table contains data on all of the types in one taxonomy from some point downward. For example, if the logical schema defines a table for all types of vehicles (see Figure 2.1), that would look something like in Table 2.4.

TABLE 2.4: An example logical schema for the first special case.

Table:	Columns:
Vehicle	`[int:id(PK), date:year_built, int:engine_id(FK)` `int:license_id(FK), string:make, string:model,` `string:color, int:front_left_tire_id(FK), etc.]`
LicensePlate	`[int:id(PK), string:number, date:valid_through, etc.]`
Tire	`[int:id(PK), float:recommended_pressure, etc.]`
etc.	

As can be concluded from Table 2.4, the first table is designed to contain data on all types of vehicles, the second on all types of license plates, and the third on all types of tires. Foreign key constraints are used to specify relationships between the entities represented in each table. This means that the VW Polo of the example, a compact car, will be represented in a table that also holds representations of SUVs and other types of vehicles, but points to other tables for data on the license plate and tires. The data instance resulting from applying this logical schema to the conceptual instance in Table 2.2 can be seen in Table 2.5.

TABLE 2.5: An example data instance for the first special case.

Table:	Rows:
Vehicle	`[id=0, year_built=2014, engine_id=0, license_id=0, make=Volkswagen, model=Polo, color=DarkGreen, front_left_tire_id=0, etc.]`
LicensePlate	`[id=0, number="0-AAA-00", valid_through=2022-1-1, etc.]`
Tires	`[id=0, recommended_pressure=2.1, etc.]`
Tires	`[id=1, recommended_pressure=2.1, etc.]`
Tires	`[id=2, recommended_pressure=2.1, etc.]`
Tires	`[id=3, recommended_pressure=2.1, etc.]`
etc.	

2. The table contains data on a type at the bottom of its taxonomy, and only that type.

In this case, the logical schema is designed such that a resulting table contains data on one entity type at the bottom of the taxonomy. For the example taxonomy seen in Figure 2.1, that means that compact cars, SUVs and other types of cars each get their own table. A logical schema that adheres to this rule would look something like in Table 2.6.

TABLE 2.6: An example logical schema for the second special case.

Table:	Columns:
Vehicle.Car.Compact	`[int:id(PK), date:year_built, int:engine_id(FK) int:license_id(FK), string:make, string:model, string:color, int:front_left_winter_tire_id(FK), etc.]`
Vehicle.Car.SUV	`[int:id(PK), date:year_built, int:engine_id(FK) int:license_id(FK), string:make, string:model, string:color, int:front_left_winter_tire_id(FK), etc.]`
LicensePlate.Car	`[int:id(PK), string:number, date:valid_through, etc.]`
Tire.Car.Winter	`[int:id(PK), float:recommended_pressure, etc.]`
etc.	

Each table in this logical schema is designed to hold data on only one entity type and use foreign key constraints to specify entity relationships. This means that the VW Polo will be represented in a table that stores data only on the same entity type (compact cars), and points to other tables for data on the license plate and winter tires. The result of applying this logical schema to the conceptual instance in Table 2.2 can be seen in Table 2.7

3. The rows in the table contain data on a combination of multiple entity types, or part thereof, possibly from different taxonomies

In this final case, the logical schema is designed such that a resulting table represents multiple entity types from different taxonomies. For the VW Polo example, this

TABLE 2.7: An example data instance for the second special case.

Table:	Rows:
Vehicle.Car.Compact	[id=0, year_built=2014, engine_id=0, license_id=0, make=Volkswagen, model=Polo, color=DarkGreen, front_left_winter_tire_id=0, etc.]
LicensePlate.Car	[id=0, number="0-AAA-00", valid_through=2022-1-1, etc.]
Tire.Car.Winter	[id=0, recommended_pressure=2.1, etc.]
Tire.Car.Winter	[id=1, recommended_pressure=2.1, etc.]
Tire.Car.Winter	[id=2, recommended_pressure=2.1, etc.]
Tire.Car.Winter	[id=3, recommended_pressure=2.1, etc.]
etc.	

could mean that the resulting table stores data on the car, its tires, and its license plate. An example logical schema that would result in such a table can be seen in Table 2.8.

TABLE 2.8: An example logical schema for the third special case.

Table:	Columns:
Vehicle	[int:id(PK), date:year_built, string:make, string:model, string:color, string:engine_name, string:license_number, date:license_valid_through, float:recommended_tire_pressure, etc.]

The resulting data instance of applying this logical schema is a single table that contains data on all vehicle types and their related entity types. The VW Polo will be represented as in Table 2.9.

TABLE 2.9: An example data instance for the third special case.

Table:	Rows:
Vehicle	[id=0, year_built=2014, make=Volkswagen, model=Polo, color=DarkGreen, license_number="0-AAA-00", license_valid_through=2022-1-1, recommended_tire_pressure=2.1, etc.]

2.2 The problem

Teams within an organization will design their logical schemas based on rules such as those covered in subsection 2.1.1. The data instances resulting from applying these logical schemas are stored in data warehouses or lakes withing the organization. With the right authorization, members of the organization can access the data instances. However there are likely a lot of data instances available, and a common problem when working with a lot of data is that it is not always obvious which conceptual entity types are represented in each data instance, making it difficult to use the data sensibly. Furthermore, when different logical schemas have been used to map the same entity type to different representations in the data instances, *semantic heterogeneity* occurs and exacerbates the problem (Giunchiglia et al., 2020). *Entity*

type recognition can be used to solve these problems by recognizing conceptual entity types for which data instances in each given data set contain data. Within an organization, the problem space for entity type recognition can be narrowed to that specific organization, making it a more feasible task. This thesis therefore makes the following assumptions to narrow down the problem space:

1. The organization knows which conceptual entity types might be encountered in their data and are interested in recognizing only those entity types.
2. Each data instance adheres to one of the rules listed in Table 2.3. The rules are covered in detail in subsection 2.1.1.
3. The organization provides example data instances for each of their entity types' distinct representations. These data instances are labeled with their respective entity type(s) (training data).
4. All entity types are defined in one conceptual schema, no entity type has more than one conceptual specification.

With the assumptions above, the problem will be defined in two parts, **P1** and **P2**. The proposed solution to both problems will be covered in chapter 5.

2.2.1 Problem one (P1)

> *Given unlabeled data instances and the data instances labeled with their respective entity type(s), recognize which of the known entity types are likely stored within each unlabeled data instance.*

Imagine that an organization stores data on, among other entity types, the vehicles they lease to their clients like in the running example used in this chapter. In this case, the vehicles are represented in the data instances following one of the three rules defined in Table 2.3. In other words, each vehicle can be stored in one of three semantically heterogeneous data instances. The goal is to identify the entity type(s) represented for each row in each data instance. For example, the entity types of the row given in Table 2.9 should be recognized as `[vehicle.car.compact, license_plate.car, tire.car.winter]`, thus distinguished from `[vehicle.car.SUV, license_plate.car, tire.car.winter]`, `[vehicle.car.compact]` and all other (combinations of) entity types the organization keeps data on.

2.2.2 Problem two (P2)

> *Given the result of P1, recognize the rules used to map the conceptual schema to the logical schema, for each logical schema.*

While P1 is focused on the mere recognition of entity types in data instances, this problem is focused on recognizing semantic heterogeneity. This is important because it enables data integration by putting data instances in terms of the conceptual schema. In essence, this is the reverse engineering of the steps covered in section 2.1. These steps can then be re-applied to create a unified view of data on the same entity type, i.e. perform data integration. Note that the "rules" in this problem definition refer to the rules listed in Table 2.3.

2.3 Formal Problem Definition

This section will conclude the chapter by formalizing the problem description given in section 2.2, providing a concise mathematical definition. An overview of the used set and element notation is provided in Table 2.10.

TABLE 2.10: The notation for sets and corresponding elements used in the formal problem definition.

Set	Definition	Element
D	The set of data instances	d
R	The set of all rows in a data instance	r
E	The set of entity types used in the organization	e
C	The set of conceptual representation of entities	c
L^e	The set of logical schemas where entity type e is defined	$\mapsto^e$

An organization stores relational data in a collection of data instances D. Each data instance $d_i \in D$ comprises a collection of rows R_i, where i indicates the i^{th} data instance in D. Analogously, with j indicating the j^{th} row, the relationship $r_j \in R_i = d_i \in D$ holds.

Each row r in any data instance represents at least one entity depending on the logical schema. As such, each row r in any data instance represents a collection of at least one conceptual entity type E_r where $|E_r| \geq 1$ from a collection of entity types $E_r \subset E$ used within the organization.

Each data instance is the result of applying a logical schema to a conceptual instance. Each logical schema $\mapsto^e \in L^e$ defines the rules to map an entity type $e \in E$ to a row r in a data instance. The process of mapping a conceptual representation c of an entity of type $e \in E$ is defined as $c \mapsto^e_m r$, where $\mapsto^e_m$ is the m^{th} logical schema in L^e. It can happen that $L^{e_i} \cap L^{e_j} \neq \emptyset$ for $\{e_i, e_j\} \subset E$ and $i \neq j$, indicating that there is a logical schema that combines entities of types e_i and e_j to one row in a data instance (rule three in Table 2.3).

The individual problems are then defined as:

P1: Given unlabeled data instances D and data instances previously labeled with their entity types D', predict the entity type(s) E_r for every $r \in D$ using the examples in D'.

P2: Given the results of P1, recognize which of the following rules[1] holds for every $\mapsto^e \in L^e$ for every $e \in E$.

1. E_R, the set of entity types represented in R, is taxonomically a subset of some entity type e, i.e. $E_R \subseteq e \in E$ and $|E_R| > 1$.
2. $|E_R| = 1$ and $e \in E_R$ has no subtypes, i.e. its only taxonomic proper subset is the empty set.
3. $|E_r| \geq 1$, $|E_R| > 1$, and $E_r \subset E_R$, where $e_i \not\subseteq e_j$ for each $\{e_i, e_j\} \subset E_R$.

[1] As listed in Table 2.3.

Chapter 3

Research Methodology

The following parts of this thesis will cover the research conducted to result in the proposed solution. This section will introduce the applied research methodology, including a description of the research environment. The decision making process throughout the thesis will be covered first.

3.1 Decision making process

This thesis succeeds the work of two theses conducted at ING, but takes a slightly different path. Both Ionescu and Psarakis conducted research into schema matching for data integration (Ionescu, 2020; Psarakis, 2020), which is different from this thesis. The decision to take this path came from discussions with the supervisors at Delft University of Technology and ING, where new approaches to data integration were initially the subject. Because of recent personal interest in data profiling, discussions were generally held on how data profiling could potentially fit into a data integration subject for this thesis. As a result, database entity type recognition was identified as a potential subtask for data integration that could be a standalone thesis topic.

After these discussions, the literature review covered in chapter 4 uncovered that there was little published research on this or similar topics, especially on literature incorporating data profiling. That discovery led to the decision to develop a database entity type recognition solution that utilizes data profiling by personal thought process and supportive literature review. Altogether, the literature review in the next chapter is divided into two parts. The first part (section 4.1) covers somewhat similar research that was recently published, while the second part (section 4.2) discusses literature that will support the solution proposed in chapter 5.

Finally, during the development stage of this thesis, the implementation to evaluate the proposed solution was developed. Some parts of the implementation were implemented based on the supportive literature covered in section 4.2, but most of the pipeline was developed through personal thought process. Some times, the tasks in the pipeline are tackled through the first solution that came to mind. If that solution worked well, it would remain in the pipeline, but a substitute was found by examining existing literature otherwise. In conclusion, the success of this approach shows that simplicity can be fruitful.

