A study of provenance in databases and improving the usability of provenance database systems

by

Omar AlOmeir

BSc in Computer Science, Prince Sultan University, 2012

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
(Computer Science)

The University of British Columbia
(Vancouver)

December 2015

© Omar AlOmeir, 2015
Abstract

Provenance refers to information about the origin of a piece of data and the process that led to its creation. Provenance information has been a focus of database research for quite some time. In this field, most of the focus has been on the subproblem of finding the source data that contributed to the results of a query. More formally, the problem is defined as follows: given a query q and a tuple t in the results of q, which tuples from the relation R accessed by q caused t to appear in the results of q. The most studied aspect of this problem has been on developing models and semantics that allow this provenance information to be generated and queried. The motivations for studying provenance in databases vary across domains; provenance information is relevant to curated databases, data integration systems, and data warehouses for updating and maintaining views.

In this thesis, I look extensively at provenance models as well as different system implementations. I compare the different approaches, analyze them, and point out the advantages and disadvantages of each approach. Based on my findings, I develop a provenance system based on the most attractive features of the previous systems, built on top of a relational database management system. My focus is on identifying areas that could potentially make provenance information easier to understand for users, using visualization techniques to extend the system with a provenance browsing component.

I provide a case study using my provenance explorer, looking at a large dataset of financial data that comes from multiple sources. Provenance information helps with tracking the sources and transformations this data went through and explains them to the users in a way they can trust and reason about. There has not been much work focused on presenting and explaining provenance information to database
users. Some of the current approaches support limited facilities for visualizing and reporting provenance information. Other approaches simply rely on the user to query and explore the results via different data manipulation languages. My approach presents novel techniques for the user to interact with provenance information.
Preface

This thesis is an original intellectual product of the author, O. AlOmeir. No part of this thesis was previously published. No ethics certificates were required.

The GLEI Watch system and data described in chapter 2, sections 2.1 and 2.2 is based on the work of V. L. Lemieux, P. Phillips, H. S. Bajwa, and C. Li. None of the text of the thesis is taken directly from previously published or collaborative articles.
# Table of Contents

Abstract ................................................................. ii  
Preface ................................................................. iv  
Table of Contents ....................................................... v  
List of Tables ........................................................... viii  
List of Figures ............................................................ ix  
Acknowledgments ........................................................ xii  
Dedication ................................................................. xii  

1 Introduction ......................................................... 1  
  1.1 Motivations .................................................. 1  
  1.2 Contributions ............................................. 3  
  1.3 Terminology ............................................... 3  
  1.4 Thesis Organization ...................................... 4  

2 GLEI Watch System ............................................... 5  
  2.1 GLEI Watch system architecture ......................... 5  
     2.1.1 GLEI Watch system current architecture ........ 6  
     2.1.2 GLEI Watch system new architecture ............ 7  
  2.2 The GLEI data ............................................. 8  
     2.2.1 Incorporating data from other sources ........... 9
## 3 Background

3.1 Types of provenance

3.1.1 Lineage

3.1.2 Why and where provenance

3.1.3 How provenance and semiring polynomials

3.2 Overview of approaches

3.2.1 Lineage tracing in data warehouses

3.2.2 Why and where semi-structured model

3.2.3 DBNotes

3.2.4 Orchestra

3.2.5 Perm

3.2.6 Other approaches

3.3 Summary

3.4 Related work

3.5 Further background

## 4 Implementation

4.1 Design principles

4.2 SQL support

4.3 PI-CS Provenance type

4.3.1 Relational algebra provenance rules

4.3.2 Provenance attributes renaming function

4.4 System architecture

4.4.1 Query translation

4.4.2 Query converting

4.5 User interface

4.6 Future work

## 5 Experimental Evaluation

5.1 Performance and overhead

5.1.1 Experimental setup

5.1.2 Data
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>GLEI data schema</td>
<td>8</td>
</tr>
<tr>
<td>3.1</td>
<td>Employee table</td>
<td>13</td>
</tr>
<tr>
<td>3.2</td>
<td>Project table</td>
<td>13</td>
</tr>
<tr>
<td>3.3</td>
<td>Q1 results</td>
<td>14</td>
</tr>
<tr>
<td>3.4</td>
<td>Intermediate view from Q1</td>
<td>16</td>
</tr>
<tr>
<td>3.5</td>
<td>Employee table with annotations</td>
<td>21</td>
</tr>
<tr>
<td>3.6</td>
<td>Project table with annotations</td>
<td>21</td>
</tr>
<tr>
<td>3.7</td>
<td>Q1 results in Perm</td>
<td>28</td>
</tr>
<tr>
<td>3.8</td>
<td>Perm query results with missing values</td>
<td>31</td>
</tr>
<tr>
<td>3.9</td>
<td>Provenance systems comparison table</td>
<td>38</td>
</tr>
<tr>
<td>4.1</td>
<td>Results of an aggregation query without provenance</td>
<td>43</td>
</tr>
<tr>
<td>4.2</td>
<td>PI-CS example results of an aggregation query with provenance</td>
<td>43</td>
</tr>
<tr>
<td>4.3</td>
<td>Relational algebra conversion rules</td>
<td>44</td>
</tr>
<tr>
<td>4.4</td>
<td>Examples of relational algebra conversion rules</td>
<td>45</td>
</tr>
</tbody>
</table>
List of Figures

Figure 2.1 The GLEI Watch system architecture ...................... 6
Figure 2.2 A geographic map visualization showing registrations per country ........................................ 7
Figure 2.3 The new suggested system architecture .................. 7
Figure 3.1 Mapping a relation into a graph .............................. 19
Figure 3.2 A data sharing graph ....................................... 25
Figure 4.1 Provenance system architecture .............................. 45
Figure 4.2 Converted ASPJ query ...................................... 47
Figure 4.3 Provenance system user interface .......................... 48
Figure 5.1 A line plot of the queries running with overhead of my system measured ........................................... 52
Figure 5.2 A line plot of the queries running without the overhead of my system measured ................................ 53
Figure 5.3 A barchart of the result sizes of different queries ......... 54
Figure 6.1 The interface of the provenance browser from [2] ...... 57
Figure 6.2 The visualization of the provenance of a value in a table in DBNotes. From [9] ................................. 58
Figure 6.3 The data visualizations with search, filter and sort options .......................................................... 59
Figure 6.4 Selecting an individual data item to see its provenance .......................................................... 60
Figure 6.5 The provenance data tables with the provenance graph ................................................................ 60
Figure 6.6 Drilling down to the ancestor of the table in Figure 6.5 ................................................................ 61
| Figure 6.7 | A visualization of the query and the rewritten provenance query | 61 |
| Figure 6.8 | The derivation history of GLEI data from origin to the final aggregated view | 64 |
| Figure 6.9 | A world map with aggregated count of registrations in a country | 66 |
| Figure 6.10 | A barchart showing aggregated count of registrations in a country | 67 |
| Figure 6.11 | A treemap with aggregated count of registrations in a country | 67 |
Acknowledgments

First of all, all praises go to Allah most merciful for his guidance.

I would like to start by expressing my utmost gratitude to my supervisor, Rachel Pottinger for her support through the past two years and for being a constant source of guidance and inspiration through normal and extraordinary circumstances.

My gratitude to the second reader, Raymond Ng, for his time, original ideas, and thorough questions and feedback.

I am incredibly grateful to my family. First and foremost, my mother and father, for their unwavering dedication to our education throughout the years and their unrelenting love and support. They stood by me throughout this whole process and I shall always be grateful.

My gratitude to my wife, Abrar, my partner through this journey. Without her love and support I would not have been able to complete this work. I am grateful for her bearing with me through this difficult process even though she was going through her own share of difficulty.

My gratitude to my siblings, Nora, Othman, and Khold for believing in me and for pouring support and encouragement whenever I needed them.

To my daughter Serene, thank you for being my guiding light.

Thanks to my fellow students and lab-mates. Special thanks to Jessica Wong for helpful discussions and feedback.

Finally, many thanks to the Saudi government and Prince Sultan University for their funding and financial support.
Dedication

To my loving family.
Chapter 1

Introduction

Data provenance is defined as any information about the origin of a piece of data and the process that led its creation. In [6], the definition is as follows “Data provenance, sometimes called lineage or pedigree, is the description of the origins of a piece of data and the process by which it arrived in a database” [6]. Provenance has been a focus of research in a number of different areas. In database research, most of the focus has been on the subproblem of tracking the source and process that contributed to the results of a query. There has been a lot of work on developing comprehensive models and representations of data provenance. There has also been work on different implementations that looked at different aspects of the problem, such as propagating provenance data and querying provenance information.

1.1 Motivations

The motivations for studying provenance in databases vary from the prevalence of curated databases to updating and maintaining views. The study of provenance started as a study of lineage in data warehousing environments [12] and later extended into all relational databases. The challenge of data provenance comes from the fact that when multiple sources contribute to a result, tracking the different sources and the transformations that took place is difficult. This difficulty is amplified by the different models and semantics used to describe provenance informa-
In this thesis, I will take an extensive look at techniques to represent and describe provenance. I aim to also look at the pros and cons of current theoretical representations and system implementations. I will also look at how different system implementations generate and query provenance information. I will look at some of the seminal works in representing database provenance such as [6] and [17], some of their implementations, as well as the numerous surveys and primers that sum up the state of provenance research, including [14] and [8]. My goal is to compare them in terms of common traits, point out the pros and cons of each approach, and find out what facilities could be useful for users when looking at provenance information.

The end goal and motivation of this project is to find novel ways to explain provenance information to users. In my case, I started out by looking at a large dataset of financial data that comes from multiple sources. This dataset requires the use of provenance information to find the sources and transformations this data went through to explain them to the users in a way they can trust and reason about. The challenge here is that there has not been much work focused on presenting and explaining provenance information to database users. Most of the focus has been on formulating the problem and coming up with formal models and query language extensions. Some of the current approaches support (limited) facilities for visualization and reporting provenance information. Other approaches simply rely on the user to query and explore the results via different data manipulation languages. I reach the conclusion that most of the work on data provenance is lacking in terms of taking the end-user’s perspective. The models can be complex and some query languages can be unintuitive. I summarize the most useful features of current models and systems, and classify them according to what I find to be the most important features. Using this scheme I implement my own provenance system and extend it with visualization facilities making it a more user-centered effort than current approaches.
1.2 Contributions