3.2 The research environment

The proposed solution, covered in chapter 5, will be made specially for ING, but can be adapted to different environments if desired. But because ING is a bank, the solution has to adhere to the expectations of a bank. This means that integrity is of the

utmost importance, and thus that privacy and explainability standards are high. The solution will therefore store no sensitive data or any data at all, and explainability will be guaranteed by using rule-based decision making.

ING will be providing two types of data: generated data and real data used at ING. The generated data should resemble real data at ING[1], but without sparking any integrity concerns when used outside of ING's secure environment; it contains no real (sensitive) information. The generated data is used because it is versatile, providing options for any desired data set size or complexity that can be employed in the evaluation of the proposed solution's performance. The real data will be used in a secure environment to validate the evaluation results and, with that, the applicability of the proposed solution to real problems at ING.

To conclude, there will be weekly contact with supervisors at Delft University of Technology and ING, and there will be meetings with teams within ING on a biweekly basis to inquire ideas for and identify possible issues in the solution. These occasions are to provide updates and receive feedback on the research progress. This is vital to the process, as it allows for validation of the research and whether the proposed solution will serve its purpose once finished.

[1]Though the generator was programmed by an expert with many years of experience at the company, the resemblance must still be validated.

Chapter 4

Background

This chapter will cover background information in two parts. In section 4.1, literature on similar research will be reviewed. However, there is little recently published literature on entity type recognition, and there will thus mostly be review of techniques that are similar to entity type recognition. Then, section 4.2 will focus on techniques that will support the proposed solution. The chapter will be concluded in section 4.3, which will include an overview of the acquisition of the reviewed literature in Table 4.1.

4.1 Similar research

This first part of the literature review is aimed at analyzing similar research. However, "entity type recognition" is either not a commonly accepted name for the field of research, or it has seen little popularity; Searching for *'"entity type recognition" -omics -biology -biomedical'*[1] on Google Scholar gives only 67 results, of which all were published between 2009 and 2021. Before this time, somewhat similar research was published as *"relational database reverse engineering"*. Instead of recognizing entity types within relational databases, this research focused on uncovering the conceptual schema from relational databases using reverse engineering. Because of the similarity between these two fields of research, literature on database reverse engineering will be reviewed in subsection 4.1.1, after which entity type recognition research will be covered in subsection 4.1.2.

4.1.1 Reverse engineering

In 1994, Andersson identified problems with relational databases that obstructed the path to distributed computing, something that was rapidly increasing in popularity at that time (Andersson, 1994). They saw that lack of documentation and non-uniformity across databases prevented them from working together or otherwise were the cause for high maintenance costs. That same year, Chiang et al. identified similar issues, stating that some databases in organizations exist without anyone knowing what the data or relationships between the data are (similar to this thesis) (Chiang et al., 1994). Nearly two decades later, Malpani et al. published a tool that would also reverse engineer databases (Malpani et al., 2010), but with the goal to bootstrap application development. The first two of these publications proposed reverse engineering relational databases into the conceptual schema. The conceptual schemas would be represented in the form of an extended Entity-Relationship (EER) model, or something similar (ECR+) in case of (Andersson, 1994). Malpani et

[1] If "omics", "biology" and "biomedical" are not excluded words, there are about two million search results. However, most of these results are outside the scope of this thesis.

al. proposed to reverse engineer into Microsoft's Entity Data Model (EDM), a model that contains implementation specific information and is therefore not a conceptual schema. The three approaches will now be analyzed individually.

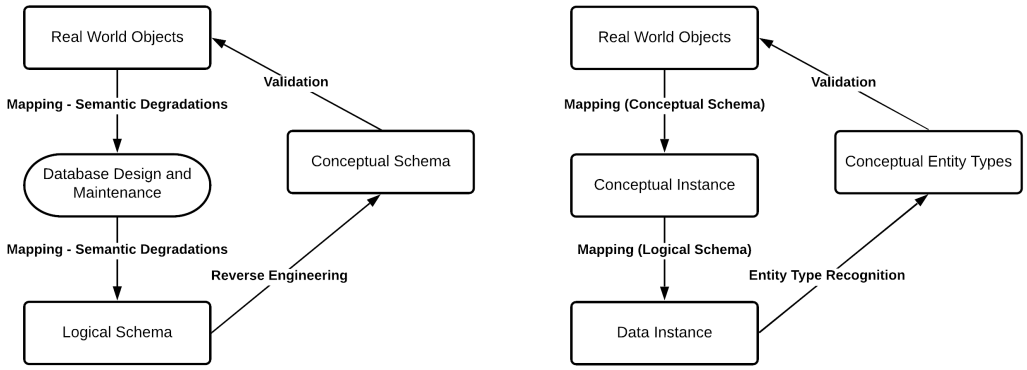
Andersson: reverse engineering to ECR+

Andersson proposed to extract the conceptual schemas from data manipulation statements present in application code. They argued that all relationships represented in a database should be present in a set of queries that reflects the database manipulation completely, and thus that their approach would completely uncover the semantics as intended by the database designer or application programmer. The problem with this approach is that the application code must cover all aspects of the database design, as well as that the relevant application code must be available. These are reasonable assumptions in Andersson's problem space, where they look for options to modernize outdated database systems of which is known what they contain data on. However, the problem in this thesis is that it is not known which entity types are represented in the data, only which entity types we might encounter. Furthermore, it might not be known which application produced the data, or there might not even be application code available. Andersson's approach is reverse engineering from the application level, not from the data level, and thus not a suitable approach for this thesis.

Chiang et al.: reverse engineering to EER

Chiang et al.'s approach to database reverse engineering is relatively similar to database entity type recognition as described in chapter 2, as is illustrated in Figure 4.1. Although rather than detecting entity types, they aim to infer the conceptual schema from data instances using the corresponding table and attribute names, and primary keys (Chiang et al., 1994). Their methodology consists of four steps to infer the EER model from the database:

1. *Infer functional dependencies and primary keys, and keep these in third normal form (3NF).* A functional dependency $X \rightarrow A$ indicates that the values of attribute set X uniquely determine the values of an attribute A (Papenbrock et al., 2015).



(A) Database reverse engineering as described in (Chiang et al., 1994, Fig. 1).

(B) Database entity type recognition as proposed in this thesis.

FIGURE 4.1: The database reverse engineering process vs. the database entity type recognition process.

Keeping the functional dependencies in at least 3NF eases the rest of the process by ensuring that each entity in the EER model is either strong, regular or weak, but not a combination of the three. Modern systems using this approach may benefit from the relatively recently published evaluation study on functional dependency discovery algorithms in (Papenbrock et al., 2015). Experiments could be done with relaxed functional dependencies ((Hai et al., 2019)) to evaluate the feasibility of this approach on data lakes instead of one DBMS.

2. *Classification of relations (strong, regular, weak).* Classification is performed by matching each relation to the following rules:

   Strong The primary key of the relation does not properly contain the key of another entity.

   Regular The primary key of the relation is the concatenation of primary keys of other entities.

   Weak The primary key consists partially of keys of other relations, and partially of keys not in any other relation.

3. *Generation of inclusion dependencies.* Chiang et al. included a set of heuristics for the generation of inclusion dependencies. Since their publication, many efforts have been made to perform inclusion dependency detection more efficiently, especially because the size and amount of available data instances has scaled up significantly since 1994. For instance, Tschirschnitz et al. have proposed an algorithm to efficiently detect single-attribute inclusion dependencies on a lot of data in (Tschirschnitz et al., 2017). Similar efforts have been made for multiple-attribute inclusion dependencies in (Shaabani et al., 2017), exploiting the characteristics of single-attribute inclusion dependency detection.

4. *Identification of entities and relationships, and assignment of attributes.* The previously classified relations are identified as entities or relationships, and attributes are assigned to each entity. The result is a model that meets EER specifications.

The result of this reverse engineering method is the conceptual schema for the database in question. This result would solve P2 (subsection 2.2.2), as it is now known how every data instance relates to the conceptual schema. It would also solve P1 (subsection 2.2.1) because conceptual schemas contain entity type specifications. However, the approach raises viability concerns with regards to time complexity; the worst-case time complexity for any dependency algorithm, i.e. for both functional and inclusion dependencies, is exponential in terms of the number of attributes (i.e. columns) (Abedjan et al., 2015). Given the very large amounts of data sets organizations have to deal with in the present, and the fact that this method calls for the detection of dependencies twice, the time complexity issue makes this specific method a non-viable approach for this thesis.

Malpani et al.: reverse engineering to EDM

Malpani et al. had the goal to bootstrap application development by reverse engineering databases (Malpani et al., 2010). They stated that available tools for that purpose shared one common shortcoming; the tools would assign each table one entity type and each foreign key one relationship, disregarding the possibility of

inheritance or other relationships. Thus, Malpani et al. propsed a method and published a tool called EdmGen++[2] that would fit a database-first approach to application development. EdmGen++ would reverse engineer relational databases into Microsoft's entity data model (EDM) in a fashion relatively similar to the methodology published in (Chiang et al., 1994). The steps in EdmGen++'s method are:

1. *Extracting a default model from the table.* This model is made by evaluation of the following rules.
    - Every table represents an entity.
    - Every foreign key represents a relationship.
    - Every table consisting merely of foreign keys to two other tables represents a many-to-many relationship.
2. *Further identify the entities and relationships in the default model by evaluating each rule specified in the method ((Malpani et al., 2010, Table 1)).* Before each rule is evaluated for each entity or relationship, certain preconditions must be met. If a precondition is not met, the rule is not evaluated for that specific entity or relationship. Preconditions were introduced in order to avoid performing computations that are likely to give meaningless results. In addition, postconditions must be met in order to avoid superfluity of information.

The upside to this method is that the resulting model enables object-oriented communication between applications and the database without manual implementation of the application code. The downside to this method is that it cannot function on (relational) data that does not reside in a functioning DBMS, as it requires access to schema specifics such as foreign key definitions. This constraint could be worked around by manually providing foreign key relationships, or by finding them through inclusion dependency detection as previously covered, in case data comes in another form such as in comma separated files.

Another issue that would arise if either Chiang et al.'s or Malpani et al.'s method were to be used in this thesis, is that both methods require the availability of sensible table and column names to result in sensible models. Furthermore, to automatically identify semantic heterogeneity, table and column naming should be consistent for the representation of similar entity types throughout the entire organization. Otherwise, these methods would require a human in the loop to verify or adjust every result. How these conditions will be avoided for the entity type recognition solution will become clear in chapter 5.

4.1.2 Entity type recognition

There are little publications available when searching for "entity type recognition" (except in the field of biology), indicating that the term itself might not be well established. However, efforts toward solving a problem similar to the one identified in this thesis were made in (Sleeman et al., 2015) and (Giunchiglia et al., 2020). Both perform entity type recognition in knowledge bases, but Sleeman et al. focuses on detecting fine-grained entity types (e.g. distinguish compact cars from other vehicles), and Giunchiglia et al. focuses on detecting semantic heterogeneity between knowledge bases. The methods will now be reviewed individually.

[2]The source code of the project is no longer available. However, an executable seems to be published at https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/edm-generator-edmgen-exe.

Sleeman et al.: fine-grained entity type recognition in knowledge bases

Sleeman et al. state that schemas for structured data are often either unavailable or semantically weak, making them unsuitable evidence for entity type recognition (Sleeman et al., 2015), especially for fine-grained entity type recognition. Thus, they propose using semantic graphs to perform entity type recognition, but acknowledge that this approach brings along challenges that have to be overcome. Semantic graphs are required to be ontologically defined to make this feasible, and on top of that, it requires the data to follow the same ontology sufficiently—something that is likely rare in many big data scenarios. The main goal of their contribution is to identify coreference within relational data. That is, identify which parts of given data represent the same entity types, while distinguishing between fine-grained entity types.