There has been a lot of work on the development of provenance enabled database management systems as explored in Chapter 3. However most of the work has been focused on developing comprehensive systems with extended query language support over SQL or other semi-structured languages. My goal in this thesis is not to propose a new system or a new model but to explore the space of provenance research, implement a system based on previous approaches with any necessary modifications, and to study and improve how users handle provenance data in the system. So to summarize I identify my contributions as:

- Take a comprehensive look at provenance research in databases and come up with a concise summary of desirable features for future systems.
- Implement an efficient and extensible system based on my findings with sufficient support for query types that I can test.
- Extend my system with web APIs that enable my visualization component. I incorporate provenance information into my visualizations in a simple and easy to use fashion.
- Develop a provenance exploration component for database provenance that is both novel and efficient.
- Verify my approach with a case study by developing an application for the GLEI financial dataset using my provenance system and provenance explorer.

1.3 Terminology

To avoid confusion, I will define my terminology in this subsection. The field of data provenance is relatively new and research is often presented with different words used to describe the same things. I would like to define and unify the terminology used throughout this thesis here.

The word model is used throughout this thesis to refer to logical models. Storage or physical models will be referred to explicitly as that. Lineage, why, where,
and how provenance will be referred to as notions or types of provenance or provenance semantics, this can be a point of confusion since some works refer to them as models. Other works refer to them as part of a bigger provenance category titled “data provenance”, I forgo this sub-categorization and call them types of provenance or provenance semantics.

Examples will for the most part be presented using relational tables and queries using SQL or SQL-like notation. However, depending on the system, exceptions will be made for systems or models that use different models and data manipulation languages.

I use the words lineage and derivation history interchangeably when talking about provenance information. I also use the word ancestor to refer to a source and immediate ancestor to refer to a direct source.

There are two main approaches to computing provenance, the information is either actively recorded and stored when the data is generated or a transformation is applied (eager) or it is generated when a user requests it (lazy). I will use these two terms to categorize systems in later discussion but there are systems that blur the line between the two.

1.4 Thesis Organization

This thesis is structured as follows: I describe our motivation and the GLEI financial system in Chapter 2. Chapter 3 contains an overview of provenance research. I describe the system I implemented in Chapter 4, complete with a description of the theory behind it. I conduct some experiments to validate the system and measure performance and present my results in Chapter 5. The visualization component is described in Chapter 6. Finally, I conclude with my conclusions and a discussion of future work in Chapter 7.
Chapter 2

GLEI Watch System

The Global Legal Entity Identifier (GLEI for short) Watch Project was established with the aim to improve financial transparency and stability monitoring. The GLEI Watch system integrates financial datasets from various heterogeneous sources called legal operating units (LOUs) and presents them to users for analysis and visualization. Each data unit (tuple) represents a legal entity identified by its LEI (legal entity identifier). LEIs are designed to be a single, universal standard identifier for any organization or firm involved in a financial transaction internationally [7].

In this chapter I look at the GLEI Watch system architecture and the changes I made to it in Section 2.1. I examine the GLEI data in Section 2.2 and finally present my GLEI provenance use case in Section 2.3.

2.1 GLEI Watch system architecture

The GLEI Watch system was already an up and running project when I joined to look over the challenges they were tackling. I start by looking at the system architecture that was in place when I joined then I move on to look at the suggestions I made to change the system.
2.1.1 GLEI Watch system current architecture

When I started working on this project, the system had an architecture composed of different solutions that run sequentially to integrate and normalize the different datasets. The results are visualizations shown to the user designed to answer a number of questions.

The GLEI system is composed of the following components:

- The **Collector** is a number of scripts that run daily to collect the data from LOUs (Legal Operating Units) either by directly downloading them or using web scraping techniques. The collector also performs some of the cleaning and normalization tasks such as normalizing dates and adding geo-location to each legal entity. The LOUs data comes in XML or CSV formats. The data files also have different schemas, the differences are resolved in the next stage.

- The **Pentaho** suite of tools provide drag-and-drop facilities for integrating data titled Spoon. The data integration scripts, described in XML, combine all data files into one file. Some normalization is performed by the tool to rename the attributes and solve some XML problems. The results are then combined into a single file. The datasets lose some details in the process, such as the different semantics for similar attributes, and the data types of attributes. The scripts are updated manually in case of new data formats or errors.

- The visualizations are implemented using **Tableau** [18]. A visualization and analysis tool that enables fast visualization and rapid prototypes. The output of the previous component is converted into a compatible file format and loaded manually into Tableau. Tableau visualizations provide the user
with the ability to browse and gain insights into the data. However there is a number of challenges with visualizing data of this volume. The least of these challenges is data clutter when attempting to show all the information in a single visualization see Figure 2.2.

![Figure 2.2: Heavy data clutter can be seen in places with large numbers of registrations like the US and Germany](image)

### 2.1.2 GLEI Watch system new architecture

![Figure 2.3: The new suggested system architecture](image)

Part of my work was to suggest a design change and a new system architecture. Figure 2.3 shows the suggested design: The new system would be implemented using general purpose programming languages instead of relying on ready-made tools.

The following components make up the new system design:

- **The Collector** would remain essentially the same. However, the cleaning and normalization tasks done at this step would be moved over to the next component.
The data extraction component would handle setting up and extracting records from XML data and performing the necessary cleaning and normalization steps for raw data obtained from the LOUs or other sources. The output of this component would be the different datasets with a unified schema ready for integration.

Entity resolution is a component to link records that correspond to the same real world entries. It is referred to as different names in the literature such as reduplication or record matching. Entity resolution is only used when new data sets are added that do not use the legal entity identifier. In the GLEI current sources, each company is identified using the LEI, adding new data sets with a different identifier can make things difficult. Matching company records using the company name and address fields for example would require entity resolution methods.

The API exports data to visualization and analysis components. Instead of Tableau I would use D3.js, this will support the goal of giving the user more flexibility to explore the data and query the system. It also gives me the ability to integrate provenance information into the visualizations.

2.2 The GLEI data

The Global Legal Entity Identifier dataset is collected from various sources for the Global LEI watch project using the GLEI Watch system described in the previous section. The combined size of all the data is approximately 70 MB of size and contains 312,295 rows and growing daily, the data describes the same number of legal entities. Table 2.1 shows the schema of the final dataset.
2.2.1 Incorporating data from other sources

One goal of the GLEI Watch system is to incorporate datasets from sources other than the LOUs. Other datasets considered for the project include:


- Company information from the Securities and Exchange Commission Electronic Data Gathering, Analysis and Retrieval system (SEC EDGAR).

- A member list from the Federal Home Loan Banks (FHLB).

- Legal entity identifier information from the GMEI utility, a pre-Local Operating Unit (LOU) of the Global Legal Entity Identifier System (GMEI LEI).


The biggest challenge is that these datasets do not utilize the LEI, making it a somewhat difficult integration task. We can use the LegalName or address fields to match depending on the datasets. There is little overlap with exact matching (200 tuples with the FFIEC reports), using research in entity resolution can help. However, most entity resolution methods give fuzzy matches with varying degrees of certainty. The variation in names across datasets could be simple or complex. Presenting the results of matches to users can enhance the trustworthiness of the data, this is one area where provenance information can help.

2.3 Provenance use case

The nature of this project uncovers various interesting data-related challenges in data integration, data cleaning, information extraction and data provenance. The last of which is my current topic of interest. I am interested in techniques to present and visualize provenance information to the users. In my work on the GLEI Watch project I hope to integrate provenance capture and presentation techniques to improve trust in the data in order to support users making decisions using the collected
and normalized final results. I use the finalized GLEI dataset to validate my data provenance approach.

Queries are entered into the system by users when asking for a visualization or a view of a subset of the data. The system and visualizations are designed to answer the users’ questions. An example query would be: What are the cities in which the most registrations occur?. Translated into SQL the aggregation query would look like this:

```
SELECT count(LEI), City FROM GLEI GROUP BY City
```

The provenance system would transform the query and generate a query that gives the data results as well as the provenance information. The results of the query with provenance would be the entire dataset. This amount of information can be overwhelming. Presenting the user with interactive visualization they could inspect and use to browse the provenance results can help. The user could also edit the data, annotate it, or export subsets of it to files. All the while maintaining the original data provenance information. The user could also dig deeper and inspect a deeper level of provenance ancestors. A full description of the experiment is provided in Chapter 6.
Chapter 3

Background

Data Provenance has been studied in a number of different contexts resulting in a number of data models and system implementations based on those models. In this chapter, I look at provenance models as well as different system implementations, compare the different approaches, analyze them, and point out the advantages and disadvantages of each approach.

My focus is on identifying areas that could potentially make provenance information easier to understand for users and help me gain insight before developing my own system and provenance exploration component. A summary of my comparison results gives a breakdown of systems according to their most important features, this scheme helps me gain insight into the research and identify essential features for a provenance system.

Guided by my findings in this chapter, I develop a system that contains the most identified desirable features. My system is also made to be applicable to a set of financial data (described in chapter 2) with dissemination techniques inspired by the facilities present in systems like DBNotes [9] and work-flow provenance systems, that also goes a step further with visualization components and provenance browsing capabilities. I believe that such a system can make for a great contribution to this field of research.

I start this chapter by looking at the different provenance types in Section 3.1. Then I explore the current system implementations and some theoretical models in Section 3.2 with a discussion of each approach. I move on to summarize my
results in Section 3.3. I present several research efforts that looked at provenance research in Section 3.4. In Section 3.5 I briefly look at other works I believe to be relevant to provenance research.

3.1 Types of provenance

The discussion will focus on four main types of “data” provenance: Lineage, why, where and how provenance. How provenance can be seen as a type of transformation provenance since it represents the operations that a tuple went through from input to output. There are also other types, that do not fall under the category of “data” provenance, such as transformation provenance shown in [16]. However, only “data” provenance types were given widespread attention in the field. Therefore, discussion of other types will be limited to their respective systems.

The goals of these four provenance types can be summarized as finding the source of a piece of data and the process it went through to make it into the results of a query. Most of the work in database literature has been focused on that same sub-problem, explaining where data in the results of a query come from and the process they went through. To define the problem formally:

**Definition 1.** For a database $D$, a query $Q$, and the results of applying the query to the database $Q(D)$. The provenance problem in database is to find the sources in $D$ that contributed to $Q(D)$.