The first step in the entity type recognition process, as defined by Sleeman et al., is acquiring a well-defined model of entity types from a knowledge base. This model is limited to a certain domain, such as the medical domain, and reinforced by using it to identify entity types in data in the same domain. The entity type model consists of equivalence, hierarchical and pattern similarity relationships.

The next steps is concerned with improving the model's efficiency by identifying the attributes of each entity type that have a high likelihood of indicating such an entity type. Entities' types can then be identified based on the presence of these specific attributes, instead of matching each attribute of each entity of unknown type to each attribute of each known entity type, thus avoiding possibly unnecessary computation. As this is an approach based on linguistics, Sleeman et al. have adopted measures to properly handle synonymous attributes.

The final step is classification, which is done through support vector machine (SVM) models. The features for the SVMs are the attribute values that remained after performing the previous steps. All entities for which no entity type could be identified, are classified as the same "*unknown*" type.

Sleeman et al.'s approach to entity type recognition relies on semantic graphs, ontology and proper presence thereof. There is no guarantee that such requirements can be satisfied in this thesis' setting due to the possible specificity of entity types used within organizations. However, if organizations are able to provide semantic graphs regarding their internally utilized entity types, this approach might show promising results.

Giunchiglia et al.: semantic heterogeneity detection for knowledge bases

In their work, Giunchiglia et al. aim to perform entity type recognition in knowledge bases without eliminating semantic heterogeneity (Giunchiglia et al., 2020). Rather, they want to understand how semantically heterogeneous representations relate to each other—quite similar to solving P2 stated in subsection 2.2.2.

Giunchiglia et al. propose to embrace semantic heterogeneity rather than trying to resolve it, because diversity in data is unavoidable. Their approach therefore involves acquiring and maintaining a set of high-quality knowledge bases, which will in turn be used to perform entity type recognition. The knowledge bases are contextually compared using unity and diversity as metrics, where unity indicates shared attributes and diversity indicates the opposite. Note that strong unity does not exclude strong diversity or vice versa.

In the method's first step, the comparison metrics mentioned above will be used to determine which reference knowledge base can best be used to perform entity type recognition on a given input knowledge base. This decision is made based on the reference knowledge base that maximizes both the unity and the diversity with the input knowledge base. These conditions should minimize misalignment between the two knowledge bases and minimize the confusion between entity types respectively.

The next step is the classification of entity types in the input knowledge base. For this purpose, Giunchiglia et al. represent the reference knowledge bases, similar to the approach in (Sleeman et al., 2015), as semantic graphs. They then use NLP to deal with stop words, synonyms, etc. in the labels of each entity type and their attributes to avoid misclassifications based on minor variations in the labels. Finally, each selected reference knowledge base is used to train its own decision tree model used to recognize entity types.

Similar to Sleeman et al.'s approach, Giunchiglia et al. require the availability of semantic graphs. However, now there should be multiple, and each should be of high quality. Furthermore, the labels in the semantic graphs should also be interpretable by an NLP pipeline. There is, again, no guarantee that these requirements can be met in this thesis. And even if semantic graphs could be provided, type and attribute labels might be too domain-specific for an NLP pipeline to provide sensible results.

4.2 Supportive research

From section 4.1 can be concluded that, in order to solve the problems defined in section 2.2, a different approach than previously published should be taken. This section will therefore cover literature on data profiling, as that supports the solution proposed in chapter 5.

Data profiling is the process of analyzing data and collecting its metadata (Abedjan et al., 2015). There are several use cases for data profiling of relational data, such as database reverse engineering, data integration and big data analytics. A classification of the possible data profiling tasks has been given in Figure 4.2. It is based on the comprehensive contributions provided in (Naumann, 2014; Abedjan et al., 2015). This section will continue by elaborating on the data profiling tasks that will likely prove to be useful for the entity type recognition solution.

4.2.1 Cardinalities

Cardinalities are arguably the simplest form of metadata that can be extracted from relational data. They provide insight on data instance characteristics such as row or column count, length of column values, amount of distinct values, etc. Counts and lengths can be determined using a single, stateless pass over each data instance, making their computations straightforward and cheap. Cardinalities that require stateful passes over data instances are computationally more expensive, but the difference is relatively small when it is done right. Counting distinct values, for example, can be done in relatively small space complexity using hashing methods.

Harmouch et al. have conducted an experimental survey regarding cardinality estimation (Harmouch et al., 2017). They state that the number of distinct values

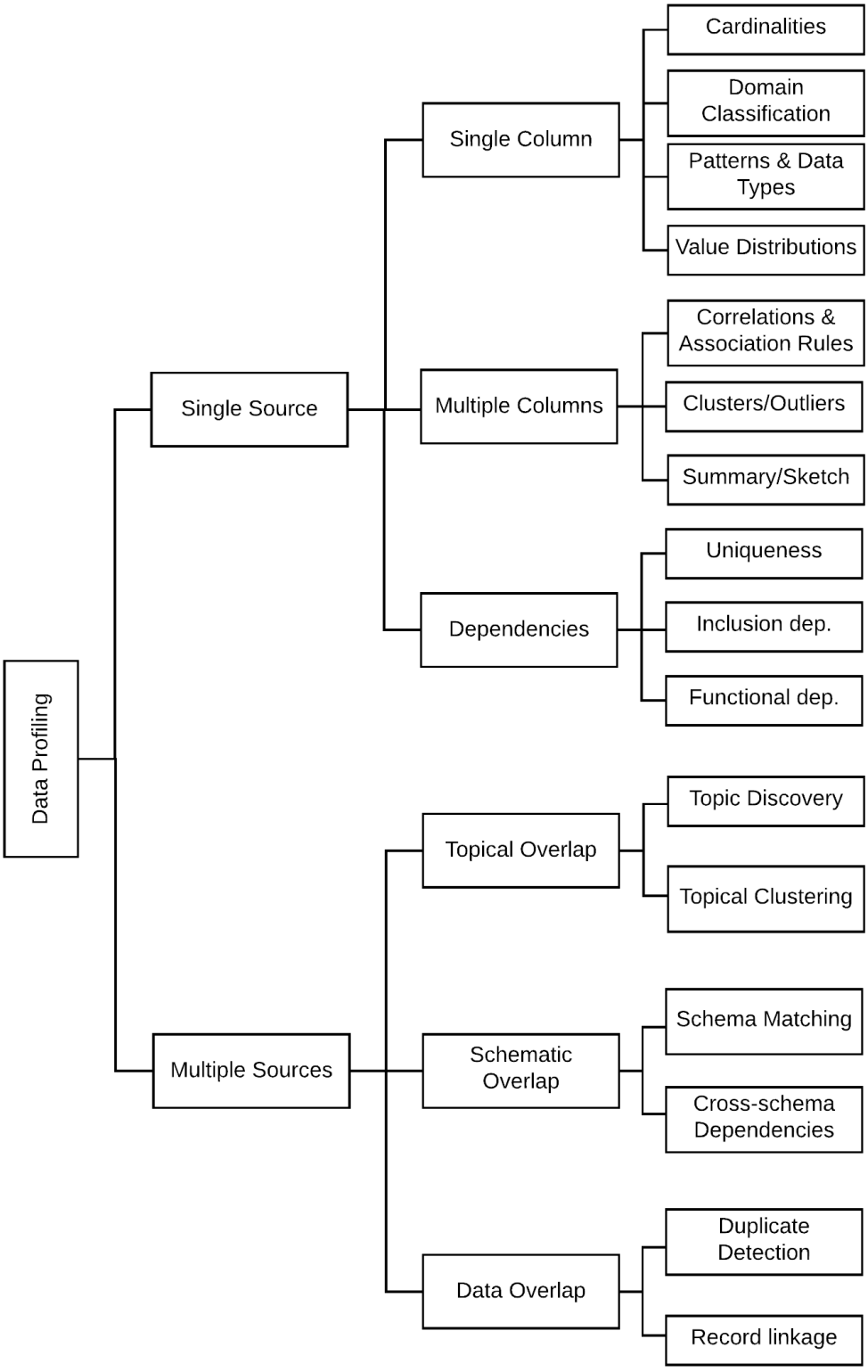


FIGURE 4.2: A classification of data profiling tasks. The leaves represent profiles, while the higher levels represent categories. Profiles are not limited to their category, but are grouped based on (Abedjan et al., 2015, Fig. 1) and (Naumann, 2014, Figure 1).

is among the most important metadata, and thus that it is important to use algorithms that efficiently estimate this cardinality. In use cases where estimations are a sufficient measure, time and space complexity can be reduced by using the right algorithm, as can be concluded from (Harmouch et al., 2017).

4.2.2 Patterns and data types

Recognizing basic column data types is relatively easy. For instance, if a column comprises merely decimal values, it can easily be classified as such. Similar rules hold for strings, characters, integers, etc. More complex data types, such as dates, usually follow strict patterns that can be exploited to identify such data types.

Patterns can also be used to determine whether a value fits a certain column. This is useful for tasks such as detecting erroneous values in a data instance (quality preservation), but also for accepting or rejecting candidates in entity type recognition. However, when the values of a column follow a certain pattern, this pattern is not necessarily defined in a data instance, but rather hidden in an application. And thus, it is difficult to verify data quality or perform entity type recognition based on this characteristic without identifying the patterns first.

A common way of specifying a pattern that should be followed is through regular expressions (regex). These expressions can get complex, especially when expecting longer, non-uniform values. It is thus desirable to perform the identification of patterns automatically, such as through the methods described in (Fernau, 2009).

4.2.3 Semantic domain classification

Semantic domain classification regards, as the name suggests, the meaning of a column. That is, rather than determining the column data type, the goal is to determine whether a column represents addresses, birth dates, IPv4 addresses, etc. Desirably, the distinction between semantic domains such as birth date and expiration date is also made.

Many efforts have been made to successfully perform semantic domain classification. Khalid et al., for instance, used a keyword-based approach to match both columns values and names (if available) to a topic (Khalid et al., 2019). Koehler et al. took a more complex approach by introducing dependencies that generate tuples for data columns in a similar semantic domain (Koehler et al., 2017). Both approaches rely on available keyword corpora, and Khalid et al.'s approach is even more restricted because it applies only to values in natural language. Column values can adhere to very specific semantic domains within organizations, making either of these approaches ill-suited to contribute to solving the problems stated in section 2.2.

Zhang et al. have recently published Sato, an algorithm that attempts to extract context from all columns in a table to identify the semantic domain of each column (Zhang et al., 2019). Out of the box, Sato works well in a simple general setting, i.e. making distinctions between countries, cities and dates. In an organization-specific setting, Sato should be (re)trained accordingly to be able to deal with such organization-specific semantic domains. This does require a very large set of labeled training data, which is likely not readily available at most organizations. However, it might be worthwhile exploring this option, as Sato should work on all semantic domains, not only those that are in natural language.

4.2.4 Inclusion dependencies

Inclusion dependency detection is used in tasks such as foreign key discovery. An inclusion dependency is defined for any column A of a relation R and column B for relation S, stating that each value in A also appears in B (Abedjan et al., 2015). This is formally denoted as $R.A \subseteq S.B$ or $A \subseteq B$. The same property applies to subsets of columns X of relation R and Y of relation S, denoted as $R.X \subseteq S.Y$ or $X \subseteq Y$, stating that each combination of values in X also appears in Y.

Inclusion dependency detection can be divided into several special cases: *unary*, *n-ary*, *partial/approximate* and *conditional* inclusion dependencies. The former two cases focus on the amount of columns on each side of the dependency, one or n respectively. According to (Shaabani et al., 2017), all algorithms for finding n-ary dependencies require the computation of all unary dependencies first. Furthermore, the worst case time complexity of any dependency detection algorithm is exponential in terms of the number of attributes (Abedjan et al., 2015). Hence, solving the n-nary dependency problem is computationally more expensive than solving solely the unary dependency problem. Tschirschnitz et al. have proposed an algorithm to efficiently detect unary inclusion dependencies on a lot of data in (Tschirschnitz et al., 2017). Similar efforts have been made for n-ary inclusion dependencies in (Shaabani et al., 2017), exploiting the characteristics of single-attribute inclusion dependency detection.