The aforementioned four notions of provenance have been the focus of most provenance research in databases. One of the earliest applications of provenance in databases is lineage in data warehouses as seen in [11]. In [6] the authors define a semi-structured model and through it, formally define the notions of why and where provenance. In [17], the authors define the notion of how provenance using semiring polynomials. All of these models have had a substantial effect on the research and have resulted in the implementations of multiple systems and query languages. It is worth noting that despite adhering to the abstract notions of provenance defined in those works (lineage, why, where and how), implementations often differ from the logical models presented in [11], [6] and [17]. Hence a discussion of abstract provenance types is necessary before we go into the different implementations.
### Table 3.1: Employee table

<table>
<thead>
<tr>
<th>name</th>
<th>salary</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Omar</td>
<td>20k</td>
<td>1234</td>
</tr>
<tr>
<td>John</td>
<td>22k</td>
<td>1233</td>
</tr>
<tr>
<td>Ali</td>
<td>25k</td>
<td>1239</td>
</tr>
</tbody>
</table>

### Table 3.2: Project table

<table>
<thead>
<tr>
<th>name</th>
<th>department</th>
<th>project</th>
</tr>
</thead>
<tbody>
<tr>
<td>Omar</td>
<td>CS</td>
<td>Printing</td>
</tr>
<tr>
<td>Omar</td>
<td>CS</td>
<td>Programming</td>
</tr>
<tr>
<td>Ali</td>
<td>Management</td>
<td>Training</td>
</tr>
</tbody>
</table>

#### 3.1.1 Lineage

Other than the fact that lineage is often used as a synonym for provenance, it is also known as one of the earliest types of provenance. Defined in [11], lineage gives tuples from the source that caused a tuple to appear in the answer of a query or in a view. To illustrate, applying the query Q1 to the data in Table 3.1 and Table 3.2, we get Table 3.3 with two distinct answers.

Q1:

```
SELECT name, phone
FROM employee e, project p
WHERE e.name = p.name
```

To answer the question: “What is the lineage of r1 in the results of Q1?” The notion of lineage gives the answer as simply the tuples that caused r1 to appear in the results. In this case, the answer would be \{t1, t4, t5\}. The notion of lineage was criticized as not being precise enough and caused the development of the next type, why provenance.

#### 3.1.2 Why and where provenance

In [6], the authors define a more formal notion of provenance. Through a semi-structured model that I will explore later on, they define the notion of why and
where provenance. For my purposes, I will present illustrative examples using SQL and relational data and refer to the next section for discussion on the semi-structured model. Intuitively, why provenance explains which tuples (witnesses) are necessary to make a tuple appear in the answer. More precisely, the authors define the idea of witnesses, witness basis and minimal witness basis, I will look at those concepts more formally when I discuss the why and where semi-structured model. For now, the intuition is to define a more precise notion of lineage where a smaller set of tuples are used to represent provenance information. To continue with our running example, the why provenance of r1 in Table 3.3 would be \( \{t1, t4\} \) or \( \{t1, t5\} \), either is sufficient to explain the presence of r1 in the results. In other words, you do not need both t4 and t5 to have r1 in the results of Q1, you only need one. This is something that lineage cannot represent.

Where provenance answers the question of where in the input that piece of data is copied from. It is a more precise notion of provenance that is only concerned with the location of a source of data. In our example, the where provenance of r1 in Table 3.3 would be \( \{t1\text{:name}, t1\text{:phone}\} \) since data is copied directly from attributes in tuple t1.

### 3.1.3 How provenance and semiring polynomials

In [17], Green et al. introduce the use of semiring polynomials to represent provenance information and the notion of how provenance. Since the two concepts are tightly related, it is worth explaining what semirings are and how they are used before discussing how provenance.
Semirings

A semiring can be defined as “a domain plus two operations satisfying certain properties” [4]. The definition and inspiration for semirings comes from the constraint satisfaction problems domain [4]. A formal definition is as follows:

**Definition 2.** In the semiring \( (\mathbb{N}, +, *, 0, 1) \), the domain is natural numbers \( \mathbb{N} \). The operations are addition + and multiplication *, both are closed, associative and commutative. 0 is the unit element of the + operation \( 0 + a = a \) and 1 is the unit element of * operation \( 1 * a = a \) and 0 is the absorbing element \( 0 * a = 0 \). * distributes over +.

These properties can be changed to represent data provenance using polynomials. One such semiring is \( (\mathbb{N}[X], +, *, 0, 1) \) where the domain is natural numbers used as coefficients for the set of tuple annotations (X). The addition operation refers to union and multiplication refers to join. This domain and operations give us polynomials that can be used to represent detailed provenance information such as the ones in the next subsection.

We lay out the theoretical foundation of semirings here and discuss them further in Section 3.2 when we discuss the Orchestra system.

How provenance

The notion of how provenance answers the question of what process the piece of data went through to end up in the output. This notion captured using polynomials can give a more detailed account of the transformation applied to the piece of data.

In our example, the how provenance of r1 in Table 3.3 would be the polynomial \( t1.t4 + t1.t5 \). This representation shows that t1 was joined with t4 and t5 and subsequent results were unioned to get the final result r1.

3.2 Overview of approaches

In this section, I will look at different implementations of provenance database systems. The focus is on how these systems represent and query provenance information. All the approaches we explore are system implementations, except for the why and where theoretical data model from [6] which as far as I know has not seen
an implementation with that exact data model. The data model in [6] is included be-
cause it is one of the earliest efforts into formally defining provenance in databases
and it gives insight into some of the possible problems that come up when dealing
with provenance data. A discussion follows every subsection to analyze and look
at the advantages and disadvantages of these approaches.

3.2.1 Lineage tracing in data warehouses

<table>
<thead>
<tr>
<th>name</th>
<th>salary</th>
<th>phone</th>
<th>department</th>
<th>project</th>
</tr>
</thead>
<tbody>
<tr>
<td>Omar</td>
<td>20k</td>
<td>1234</td>
<td>CS</td>
<td>Programming</td>
</tr>
<tr>
<td>Omar</td>
<td>20k</td>
<td>1234</td>
<td>CS</td>
<td>Printing</td>
</tr>
<tr>
<td>Ali</td>
<td>25k</td>
<td>1239</td>
<td>Management</td>
<td>Training</td>
</tr>
</tbody>
</table>

Table 3.4: Intermediate view from Q1 (the last column shows the lineage for illustration purposes)

In [11] and [12] there is one application of lineage (provenance) tracing in data
warehouses. In a warehouse environment, data is collected and integrated from
different sources. The data is then presented to users as materialized views. The
user of a data warehouse may need to drill down and see the source of a data item.
This system provides that ability to the users.

The system guarantees that the lineage it computes will contain only relevant
tuples and that it would be complete, it will contain all contributing tuples. Lineage
is computed at tuple granularity, users can ask for lineage of any tuple in a trace-
able view, the system would use the inverse query to generate the source tuples.
Provenance information is simply represented as the source tuples. An interesting
aspect of the system is that the user specifies, when defining a view, whether a view
is traceable. If it is, the auxiliary views are generated immediately. The tracing and
maintenance procedures are also generated and stored as meta-data. So while this
system is lazy in that the provenance data is only computed on request, it eagerly
generates the facilities that allow the lineage to be traced are computed and stored
once a view is defined as traceable.

To illustrate how the system works we can look back at our example of query
Q1. The system looks at the view definition Q1, divides it into two ASPJ seg-
ments, before and after the join and defines the necessary intermediate view, in this case Table 3.4. Then it generates a set of queries on the intermediate views for the lineage of the results. After that, it traces the lineage of the intermediate views from the source tables as shown in Table 3.4. For every tuple, it unions the provenance from the intermediate views that contribute to the source tuple. Hence the provenance of r1 would be returned as the union of the lineage of the two tuples in Table 3.4: \( \{t_1, t_4\} \cup \{t_1, t_5\} = \{t_1, t_4, t_5\} \).

**Discussion**

Lineage is criticized for being imprecise compared to why provenance. However, to its advantage, it is very intuitive and relatively easy to compute. Lineage of views also offers some strong guarantees on the relevance of provenance generated. Despite being an early system, the representation of lineage is simple and intuitive as simply source tuples. The generation of intermediate views and queries when a view is defined is very efficient. While the approach is lazy it eagerly computes all the necessary components for efficient generation of provenance for traceable views. The lazy approach makes storage space a non-issue, the trade-off between space and time is a key point we will see with future systems. On a different note, support for aggregation queries from such an early implementation of provenance in databases is interesting to see since it is missing from some later systems.

On the downside, the fact that lineage is not as precise as the notion of why provenance could cause problems in explaining the provenance of some results (as seen in the subsection on why provenance in 3.1). Furthermore, lineage is only presented as source tuples and the view definition. Such a representation can make it difficult to explain how the results actually came to be, especially in complex queries and limits dissemination tools design. One of the biggest missing features is the lack of storage of provenance information, the system is defined to only explain the tuples in views and not much else. However, this could be due to the nature of data warehousing problems that the system aims to tackle.

Lineage in data warehouses is limited to functions that cannot be generalized and used in my case. No storage of provenance data and reversing queries would limit provenance dissemination to simply showing the ancestors of a piece of data.
when requested. This may be sufficient in view update problems but it is not enough to explain derivation history of a piece of data or for data trust applications.

3.2.2 Why and where semi-structured model

In [6], the authors define the why and where notions of provenance within the context of a semi-structured model. The data model described in [6] is deterministic, each location of a piece of data can be described by a unique path. It’s an edge-labeled tree model, the out edges of each node are labeled with pieces of data. Relations can be expressed in this model by mapping keys to edges.

The following is a summary of some properties of this model:

- The model represents label:value pairs as follows x:y. These pairs represent the edge x and the sub-tree it leads to y.
- \{a:1\} represents the path a leading to value 1.
- x.a:1 is equivalent to \{x:{a:1}\}.
- v(p) denotes that p occurs in v otherwise it is undefined.
- The path representation of \{a:{1:c,3:d}\} is \{(a.1,c),(a.3,d)\}.
- w is a substructure of v, if the path representation of w is a subset of the path representation of v.
- The deep union of \(v_1\) and \(v_2\) is the value with path representation that is the union of both path representations of \(v_1\) and \(v_2\). This is only defined if the result is a partial function (If the result is not a set of unique paths it is not defined). Example: \{a:1, b:{c:2, c:3}, e:5\} is not a partial function, it violates injective property of partial functions (c leads to both 2 and 3!).