The latter-mentioned three special cases are non-exact special cases. That is, there is no guarantee that they hold for all rows in a data instance. Partial dependencies hold for a fraction (e.g. 90%) of the rows. If such a dependency is found, it might be worthwhile to evaluate whether this dependency holds for a certain condition, e.g. for all rows where the creation date is after the year 2000. If that is the case, then this dependency is also considered conditional. Approximate dependencies, on the other hand, are non-conditional dependencies that are not guaranteed to hold for all rows in a data instance. Rather, they hold for a sample of the data instance. In use cases where being exact is important, approximate dependencies should be avoided. Similarly, partial and conditional dependencies should only be used in use cases where their characteristics are desired.

4.3 Conclusion

From the similar research, reviewed in section 4.1, it can be concluded that there is little to no published research that connects well to entity type recognition as identified in chapter 2. This indicates that the solution that will be proposed in chapter 5 should explore different approaches than the database reverse engineering and entity type recognition solutions proposed in the reviewed literature. Semantic graphs, as used in the literature covered in subsection 4.1.2, are still an option for future research, but their requirements cannot be met during this thesis' time frame.

Because of these limitations, there is a need for different approaches to summarize the data used within organizations. In section 4.2, data profiling was identified as a promising method to enable entity type recognition. Especially semantic domain classification (subsection 4.2.3) could prove valuable for this task, once high quality training data can be provided for the organization-specific semantic domains. For the time being, cardinalities, patterns and data types will be used to recognize similarities in representations of conceptual entity types. Rule three defined in Table 2.3 enables representations of conceptual entity types to be spread over multiple tables. As such, inclusion dependencies will be used to connect rows

in different tables that likely belong together. How such data profiles are used for their intended purposes will be elaborated on in chapter 5.

An overview of the acquired literature, and how it was acquired, can be found in Table 4.1.

TABLE 4.1: An overview of how literature was acquired. Cursive phrases indicate search queries used to browse Google Scholar.

§	Topic	Literature	How acquired?
	database reverse engineering	(Andersson, 1994) (Chiang et al., 1994)	*reverse engineering database*
		(Malpani et al., 2010)	recommended by supervisor
4.1	functional dependencies	(Papenbrock et al., 2015) (Hai et al., 2019)	*functional dependency discovery*
	entity type recognition	(Sleeman et al., 2015) (Giunchiglia et al., 2020)	*"entity type recognition" -omics -biology -biomedical*
	data profiling	(Abedjan et al., 2015) (Naumann, 2014)	*data profiling*
	cardinality estimation	(Harmouch et al., 2017)	F. Naumann's profile
	value patterns	(Fernau, 2009)	*learning regular expressions*
4.2	semantic domain classification	(Khalid et al., 2019)	Springer website
		(Koehler et al., 2017)	IEEE website
		(Zhang et al., 2019)	*semantic type detection*
	inclusion dependencies	(Tschirschnitz et al., 2017) (Shaabani et al., 2017)	*inclusion dependency detection*

Chapter 5

Proposed Solution

This chapter will propose a solution to the entity type recognition problem identified in chapter 2. From section 4.1 can be concluded that there is little published similar research, and that there is thus need for a new approach toward solving this problem. The proposed solution will therefore utilize data profiling techniques covered in section 4.2 to perform entity type recognition.

This chapter is structured as follows. First, the general idea behind the solution will be described in section 5.1. This section divides the solution into two phases, which will be elaborated on in detail in section 5.2 and section 5.3. And finally, a comprehensive summary of the two phases will be provided in section 5.4.

5.1 The solution in general

The solution proposed in this chapter should solve the two problems described in subsection 2.2.1 and subsection 2.2.2 in a given organization. That is, the solution should recognize which conceptual entity types are represented in any given data instance, and which of the rules listed in Table 2.3 was used to get to each representation. The following assumptions were made to ensure that solving these problems remains a feasible task:

1. The organization knows which conceptual entity types might be encountered in their data and are interested in recognizing only those entity types.
2. Each data instance adheres to one of the rules listed in Table 2.3. The rules are covered in detail in subsection 2.1.1.
3. The organization provides example data instances for each of their entity types' distinct representations. These data instances are labeled with their respective entity type(s) (training data).
4. All entity types are defined in one conceptual schema, no entity type has more than one conceptual specification.

The first assumption is made to narrow down the problem space, and thus to ensure that the solution expects to only find entity types used within the organization. Assumption two restricts the solution from recognizing special cases that are not commonly used within the organization. Assumption three helps the solution to recognize relevant entity types by providing ground truth. And assumption four ensures that representations of the same entity types are always conceptually compatible.

The proposed solution consists of two phases, as depicted in Figure 5.1, which will now be introduced and then covered in more detail in section 5.2 and section 5.3.

1. **Clustering & Data Profiling:** Cluster rows in data instances based on provided labels or similarity and create *data profiles* for each cluster.
2. **Entity Type Recognition:** Use the acquired clusters and their corresponding data profiles to recognize similar rows in previously unseen data instances.

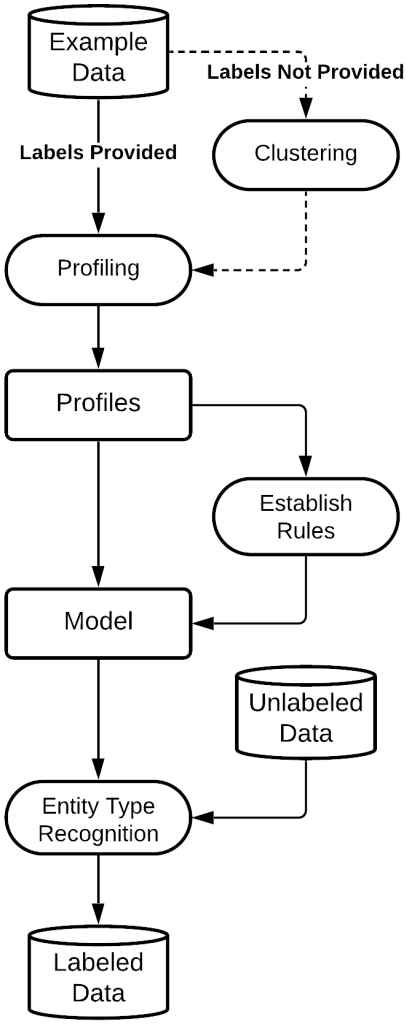


FIGURE 5.1: The method flowchart for the proposed solution.

5.2 Clustering & data profiling

Before entity type recognition can be performed, a ground truth has to be established. The organization will therefore provide example data instances for each of their entity types' distinct representations. This data will be used as example data for phase two. But to do so, similar representations must be clustered together, and data profiles should be extracted for each cluster. In that way, rows in previously unseen data instances can be matched to the acquired data profiles in phase two.

The input for phase one is thus a collection of data instances. Ideally, the rows in these data instances have been labeled with the conceptual entity types they represent. However, providing entity type labels can be deferred to later in the process, resulting in the two following different approaches.

Labels provided: The rows in the given example data instances are clustered based on their provided labels.

Labels not provided: The rows in the given example data instances are clustered based on the methods described in subsection 5.2.1. The acquired clusters are assigned a generated label, but must later be validated and labeled by the user.

It can be concluded that working with labeled input data instances is the most straight forward; cluster as specified and extract data profiles. Working with unlabeled input data is more difficult because it requires the clustering algorithm to determine how rows in each data instance should be clustered. Thus, before covering the data profiling specifications in subsection 5.2.2, subsection 5.2.1 will elaborate on how to deal with unlabeled example data.

5.2.1 Clustering methods for unlabeled example data

To maximize the ease of explainability of the employed clustering method, the proposed solution will incorporate its own custom clustering algorithm based on binary representations of the data. This method is easily explainable and should work given that the rows in unlabeled data instances each adhere to one of the rules stated in Table 2.3, as per assumption two in section 5.1. This means that the solution can use this knowledge as a means to perform the clustering of rows inside each example data instance. Each of the rules requires its own approach, as will now be covered in detail.

1. The table contains data on multiple subtypes from some entity type in its taxonomy.

In this case, the table contains data that can represent any entity type in one taxonomy. For the taxonomy illustrated in Figure 2.1, for example, that would mean that each row in the table can represent a different type of vehicle (e.g. compact, SUV, etc.). An example of such a table is given in Table 5.1.

If no labels have been provided to indicate which row represents which type in the taxonomy, the algorithm should figure out how to cluster the rows in this table itself. That means that the algorithm should make distinctions between the different representations. One way of doing so would be to represent each row in the example data as a binary vector; a zero or one indicates an empty or populated cell respectively. There are several other ways of doing this, however, this is theoretically a cheap solution to finding distinct representations in one table, as it requires only one bit of information to be stored per cell in the data. Furthermore, binary representations can be saved without causing privacy concerns, as there is no real data involved. There are drawbacks to this method, however, as will become evident shortly. The binary representation of Table 5.1 is shown in Table 5.2.

When analyzing both tables in this manner, it becomes clear that not every row populates every column. This indicates that rows that populate different columns

TABLE 5.1: An example data instance that adheres to the first rule.

idx	P_1	P_2	P_3	P_4	P_5	P_6	...	P_{k-1}	P_k
0	41230	FALSE	AUTO28JB61	35216			...		
1	52882	FALSE	AUTO28JB33	52904			...		
2	76508	FALSE	AUTO28JB31		88cPigT	TRUE	...		
3	12614	TRUE	AUTO28JB15		88cSlothY	TRUE	...		
...	...	...	...	...	...	...	...	...	...
n-1	26578	FALSE	AUTO28JB42				...	11-05-20	
n	38020	FALSE	AUTO28JB23				...	11-05-21	69199

TABLE 5.2: The binary representation of Table 5.1.

idx	P_1	P_2	P_3	P_4	P_5	P_6	...	P_{k-1}	P_k
1	1	1	1	1	0	0	...	0	0
1	1	1	1	1	0	0	...	0	0
1	1	1	1	0	1	1	...	0	0
1	1	1	1	0	1	1	...	0	0
...	...	...	...	...	...	...	...	...	...
1	1	1	1	0	0	0	...	1	0
1	1	1	1	0	0	0	...	1	1

likely represent different entity types. However, this only holds under the assumption that columns in a table do not overlap semantically. For instance, a single column cannot represent a creation date in some rows, and an address in other rows. Tables where column overlap does occur should be labeled before they are used in this solution, or another clustering method can be used.

In conclusion, when Table 5.1 is used as example input, it can be seen that P_1, P_2 and P_3 are shared among all different entity type representations in this table. But the distinction between entity type representations can be made based on whether P_4; P_5 and P_6; or P_{k-1} and/or P_k have been populated (P_k is nullable in this example). The resulting clusters are listed in Table 5.3.

TABLE 5.3: Entity type representation clusters for Table 5.1.

Cluster:	Row indices
T_1	[0, 1, ...]
T_2	[2, 3, ...]
T_3	[n-1, n, ...]

2. The table contains data on a type at the bottom of its taxonomy, and only that type.

In this case, binary representations are not of much use. Each cell in each row will be populated, unless dealing with nullable columns. The solution should give a single cluster for tables that adhere to this rule.

3. The rows in the table contain data on a combination of multiple entity types, or part thereof, possibly from different taxonomies.

When this rule applies to a table, there are two special cases that arise and should be handled differently. The special cases are as follows:

1. All of the entity type's attributes included by the logical schema are present in the same row.

2. Some rows in the table may represent only part of one or more entity types and refer to rows in other tables for the rest of their attributes.