Relations are encoded into this model as follows: the relation’s name is the first edge at the root, then the keys of the relation are mapped as edges, each key is mapped to the tuple it identifies, if there is no key the entire tuple is modeled as an edge. Compound keys are modeled as semi-structured pieces of data on the edge following each other. Figure 3.1 shows a mapping of table 3.1 into this model.
The query language for the above model is referred to as DQL (D for deterministic) and is defined as follows:

where

\[ p_1 \text{ in } e_1 \]
\[ : \]
\[ p_n \text{ in } e_n ; \]
condition
collect x

\( p_i \) is a pattern of the data in the model but it can also include variables. \( e_i \) is like \( p_i \) but can include nested where.. collect statements. condition is a predicate applied to \( p_i s \). The query basically means: consider each assignment of variables that makes \( p_i \) a substructure of \( e_i \), evaluate the condition for each statement and if true add the value of \( e \) to the results. Finally, union all results. In this case the union used is a deep union (see definition above). It is worth noting that since this is derived from semistructured query languages, there are similarities. One can think of the query as finding matching patterns to \( p_i \) and conditions, if available, and returning the structure defined for the output in collect.

The aforementioned model and query language are used as a setup to computing why and where provenance. Why provenance is determined by finding the

\[ \text{Figure 3.1: Mapping a relation into a graph} \]
witness for a result in the query by syntactic analysis of queries. A witness is formally defined as follows:

**Definition 3.** A value s is a witness for a value t with respect to a query Q and a database D, if t is a subset of the results of Q(s) and s is a subset of D.

The definition of a witness basis is more restrictive than the full set of witnesses. A witness basis is a smaller witness set determined by looking at the query and the data. There is also an algorithm to efficiently calculate the witness basis in [6] that I will not cover here. The downside to witness bases is that they are not invariant for query rewrites, a minimal witness basis is. Equivalent queries are defined as those that generate equivalent results from the same sources. A minimal witness basis is defined as a set of all minimal witnesses where a witness is minimal if none of its members is also a witness. In my example, \{t1, t4\} is a minimal witness and so is \{t1, t5\}. \{t1, t4\} and \{t1, t5\} can be included in a minimal witness basis, but the set \{t1, t4, t5\} is not included since it contains the previous two sets which are witnesses.

For where provenance, the intuitive approach is to follow a syntactic analysis of queries similar to the one seen in why provenance. However, this proves to be difficult for a number of reasons. For instance, a query that contains a constant value can have the where provenance as the query itself. There can also be multiple contributing inputs to the output. Since a syntactic approach is difficult, a class of traceable queries is introduced that limits the queries to avoid undesirable properties and defines properties that make where provenance easy to compute and invariant over rewrites.

**Discussion**

The why and where provenance approach, using a data model based on semi-structured data models, was the first to introduce the notions of why and where provenance. While the notions themselves were very influential, the semi-structured model and definitions did not catch on. There are some merits to it, using a deterministic model with a unique path for each value can prove useful for computing provenance information for a tuple. Furthermore, the query language lends itself to syntactic analysis to find and generate the provenance information. A ma-
<table>
<thead>
<tr>
<th>name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Omar</td>
<td>{good}</td>
</tr>
<tr>
<td>John</td>
<td>{bad}</td>
</tr>
</tbody>
</table>

Table 3.5: Employee table with annotations

<table>
<thead>
<tr>
<th>name</th>
<th>department</th>
<th>project</th>
</tr>
</thead>
<tbody>
<tr>
<td>Omar</td>
<td>{great}</td>
<td>CS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Printing</td>
</tr>
</tbody>
</table>

Table 3.6: Project table with annotations

The major contribution of this work is the focus on defining subclasses of queries where provenance information would be invariant to query rewrites. In other words, all equivalent queries would have equivalent provenance information. Thinking about query rewrites will affect future works as we will see later in this section, but this approach and associated algorithms were not as effective.

The model also imposes a number of difficulties. It can be difficult to read and reason about the data model, it can also be difficult to query. Reminiscent of XML shredding, converting relational tables to a graph can be cumbersome. The query language, also similar to semi-structured languages, is very difficult to use and understand. This is probably due to the theoretical nature of this work, but there is no real concern for querying or viewing the provenance information itself, which proved to be a challenge for later approaches.

### 3.2.3 DBNotes

DBNotes [9] [3] is a system for annotating databases. It allows for adding annotations to any tuple in a database and propagating that annotation into query results. The system also explains provenance to users by showing a possible journey that a value in the results took to get there. Annotations do not adhere to the database schema, they are added to any attribute in a tuple and stored as extra information next to the attributes they annotate. Annotations are propagated in different ways specified by the user when writing a query. The default scheme (default) propagates the annotations according to where the data is copied from. It naturally copies annotations from the source of the data to the results. The downside is that
two equivalent queries can have the same results with different annotations. To demonstrate, Table 3.5 and Table 3.6 are two tables with annotations to the name attribute, shown in curly brackets. The annotations measure performance on a scale (bad-good-great). Applying the following query to the two tables:

```
SELECT DISTINCT e.name, e.salary, p.project
FROM Employee e, Project p
WHERE e.name = p.name
PROPAGATE DEFAULT
```

Gives us the result: (Omar {good}, 20k, Printing). However, applying an equivalent query:

```
SELECT DISTINCT p.name, e.salary, p.project
FROM Employee e, Project p
WHERE e.name = p.name
PROPAGATE DEFAULT
```

Would give us the same result with a different annotation: (Omar {great}, 20k, Printing).