Tables that fall in the category of the first special case can be handled using the same approach as for rule one or two. The only difference is that the labels for the acquired clusters should reflect that each cluster represents more than one entity type.

For the second special case, however, it is also important to establish links between the rows that refer to each other because they represent entity types together. In Table 5.4, an example of such a special case is given. The last column in Table 5.4a represents a foreign key relationship with Table 5.4b. Finding foreign key relationships in the example data instances can be done through the inclusion dependency algorithm mentioned in subsection 4.2.4 that best fits the use case. For instance, approximate inclusion dependency detection can be employed to speed up this process, at the sacrifice of accuracy. The implementation of this proposed solution will include only the detection of unary inclusion dependencies to facilitate precise results for a wide array of use cases.

TABLE 5.4: Example data instances that adhere to the third rule.

(A) Table one, FK_4 refers to table two's index.

idx	P_1	P_2	P_3	FK_4
0	76508	FALSE	AUTO28JB31	0
1	12614	TRUE	AUTO28JB15	1
2	41230	FALSE	AUTO28JB61	2
3	52882	FALSE	AUTO28JB33	3
...	...	...	...	...
n-1	26578	FALSE	AUTO28JB42	m-1
n	38020	FALSE	AUTO28JB23	m

(B) Table two.

idx	P_5	P_6
0	88cPigT	TRUE
1	88cSlothY	TRUE
2	88cCodA	FALSE
3	88cGulZ	TRUE
...	...	...
m-1	88cBisonX	FALSE
m	88cDuckK	FALSE

Clusters that were extracted from different tables but represent the same entity types similarly should be merged, for example if there would be two different instances of Table 5.1. Binary representations cannot be used to do this. Rather, the data profiles described in subsection 5.2.2 should be compared, and their corresponding clusters merged accordingly if there is a sufficient match. The merging of clusters can be done automatically, but should ask for user verification if desired.

5.2.2 Specification of the data profiles

After acquiring the clusters for each example data instance, these clusters' characteristics should be summarized in data profiles. The data profiles in this solution will consists of column data type specifications and several cardinalities. The benefit of using data profiles is that they require a minimal amount of example data and give a humanly comprehensible overview of the data's characteristics, without storing any real data. Furthermore, they can be used for multiple other applications than entity

type recognition and can be substituted by other data profiles if desired—as long as the rest of the process (e.g. classification rules) is adapted accordingly. Table 5.5 lists all the data profile attributes used in this solution, and section 5.3 elaborates on how the data profiles will be used for entity type recognition.

Some of these attributes, such as the number of string columns, represent explicit characteristics of the rows in each cluster, and can be acquired with as little as one example row of data. They are expected to be effective characteristics for entity type recognition and require cheap matching rules in terms of time complexity. For example, if a given row comprises twenty attributes but a given data profile specifies fifty attributes, the two are likely not a match and the profile can be eliminated from the matching process. These attributes are expected to eliminate most data profiles as possible matches.

Other attributes represent implicit characteristics and require more expensive measures to be worth including for entity type recognition. The degree of unique values, for example, is only useful if all the values in its corresponding column are

TABLE 5.5: Data profile attributes used in this solution, with their respective descriptions. The thick line separates explicit from implicit attributes.

Attribute	Description
num_attr	Specifies the number of attributes (columns) the rows in the cluster populate.
num_nullable	Specifies the number of attributes that are nullable.
nullable_attr	A binary vector where zero and one indicate a non-nullable and nullable attribute respectively.
num_str	Specifies the number or string attributes.
num_int	Specifies the number of integer attributes.
num_bool	Specifies the number of boolean attributes.
num_float	Specifies the number of float attributes.
num_datetime	Specifies the number of datetime attributes.
num_timedelta	Specifies the number of timedelta attributes.
num_object	Specifies the number of attributes that could be classified as one of the types above.
attr_types	A vector of all attribute data types in left-to-right order.
unique	A vector representing the degree of unique values for each attribute in left-to-right order.
range	A vector that specifies the range for each numeric attribute, null value otherwise in left-to-right order.
mean_std	A vector that specifies the mean and standard deviation for the numeric value attributes, null value otherwise, in left-to-right order.
len_range	A vector that specifies the range of string length for string attributes, null value otherwise, in left-to-right order.
common_string	A vector that indicates whether all values of a string attribute contain a certain substring, null value otherwise, in left-to-right order. If a common substring is found, its hashed value is stored in the vector for comparison during entity type recognition.

TABLE 5.6: Example data profiles. Rows, columns and values have been truncated for brevity.

cluster	#_attr	#_str	#_int	#_bool	...	mean_std
T_1	16	7	5	2	...	`[..., [26452, 6123.42]]`
T_2	77	42	21	9	...	`[[125746, 3214.11], ...]`
...	...	...	...	...	...	...
T_n	15	6	5	2	...	`[..., [34600, 807.05]]`

stored somehow, preferably in a way that eliminates privacy concerns. These attributes are expected to eliminate data profiles that have similar explicit attributes but should not match, and should be evaluated after the explicit attributes because they require more resources.

As stated before, the attributes in the data profile can be substituted for other attributes if desired. For example, if you know that your data has very specific characteristics that can be used for entity type recognition, these can be included as an attribute. The ones in Table 5.5 have been chosen based on personal thought process with regard to *how an algorithm could distinguish one representation from another without using semantic comparison*, and are expected to be helpful in general. As of yet, there is no evidence that these attributes will successfully help perform entity type recognition. Their effectiveness will be evaluated in section 6.3.

All example data can be omitted once these data profiles have been acquired, eliminating privacy concerns when dealing with sensitive data. The data profiles can be saved to a file so that they do not have to be acquired again in the future, or to be used on another system. How data profiles, such as those in Table 5.6, are used for entity type recognition will be elaborated on in the next section.

5.3 Entity type recognition

The data profiles that are acquired during the first phase will be used as ground truth in the second phase to perform entity type recognition. The general idea of phase two is to match each row in each input data instance to one such data profile. For example, the solution should recognize which data profile in Table 5.6 (T_1, T_2 or T_n) best resembles the each row in an input data instance such as Table 5.1. And as such, each input row is assigned an entity type label that states which conceptual entity types are represented.

This section continues by describing how the proposed solution recognizes entity types in subsection 5.3.1. Then, in subsection 5.3.2, the output of the proposed solution will be discussed, as well as how the output serves a solution to the problems described in chapter 2.

5.3.1 Rule-based classification

The entity type recognition process will be fulfilled by a rule-based classification algorithm. A rule-based approach is chosen to maximize the process's transparency and the ease of explainability of results, as this allows for easy evaluation of each individual rule. The algorithm takes a single row, a table or a collection of tables as input and evaluates rules to match each input row to the previously acquired data profile that matches best. To maximize the versatility of the algorithm, the data profile attributes listed in Table 5.5 can be substituted for any attributes that the user

desires to work with. The same analogously applies to the set of rules used for the classification of entity type representations. Therefore, the algorithm allows for the use of user-defined rules, but only rules that apply to the attributes mentioned in this thesis will be covered in this section.

The algorithm evaluates rules in a specified order and allows for strict evaluation, meaning that if a data profile does not match an input row for a given rule, the algorithm can disregard that data profile during evaluation of the following rules. This should allow for some speed up of the entity type recognition process, but might not be desirable in some situations—hence its optionality. The rules included in this thesis are the following:

1. **Evaluate the number of attributes:** match the number of attributes in an input row to each data profile. This number should be in the range of $[\texttt{num\_attr} - \texttt{num\_nullable}, \texttt{num\_attr}]$ for each data profile, considering possible nullable attributes.

2. **Evaluate the number of data types:** for each attribute data type, match the number of attributes of that data type in the input row to the corresponding attribute in the data profile. For example, if an input row contains 7 string attributes, a matching data profile's `num_str` should be 7. In case $\texttt{num\_nullable} = a \neq 0$ in a given data profile, allow for the input row to have at most a less attributes of a data type across all attribute data types.

3. **Evaluate the column data types:** evaluates the order of attribute data types as specified in each data profile in `attr_types`. In case $\texttt{num\_nullable} = a \neq 0$ in a given data profile, evaluate at most a attributes to the right in the input row (in increments of 1).

4. **Evaluate the degree of unique values:** for each value in a given data profile's `unique` attribute that is (nearly) 1, evaluate whether the value for the corresponding attribute in the input row has not yet been seen. This requires an efficient and privacy-respecting approach to save all values of the corresponding attribute in the example data during phase one.

5. **Evaluate the range of numeric values:** for each numeric attribute in an input row, evaluate whether it falls in the range of the corresponding value in a given data profile's `range` attribute. This rule should again consider $\texttt{num\_nullable} = a$ if $a \neq 0$, as its evaluation might be meaningless otherwise—for example when trying to evaluate a numeric range on a string value.

6. **Evaluate the mean and standard deviation of numeric values:** for each numeric attribute in an input row, evaluate whether its value falls within two (or three) standard deviations of the mean value in a given data profile's `mean_std` attribute. This rule should only be applied to attributes of which the values follow a normal distribution, as 95% (or 99.7%) of the values fall within two (or three) standard deviations of the mean in that case[1]. This rule should again consider $\texttt{num\_nullable} = a$ if $a \neq 0$, as its evaluation might be meaningless otherwise.

7. **Evaluate the length of string values:** for each string attribute in an input row, evaluate whether its length falls in the range of the corresponding value in

[1] This is called the 68–95–99.7 rule.

a given data profile's `len_range` attribute. This rule is especially meaningful in cases where the minimum and maximum of the range are equal, i.e. where the string should be of an exact length. This rule should again consider `num_nullable` $= a$ if $a \neq 0$, as its evaluation might be meaningless otherwise.

8. **Evaluate the presence of common substrings:** for each string attribute in an input row, evaluate whether it contains the corresponding value of a given data profile's `common_string`. This rule should again consider `num_nullable` $= a$ if $a \neq 0$, as its evaluation might be meaningless otherwise.

These rules, were established through personal thought process with the aim to effectively exploit the attributes provided in the data profiles. This is also why they have to be evaluated in a particular order. The first three rules are computationally cheap and straightforward to evaluate and are expected to eliminate the vast majority of data profiles; those that should obviously not be considered as a match. As such, the remaining, computationally more expensive rules can be evaluated for a limited number of data profiles to uncover implicit, more detailed similarities between given rows and data profiles. Again, there is not yet evidence that these rules will live up to their expectations. This will also be evaluated in section 6.3.

5.3.2 Output & interpretation

Once all the rules have been evaluated for a given row, the algorithm provides an assessment of which data profile best resembles the input row. This assessment consists of a score on a range of $[0, 1]$ for each data profile. For example, running the row in Table 5.7a through the algorithm results in the scores in Table 5.7b. It becomes obvious that T_n is the best matching profile for the given input row, and as such the input row is classified as a representation of the conceptual type(s) that T_n represents. Unlike in the entity type recognition method proposed in (Sleeman et al., 2015), this solution does not provide "unknown" labels because it should not have to, as per assumption one in section 5.1. For any input row, the classification process can result in one of the two following special cases:

1. **There is a clear best-matching data profile (like in Table 5.7).** In this case, the algorithm can go ahead and assign an entity type label to the input row.

2. **Multiple data profiles have acquired an (almost) equal score and there is no data profile with a higher score.** In this case, the algorithm does not know which entity type label to assign. As such, it should present the highest scoring data profiles as candidates to the user and ask the user to make the final decision on the entity type label.

Once each row has been classified as a representation of specific conceptual entity types, these results can be aggregated for each input data instance to solve problem P1 and P2 as described in section 2.2. Such an aggregation should be an overview of the distinct entity type labels for each input data instance. An example of such an aggregation is shown in Table 5.8.

Problem P1, *"given unlabeled data instances and the data instances labeled with their respective entity type(s), recognize which of the known entity types are likely stored within each unlabeled data instance,"* has now been solved; each initially unlabeled data instance has now received entity type labels.

Problem P2, *"given the result of P1, recognize the rules used to map the conceptual schema to the logical schema, for each logical schema,"* can be solved by analyzing the

TABLE 5.7: An example input row and the result of running it through the entity type recognition algorithm.

(A) The input row.

idx	P_1	P_2	P_3	P_4	P_5	P_6	...	P_{14}	P_{15}
0	41230	FALSE	AUTO28JB61	35216	88cPigT	TRUE	...		1

(B) The evaluation scores for the input row for each data profile listed in Table 5.6.

idx	T_1	T_2	...	T_n
0	0.78	0	...	**0.924**

entity type labels assigned to each initially unlabeled data instance. Each rule in Table 2.3 can be identified as follows, in the same order:

1. The data instance has received more than one entity type label, but none of these contain more than one conceptual entity type.
2. The data instance has received one entity type label containing exactly one conceptual entity type.
3. The data instance has received one or more entity type labels, of which at least one contains more than one conceptual entity type.

TABLE 5.8: Example aggregated results. The data instances each adhere to, in similar order, one rule in Table 2.3.

Data instance id	clusters	entity types
1	T_1, T_2	[[vehicle.car.compact], [vehicle.car.SUV]]
2	T_1	[vehicle.car.compact]
3	T_n	[[vehicle.car.compact, engine.gasoline, tires.winter.car]]

5.4 Conclusion

The proposed solution performs entity type recognition in two phases. First, example data is clustered and analyzed to obtain comprehensive data profiles. These data profiles are then used in phase two to classify previously unseen data instances in a rule-based manner, and assign entity type labels accordingly. The output of the solution gives a comprehensive overview of which conceptual entity types have been found in each data instance, such as in Table 5.8 for example. Analyzing the assigned entity type labels enables the identification of which rules have been used to map the conceptual schema to each logical schema. And as such, the problems stated in chapter 2 have been solved.

The data profiles and the corresponding rules covered in this chapter are expected to be suitable means for database entity type recognition in cases where the data profiles substantially point out diversity between entity type representations. Furthermore, in scenarios where the explicit characteristics of entity type representations are mostly similar, the provided rules may be ineffective and their evaluation slow. For example, if multiple entity type representations have the same number of

attributes as well as similar numbers of attribute types, this set of rules might not be successfully applicable. Therefore, it must be noted that data profiles and their corresponding classification rules are likely best evaluated for each individual use case where the proposed method is employed.

Chapter 6

Results & Evaluation

The entity type recognition method proposed in chapter 5 will be evaluated through performing fitting experiments on its implementation in Python. The used experimental setup will be described in section 6.1. Then, some results of the clustering and data profiling process (section 5.2) will be presented in section 6.2 to evaluate its viability. The classification of entity types (section 5.3) is experimented with and evaluated in section 6.3, after which the results will be summarized and discussed in section 6.4.

6.1 Experimental setup

This thesis is conducted in cooperation with a bank, which dictates some rules for experimentation. First of all, experiments are not conducted on real data that contains any client's information. Second, any real data may only reside or be experimented with in the bank's secure environment. These two restrictions make evaluation on real data difficult, though not impossible, because it limits the data available to perform evaluation on. The bank has therefore provided a data generator that outputs data of any desired size, containing no real data but resembling the it well. Thus, the data used for experimentation are:

- Generated data in varying sizes. The size is defined by the number of data instances, rows per data instance and different entity type representations per data instance. Furthermore, the generator provides the logical schema for each generated data instance, which serves as ground truth. The size of the generated data is specified for each experiment in the following sections. What can be expected of the data generator will be elaborated on in subsection 6.1.1.
- About 170,000 rows of real data sampled from nine tables, for which the ground truth is provided by an expert at the bank. Experiments on real data will be used to determine whether the experiments on generated data are of any value.

Furthermore, the metrics used for evaluating the experimentation are provided in Table 6.1. Accuracy_r is the strictest used metric aimed at uncovering how well the method works for each individual row. Accuarcy_D however, evaluates how well the method solves P1 in subsection 2.2.1, as P1 aims at recognizing which conceptual entity types are represented in each data instance rather than each row. False positives and negatives are measured to support Accuarcy_D. Finally, rule efficacy evaluates how much influence each rule introduced in subsection 5.3.1 has on the outcome of each experiment. This metric can be used to evaluate whether some rules should be replaced in the future, but also indirectly evaluates whether the attributes listed in Table 5.5 are at all useful.

TABLE 6.1: The metrics used for evaluation of the method.

Metric	Definition
Accuracy_r	The degree of rows for which entity type labels were predicted exactly correctly ($9/10 \rightarrow 90\%$).
Accuracy_D	The degree of data instances for which distinct entity type labels were predicted exactly correctly ($9/10 \rightarrow 90\%$).
False positive	Occurs when a data instance received an entity type labels of a representation that it does not contain.
False negative	Occurs when a data instance did not receive an entity label of a representation that it does contain.
Rule efficacy	The absolute percentage of data profiles eliminated as match candidate, measured per rule.

6.1.1 The data generator

The data generator that will be used for experimentation is developed by an expert at ING. Its purpose is to generate relational data of desired any size without sparking privacy or integrity concerns. This means that the resulting relational data contains no real information and can thus be used outside of the bank's secure environment (on faster machines). Furthermore, it also means that the relational data can be shaped to evaluate the proposed solution under specific conditions (e.g. five versus hundreds of attributes per entity type). The individual advantages of using generated relational data for the majority of the experiments are as follows.

- It provides relational data of any desired size in terms of entity type taxonomies, entity types, tables and rows.
- It allows for the specification of characteristics of the relational data, such as the range of number of attributes per entity type, the range of rows per table, and maximum number of specific data type attributes per entity type.
- It guarantees reproducibility under fixed parameters. That is, the data generator will provide the same relational data if and only if all specified characteristics are the same.
- It allows for use of the relational data on any system.

Under the user-specified parameters, the generator will start off by generating the conceptual schema. This specifies the entity type taxonomies and attributes per entity type. It then generates a random amount of entities for each entity type, populating each attribute of each entity with a value. These values are either random or follow a pattern, depending on the conceptual schema. Finally, the data generator outputs the generated relational data in three ways:

1. As combined as possible. Each table contains data on all entity types of the same taxonomy (e.g. Table 5.1).
2. Separated complete entity types. Each table contains data on one entity type, including the attributes that are specified in parent entity types.
3. As separated as possible. Each table contains data on the attributes of one entity type, but refers to attributes of parent entity types in other tables.

These three special cases are relatively similar to the special cases mentioned in Table 2.3. Only rule three in that table is not covered completely. However, as was determined in subsection 5.2.1, tables that adhere to that rule can be clustered similarly to those that adhere to rule one. In conclusion, there should be no problem in covering all the rules specified in Table 2.3 during experimentation.

6.2 Clustering results

This section will evaluate whether the clustering methods proposed in subsection 5.2.1 deliver their intended results. That is, in case example data is provided without entity type labels, these methods should identify exactly one cluster per distinct entity type representation in a table, and each row in that table should be included in the cluster that correctly reflects the entity type(s) it represents. To do so, clustering results will be presented for two generated data instances. One of these data instances adheres to rule one, and the other to rule two listed in Table 2.3. As data instances that adhere to rule three can be handled the same as those adhering to rule one, this special case will not be evaluated separately. Finally, the method is also evaluated on real data.

6.2.1 All entity (sub)types from the same taxonomy in the same table

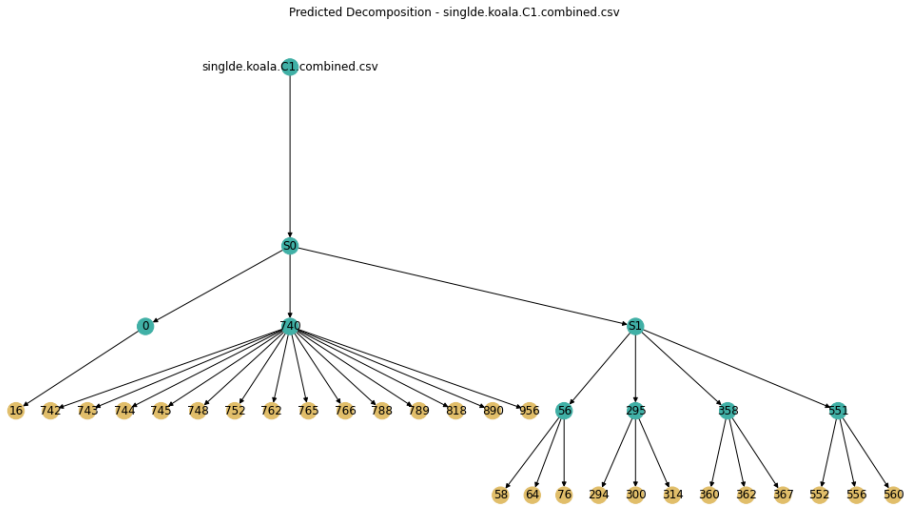


FIGURE 6.1: The result of clustering a data instance that adheres to rule one defined in Table 2.3.

The classification in Figure 6.1 displays the result of clustering entity type representations in a table that contains data on several entity types from the same taxonomy. The figure represents a classification of the entity type representations where:

- **Turquoise vertices (gray for people with protanopia, deuteranopia)** represent clusters, each representing their own entity type(s).
- **Cream vertices (pink for people with tritanopia)** represent clusters that represent the same entity type as their parent vertex, but differ slightly in representation because of null values.
- **Vertex names** can be substituted by real entity type labels, but:

  - **Numeric values** indicate the first index in the data instance where this entity type representation was encountered.
  - **String values** indicate a parent entity type representation (e.g. car $\rightarrow$ {compact, SUV}).

- **Vertices with the same parent** are subtypes of the same entity type.

The results were verified by matching the clusters to the logical schema provided by the data generator. Issues arose when a table contained entity type representations with nullable attributes, but with no single row that populated all its corresponding attributes. In other words, for each entity type representation in a table, for each corresponding row, at least one nullable attribute is null. This phenomenon caused the clustering implementation to consider each such case to be its own entity type representation, resulting in separate clusters for rows that actually represented the same entity types. Apart from that, the method worked well. It can thus be concluded that this method for clustering works on data instances that represent multiple entity type of the same taxonomy, or where each row in the data instance can represent part of, one or multiple entity types, but requires improvement.

6.2.2 One entity type represented in a table

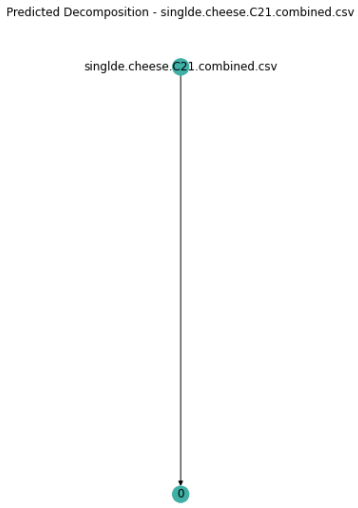


FIGURE 6.2: The result of clustering a data instance that adheres to rule two defined in Table 2.3.

The classification in Figure 6.2 displays the result of clustering entity type representations in a table that contains data on one entity type. The figure represents a classification of the entity type representation similarly to the one presented in subsection 6.2.1. The implementation experienced the same issues as with the previous case, but worked well in most experiments performed on generated data.

6.2.3 Discrepancies with real data

Although the generated data should resemble the real data at the bank well, there were some differences between these two that caused the clustering methods to perform suboptimally. The first issue that arises is that the clustering method separates

the real data into too many clusters. In other words, it detects more different entity type representations than are actually present in each data instance. This likely has to do with the same null-value issue as identified while experimenting with generated data; the algorithm gets confused if there is no single row of an entity type representation that populates all of its attributes, including nullable attributes. Furthermore, the algorithm also started making mistakes if one of the columns in a table was completely null. In the future, this requires more careful handling of null values in each data instance or can be avoided completely if entity type labels are provided beforehand. How this discrepancy influences the results of phase two of the method will be covered in section 6.3.

One difference between the generated and the real data, is that the real data contained columns that were always null. Such cases are especially tricky because there is no guarantee that the attribute of an incoming row of the same entity type is also null. As such, this could cause erroneous classification when evaluating some of the implemented rules (e.g. the number of attributes). Future implementations should consider this issue.

6.3 Classification experiments & results

This section will first cover a wide variety of experiments on generated data, after which experiments on the real data will be performed. The generated data will be different for each experiment, as for each experiment the generator will:

- generate at least one table in accordance with each logical schema design rule specified in Table 2.3.
- generate a different amount of entity type taxonomies.
- apply different maximum taxonomy widths and depths, resulting in a larger number of entity types per taxonomy as well as a larger possible number of subtypes per entity type.
- apply different ranges of attributes per entity type, resulting in different ranges of diversity between entity types in terms of the number of attributes as well as the combinations of attribute data types. The diversity will be expressed in terms or the range, mean and standard deviation of the number of attributes per entity type. Justifiably so because each attribute can be of one of seven possible data types, resulting in 7^n different possible combinations of attributes, where n is the amount of attributes of a given entity type.

All experiments, except for the one on real data, will be performed using two ratios of example to test data, namely nine-to-one and one-to-nine. This will be done to evaluate whether there are notable differences in performance when little example data is available, as this best represents a realistic scenario. This also means that there is more room for error because there are considerably more rows to be classified.

It must be noted that, although this was initially the intention, the experiments will not be performed on more than 100,000 generated rows of data. This is due to issues with the current version of the data generator that disabled it from generating such amounts of data.

6.3.1 Experiment 1: relatively little diversity

TABLE 6.2: The data characteristics for the first experiment.

Characteristic:	Value:
Generated or real data	generated
Total number of tables	50
Total number of entity types	71
Maximum number of entity types in data instance	11
range, avg. and std. of number of attributes per type	[2, 18] | 8.38 | 4.58
Total number of rows	16,460
Example to test ratio	9:1 - 1:9

The first experiment will be performed on a small amount of data with relatively little diversity. That is, entity types can have between two and eighteen attributes, with an average of eight and a standard deviation of about five. There are still many combinations of attribute types possible, but a lot less than for most of the upcoming experiments. It also means that there is a relatively small presence of nullable columns and that are relatively many entity types with (about) the same amount of attributes in total. The characteristics of the used data are covered in Table 6.2.

The goal of this experiment is to evaluate whether the proposed solution works well in a situation with some diversity, but with a low probability of having very similar entity type representations. This offers possibilities of evaluating whether the rules for this solution are chosen sensibly under conditions where it is unlikely that two distinct entity type representations look very similar.

TABLE 6.3: The resulting metrics of the first experiment.

Metric:	9:1	1:9
Accuracy_r	93.86%	91.40%
Accuracy_D	74.0%	20.0%
False positive	12(66.67%)	79(95.18%)
False negative	6(33.33%)	4(4.84%)
Rule efficacy (%)	83.09 | 5.27 | 5.42 | 0.79 | 0.02 | 1.95 | 0.0	

The results of this experiment, provided in Table 6.3, cannot be marked promising or unfavorable at first sight. It is clear that the nine to one example to test data experiment performed well: the model scored high overall accuracy with seemingly little false positives and negatives. The one to nine experiment scored poorly on the data instance accuracy metric. However, this does not immediately indicate a poor overall result, as this is almost entirely the case because of false positives. In fact, the number of false negatives has decreased in comparison to the experiment with less test data[1]. In conclusion, this means that the the value of using the proposed method in this case depends on the use case. The results of this experiment indicate that the proposed solution can be useful in cases where the classification of individual rows is more important than the prediction accuracy for data instances, or in cases where false positives are no issue.

[1]The decrease in false negatives, with regard to the experiment with less test data, has occurred because of the difference in data samples between the two experiments.

The measured rule efficacy indicates that the classification results were mostly thanks to the first three rules, focusing on the number of attributes, the number of attribute data types, and the left-to-right order of attribute data types. These three rules together eliminated on average nearly 95% of the data profiles as possible matches for each row. This indicates that there must have been high diversity between the different data profiles in terms of attributes, something that this experiment was supposed to have relatively little of. In retrospect, this result was to be expected, as the standard deviation of the number of attributes is quite high at this range. Partly therefore, the experiment in subsection 6.3.4 has been introduced, in which the data has a standard deviation and range of the number of attributes that is arguably more suitable to evaluate low attribute diversity performance. Finally, the fact that 95% of the data profiles was eliminated as potential matches after evaluating three rules indicates that evaluating rules in order helps in reducing the time it takes to classify all rows.

6.3.2 Experiment 2: many entity types with relatively little diversity

This experiment is focused on evaluating the method's performance in a scenario with many different entity type representations with relatively little attribute diversity, as covered in Table 6.4. After seeing the results from the previous experiment, it could be argued that the standard deviation of the number of attributes per entity type is still quite high. However, this number is about half of that in the collected real data, as can be seen in Table 6.10, and is therefore still considered relatively small.

TABLE 6.4: The data characteristics for the second experiment.

Characteristic:	**Value:**
Generated or real data	generated
Total number of tables	108
Total number of entity types	282
Maximum number of entity types in data instance	11
range, avg. and std. of number of attributes per type	[2, 28] ∣ 14.13 ∣ 5.97
Total number of rows	58,161
Example to test ratio	9:1 - 1:9

The goal of this experiment is to evaluate if and how the proposed solution would scale with three times as many, slightly more diverse entity types as the previous experiment. Expectations are that the rule efficacy will stay roughly the same as that in the previous experiment, due to the on average $7^{14.13}$ possible different combinations of attributes per entity type. In fact, this metric will likely only show different results in scenarios where the vast majority of entity types have little attributes, with maximum difference in scenarios where all entity types have only one attribute.

Other than the rule efficacy, it is expected that the accuracy for both data instances and rows will go up slightly. This is because of the higher diversity between entity type representations in terms of attributes.

Judging from the results found in Table 6.5, the expectations are mostly satisfied, with one small exception. Namely the data-instance-based accuracy for the experiment with nine times more example test data than test data. This result is likely incidental, as all the other metrics point to better performance in comparison with the previous experiment.

TABLE 6.5: The resulting metrics of the second experiment.

Metric:	9:1	1:9
$\text{Accuracy}_{\text{r}}$	94.11%	93.73%
$\text{Accuracy}_{\text{D}}$	72.22%	30.56%
False positive	41(82.0%)	269(96.75%)
False negative	9(18.0%)	9(3.25%)
Rule efficacy (%)	83.20 | 5.29 | 5.41 | 0.75 | 0.02 | 1.97 | 0.0	

6.3.3 Experiment 3: many entity types with relatively high diversity

The third experiment will have the same number of entity types and tables as the previous, but with a drastic increase in attribute diversity. The entity types in this experiment will have, on average, about ten more attributes and an almost twice as high standard deviation as the entity types in the previous experiment. The characteristics of the generated data can be found in Table 6.6.

TABLE 6.6: The data characteristics for the third experiment.

Characteristic:	Value:
Generated or real data	generated
Total number of tables	108
Total number of entity types	282
Maximum number of entity types in data instance	11
range, avg. and std. of number of attributes per type	[3, 58] | 23.33 | 11.19
Total number of rows	58,161
Example to test ratio	9:1 - 1:9

The goal of this experiment is to evaluate the proposed solution in a scenario with a high diversity in terms of attributes. The expectation is that the rule efficacy will shift even more toward the first three rules due to their strict evaluation of the number of attributes, attribute data types and the order thereof. Furthermore, the accuracy scores will likely be better than those witnessed in the previous experiment, because of the higher diversity but similar size of data instances. The results can be found in Table 6.7.

TABLE 6.7: The resulting metrics of the third experiment.

Metric:	9:1	1:9
$\text{Accuracy}_{\text{r}}$	89.61%	91.07%
$\text{Accuracy}_{\text{D}}$	72.22%	27.78%
False positive	32(65.31%)	200(95.69%)
False negative	17(34.69%)	9(4.31%)
Rule efficacy (%)	86.41 | 7.29 | 4.41 | 0.08 | 0.00 | 0.97 | 0.0	

The resulting rule efficacy holds up to its expectations. However, the accuracy scores indicate that the effect of the rules based on attribute diversity on classification accuracy saturate at some point. That is, employing the proposed solution on data instances with ever-increasing attribute diversity does not necessarily lead to ever-improving results. As a result, it can be argued that, starting at rule four, the rules are not sufficiently effective for the proposed solution.

When comparing the results to those in subsection 6.3.2, it is surprising that the accuracy dropped slightly while the number of false positives and negatives have both improved. However, this merely indicates that misclassifications are more spread out over data instances than before.

6.3.4 Experiment 4: little entity types with very little diversity

This experiment will be performed on data with the smallest attribute diversity among all experiments. The tables in the generated data will each comprise a single entity type with about five attributes on average, as can be seen in Table 6.8.

TABLE 6.8: The data characteristics for the fourth experiment.

Characteristic:	**Value:**
Generated or real data	generated
Total number of tables	42
Total number of entity types	42
Maximum number of entity types in data instance	1
range, avg. and std. of number of attributes per type	$[2, 10]$ ǀ 5.14 ǀ 1.84
Total number of rows	11,773
Example to test ratio	9:1 - 1:9

The results are expected to reflect a lower accuracy than measured before due to the lesser diversity in terms of attributes and the influence the first few rules had in the previous experiments. The rule efficacy will shift slightly to the right, and if it does so sufficiently, the accuracy is expected to still be high because of the increasing level of detail each of the subsequent rules evaluates.

TABLE 6.9: The resulting metrics of the fourth experiment.

Metric:	**9:1**	**1:9**
Accuracy_r	97.49%	94.58%
Accuracy_D	80.95%	21.43%
False positive	8(88.89%)	66(100%)
False negative	1(11.11%)	0(0%)
Rule efficacy (%)	72.93 ǀ 8.81 ǀ 8.27 ǀ 1.29 ǀ 0.03 ǀ 3.9 ǀ 0.01	

From the results in Table 6.9, it can be concluded that the proposed solution also achieves high accuracy in scenarios with little attribute diversity. However, this generated data for this experiment contains a relatively small amount of entity types. In comparison to previous experiments, these results reflect a relatively high amount of false positives per table, but very low false negatives. Thus, the results of this experiment look promising if false positives are of little concern. It is, however, difficult to think of example scenarios where false positives matter so little or rather cause little inconvenience when performing database entity type recognition.

6.3.5 Experiment 5: evaluation on real data

The final experiment will be an evaluation of the proposed solution on real data in tenfold. The real data will be composed of a total of ten tables, containing a total of nineteen different entity types, and will draw different samples from the tables

in every iteration. As can be seen in Table 6.10, the average and standard deviation of the number of attributes are high and indicate that this particular data source at the organization potentially has high attribute diversity among among entity type representations.

TABLE 6.10: The data characteristics for the fifth experiment.

Characteristic:	**Value:**
Generated or real data	real
Total number of tables	9
Total number of entity types	19
Maximum number of entity types in data instance	6
range, avg. and std. of number of attributes per type	$[11, 56]$ ǀ 39.11 ǀ 11.79
Total number of rows	171,179
Example to test ratio	$1:9(\times 10)$

The goal of this experiment is to validate the results of experiments on generated data. That is, results of experiments performed on generated data are only valid if the results achieved by applying the proposed solution on real data are similar. This is because the data generator could have been generating data that is perfectly distinguishable by the proposed solution, while real data might prove to be different. And above that, a solution that does not work on real data is arguably worthless.

The results for this experiment are presented in Table 6.11, though slightly different than for the previous experiments. Instead of performing experiments in two different example to test data ratios, this experiment performed ten iterations with the same one-to-nine ratio on different data samples. This choice was made because setting up experiments in the organization's secure environment takes considerably more time than when doing so on generated data outside of this environment, and can be justified by the consistent performance at this ratio in previous experiments. The results thus represent the range of results from worst to best.

TABLE 6.11: The resulting metrics of the first experiment.

Metric:	**1:9 (worst)**	**1:9 (best)**
$\text{Accuracy}_{\text{r}}$	78.49%	88.78%
$\text{Accuracy}_{\text{D}}$	77.78%	100.0%
False positive	1(50%)	0
False negative	1(50%)	0
Rule efficacy (%)	61.44 ǀ 5.54 ǀ 0.0 ǀ 2.17 ǀ 1.24 ǀ 0.0 ǀ 0.0	

The resulting accuracy ranges between roughly 78 and 88 percent and 77 and 100 percent for rows and data instances respectively. While this is a decrease in performance on individual rows, this is a considerable increase in performance on data instances. It could be argued this result is unsurprising due to the small number of data instances considered. However, the resulting rule efficacy would suggest otherwise. In essence, together with the cumulative sum of the rule efficacy scores of roughly 70 percent, this indicates that entity type representations in each data instance are not at all diverse in terms of attributes. This explains that, while individual row accuracy might have fallen behind slightly, the proposed solution classified rows in data instances as the wrong entity type representation from the correct data instance. Hence the strong results on data instance accuracy.

As these results are quite similar to the results of experiments on generated data, the experiments on generated data can be considered a relevant evaluation of the proposed solution as a concept. However, it is advised that more experiments are ran on a wide variety of real data to specifically scrutinize and tailor-make the employed data profiles and corresponding rules. That is the case because there is likely no one solution that fits all use cases, but the concept of the solution could stay the same.

6.4 Conclusion & Discussion

This sections will summarize the results of the performed experiments and shed light on concerns that arose. There are a few key takeaways that can be summed up as follows.

Individual row accuracy ranged between 78 and 97 percent and seemed to be hardly affected by a decrease in example data size. This is a promising result considering the simplicity of the proposed solution. It should be noted that the experiments on real data failed to reach row accuracy above ninety percent. However, this does not discredit concept of the proposed solution, but rather indicates that data profiles and their corresponding classification rules should be modified to fit the general data characteristics in the organization.

Data instance accuracy was poor on generated data but fine on the small amount of gathered real data. It is difficult to draw conclusions from this result because of the discrepancy in results between experiments on real and generated data. In conclusion, the gathered real data was insufficient in terms of size and attribute diversity to validate the data instance accuracy results.

False positives and negatives show mixed results. The number of false negatives looks promising, but the number of false positives is larger than desired. If the goal is to recognize which conceptual entity types reside in each given data instance, it would be inconvenient to set up the proposed solution on real data and then have to filter out hundreds of false classifications. A minimum could be set on the number of times a conceptual entity type is classified in a data instance before it is considered as actually present. However, this introduces a new potential problem, as this requires a sensible threshold to be established and would cause false negatives in cases where some entity types in a data instance do not meet the threshold.

Rule efficacy shows that ordering the classification rules has the desired effect of skewing the number of data profiles considered as a potential match per rule. As a result, the time complexity is reduced for the overall classification process. However, it also indicates that improvement can be found in modifying the rules for each use case. In fact, the mean and standard deviation, and the common substring attributes and their corresponding rules proved to be hardly to not effective. This is likely because the formermentioned attribute captures a characteristic that is too general, while the lattermentioned captures a characteristic that is too specific. Finally, it should also be noted that changes in the rule order likely change the rule efficacy scores, as each consecutive rule has less candidates to evaluate and reject. This analogously means that how valuable each rules and their corresponding attributes are depends on the order in which the rules are evaluated.

Runtime can likely be improved dramatically. The implementation of the solution for this thesis paid little attention to runtime complexity, with the order of rules being the only measure to improve it. And though it was not included in the individual results, it is worth noting that the implementation seems to classify a minimum of 22 rows per second[2]. Increasing the average number of attributes per entity type from 14 to 23 seemed to have little effect on the runtime. The runtime results on real data cannot be compared because they were run on a different system.

In essence, these takeaways indicate that the proposed solution works well as a concept, but requires tailoring to specific use cases if it is to be employed in organizations. The solution would require proper establishment within the organization, including (preferably labeled) example data input from each team within the organization, if the desired result is to reach as close to 100% accuracy as possible.

The proposed solution was developed through personal thought process after identifying that there was little to no published research available on database entity type recognition (see chapter 4). Coincidentally, the results look very promising considering the simplicity of the solution. And as an added advantage, the classification model is easily explainable with little effort, making it applicable within organizations that wish to radiate integrity. In conclusion, this means that interested organizations should look into tailoring the concept that is presented in this thesis to their needs, and they will likely have a well-explainable automated solution for database entity type recognition, from clustering and profiling through the classification model.

[2]Performance measured in a single-threaded setting, on a UNIX-based machine (2.3GHz 8-core Intel i9, 16GB RAM).

Chapter 7

Conclusion

This thesis aims to introduce a novel method for relational database entity type recognition within organizations. Apart from recognizing conceptual entity types present within data instances, the solution should also aim to uncover semantic heterogeneity, which is the phenomenon that occurs when multiple data sets contain data on the same conceptual entity type but represent it differently (Giunchiglia et al., 2020). Semantic heterogeneity occurs due to the use of multiple different logical schemas in this case. Narrowing the scope of the problem to a specific organization constraints the problem space and enables domain-specific assumptions. Such assumptions include the availability of example data for each distinct entity type representation, or that the set of logical schema design rules is identifiable.

During the review of relevant literature, it became evident that there was little published research on tackling database entity type recognition. Furthermore, potentially helpful tools such as Sato (Zhang et al., 2019) were not ready to be employed as is, and required too much time to be prepared during this thesis' time frame. These findings resulted in a personal thought process that led to the solution proposed in this thesis.

The collaboration with a bank led to two minimum requirements for the proposed solution: maximal explainability and the elimination of integrity concerns. Based on these requirements and personal interest, the choice was made to use data profiling to make representations of the characteristics of the example data. This allowed for the solution to keep a summary of the data without storing any real information, eliminating integrity concerns. The entity type recognition process would then classify not-before seen data in a rule-based manner. This allows for easy explanations of the results in the form of a step-by-step walkthrough. Furthermore, data profile attributes and their corresponding rules are easily substitutable, maximizing the adaptability of the proposed solution.

The proposed solution was evaluated through performing experiments on its implementation. The data used for these experiments was subject to the bank's policies on securely handling data. Therefore, most of the experiments were conducted using generated data, and only one of the experiments was conducted using real data to validate the other results. The generated data was acquired by using a data generator developed by an expert at ING. It should resemble real data at the bank well. That is, the generated data should have roughly similar table, column and row characteristics as real data. And while the generated data did not perfectly resemble the real data, the experiments show that the evaluation on generated data is valid.

Results from all experiments reflect that the proposed solution is a promising concept for relational database entity type recognition, but that some tweaks in the implementation are necessary to optimize results. This requires organization-specific preparation, including the gathering of example data, preferably with entity type labels, as well as the establishment of fitting data profiles and classification

rules. It is important that this preparation is done with care to avoid inaccuracy. But with proper preparation, the proposed solution is a promising concept for entity type recognition in relational databases stored within organizations, and shows that simple solutions can be fruitful.

7.1 Future Work

The main purpose of developing a database entity type recognition solution was to help individuals or teams within organizations with identifying the conceptual entity types that are represented in data maintained by other individuals or teams, for which the proposed solution shows promising results. The proposed methodology could also be taken one step further, such as being applied to data integration. Rather than using schemas and data content directly as the basis for data integration, as is done in a practice called schema matching, database entity type recognition could be used to gather semantic context of the data instances in a data lake or warehouse. However, this would probably require a more sophisticated approach to uncovering semantic heterogeneity. In conclusion, entity type recognition solutions such as the one presented in this thesis might be a prevalent subtask of the data integration pipeline in future work.

Other future work in relation to database entity type recognition should look into the development of semantic graphs, such as done recently in (Sleeman et al., 2015; Giunchiglia et al., 2020). Semantic graphs appear to be a promising technique for keeping semantic context extracted from many data sources, albeit more complex than the approach taken for the solution in this thesis, but also providing a more sophisticated semantic context. Constraining semantic graphs to the domain within a specific organization arguably makes their application a lot more feasible than when doing so in the open world. And while their development likely requires quite some effort from experts at the organization, so does acquiring sufficient labeled example data for the solution presented in this thesis.

Bibliography

Abedjan, Ziawasch et al. (2015). "Profiling relational data: a survey". In: *The VLDB Journal* 24.4, pp. 557–581.

Andersson, Martin (1994). "Extracting an entity relationship schema from a relational database through reverse engineering". In: *International Conference on Conceptual Modeling*. Springer, pp. 403–419.

Chiang, Roger HL et al. (1994). "Reverse engineering of relational databases: Extraction of an EER model from a relational database". In: *Data & knowledge engineering* 12.2, pp. 107–142.

Elmasri, R et al. (2000). *Fundamentals of Database Systems*. Springer.

Fernau, Henning (2009). "Algorithms for learning regular expressions from positive data". In: *Information and Computation* 207.4, pp. 521–541.

Giunchiglia, Fausto et al. (2020). "Entity type recognition–dealing with the diversity of knowledge". In: *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*. Vol. 17. 1, pp. 414–423.

Hai, Rihan et al. (2019). "Relaxed functional dependency discovery in heterogeneous data lakes". In: *International Conference on Conceptual Modeling*. Springer, pp. 225–239.

Harmouch, Hazar et al. (2017). "Cardinality estimation: An experimental survey". In: *Proceedings of the VLDB Endowment* 11.4, pp. 499–512.

Ionescu, Andra (2020). "Reproducing state-of-the-art schema matching algorithms". MSc Thesis. Delft University of Technology.

Khalid, Hiba et al. (2019). "Metadata Discovery Using Data Sampling and Exploratory Data Analysis". In: *International Conference on Model and Data Engineering*. Springer, pp. 106–120.

Koehler, Martin et al. (2017). "Data context informed data wrangling". In: *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, pp. 956–963.

Malpani, Ankit et al. (2010). "Reverse engineering models from databases to bootstrap application development". In: *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*. IEEE, pp. 1177–1180.

Naumann, Felix (2014). "Data profiling revisited". In: *ACM SIGMOD Record* 42.4, pp. 40–49.

Papenbrock, Thorsten et al. (2015). "Functional dependency discovery: An experimental evaluation of seven algorithms". In: *Proceedings of the VLDB Endowment* 8.10, pp. 1082–1093.

Psarakis, Kyriakos (2020). "Holistic Schema Matching at Scale". MSc Thesis. Delft University of Technology.

Shaabani, Nuhad et al. (2017). "Incremental discovery of inclusion dependencies". In: *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*, pp. 1–12.

Sleeman, Jennifer et al. (2015). "Entity type recognition for heterogeneous semantic graphs". In: *AI Magazine* 36.1, pp. 75–86.

Tschirschnitz, Fabian et al. (2017). "Detecting inclusion dependencies on very many tables". In: *ACM Transactions on Database Systems (TODS)* 42.3, pp. 1–29.

Zhang, Dan et al. (2019). “Sato: Contextual semantic type detection in tables”. In: *arXiv preprint arXiv:1911.06311*.