The other scheme (default-all) propagates the annotations according to where the data comes from in all equivalent queries, essentially guaranteeing invariance under rewrites. For example, applying any of the previous two queries with the default-all scheme:

```
SELECT DISTINCT p.name, e.salary, p.project
FROM Employee e, Project p
WHERE e.name = p.name
PROPAGATE DEFAULT−ALL
```

Would give us the following results: (Omar {good, great}, 20k, Printing). The user can also define a custom scheme, specifying which annotation to favor in such cases. These schemes are specified at the time the query is written. The default-all scheme handles variation in query rewrites, the system can compute a finite set of equivalent queries that represents the infinite possibilities of equivalent queries. Queries are equivalent as long as the data in the results is the same, regardless of annotations. This set of equivalent queries is called a query basis and is defined as
a finite set of pSQL queries with default propagation schemes that are equivalent to the query. The provenance for a query under the default-all scheme is the union of all annotations of the output returned by the query basis. The authors define an algorithm for computing the query basis in [3] that I will not cover here.

The approach to provenance is based on the notion of where provenance. This is evaluated eagerly by propagating annotations from the source or lazily by computing a reverse query if a user asks for an explanation of why a value is in the results.

DBNotes also introduces a query language pSQL. In pSQL, there are two new features: a PROPOGATE clause where a user specifies one of the 3 methods of propagation available. To describe how querying works “for each tuple in the Cartesian product of relations in the FROM clause, we check that the conditions in the WHERE clause are satisfied and then propagate the qualifying annotations to the output as specified in the PROPAGATE clause.” [9]. For duplicate tuples, the annotations are merged. The keyword DISTINCT is always used since the system does not support bag semantics. To query provenance annotations, the ANNOT keyword can be used, which makes annotations first class citizens that a user can match against. This can be seen in the following example query where the annotation on the name field in Employee table is matched to a pattern:

```
SELECT DISTINCT *
FROM Employee e, ANNOT(e.name) a
WHERE a LIKE %bad%
PROPAGATE DEFAULT
```

The resulting tuple would be: (John {bad}, 22k).

DBNotes features a number of other facilities to help users understand provenance information. To explain provenance, when a user asks about a piece of data, DBNotes generates a reverse query that extracts the sources and binds them to variables. Then it explains the results by showing the user the bindings and the conditions (WHERE and SELECT clauses) that give the results. DBNotes can also produce a graph of the journey a variable took through different databases and transformations. It stores a database for the results of a query and it keeps track of sources and transformations via annotations. This way it produces a visualization
for the user explaining where a piece of data came form and through what queries.

**Discussion**

DBNotes uses a simple query language, very similar to SQL, with intuitive extensions to query annotations. It also implements a set of facilities that a user could use to interact with provenance information. It is the only database system I could find that offers visualization and explanation of provenance results. The generic approach to annotations is very interesting and allows for the implementation of multiple types of provenance. DBNotes’ approach to handling equivalent queries is intuitive although performance is clearly an issue as seen in the evaluation in [3]. To quote: “the performance of a query under the default-all scheme can be at worst eight times slower than the performance of the same query under the default or no propagation scheme (i.e., SQL query). At best, it runs about twice as slow.” [3], considering the default scheme is already slower than a regular query by 18-40%.

On the downside, eager annotation approaches can also have serious scalability concerns in terms of space, this is not an inherent flaw in DBNotes but it is one that is not mitigated by DBNotes designers, for example by limiting the number of levels to propagate annotations. Although admittedly the designers attempt to trade space for time. The tight-coupling of provenance annotations and storing it with the data prevents the system from implementing bag-semantics and hinders the flexibility of the system. Finally, when explaining provenance, the verbosity of output can be a concern. Looking at a large amount of text with pieces of the query to explain a simple result can be prohibitive to users.

For my purposes, DBNotes is a generic system for handling annotations and can be generalized to any context. However, if those annotations are complete provenance information, the system may not be able to handle such a load. The examples shown in [9] [3] have a small number of notes compared to the number of attributes. More importantly, the GLEI system does a lot of aggregation and resolving aggregation for such an approach with where provenance can prove to be difficult. I will leave discussing the provenance explaining and visualization components in DBNotes to Chapter 6.
3.2.4 Orchestra

Figure 3.2: A data sharing graph. The oval with + is the data source, rectangles are tuples and m is the schema mapping

In [17], the authors propose that the use of symbolic representations of semirings are sufficient to track and carry information from input to output in relational algebra operations. The authors use polynomials with integer coefficients. This representation also works in the space of incomplete, uncertain, and probabilistic databases with a theorized superior representation to such structures as maybe tables, c-tables, and probabilistic event tables. Polynomial representations of provenance can capture provenance detailed information with the semiring \((\mathbb{N}[X], +, \cdot, 0, 1)\). We discussed semirings previously in Section 3.1.

Orchestra uses provenance semirings to represent provenance information. Orchestra is a collaborative data sharing system where users contribute and share data using schema mappings. This network of mappings and relations explains the model that is the basis for Orchestra. Orchestra also supports a language for querying provenance called ProQL, as well as methods for storing and indexing provenance information.

The data model is basically a graph of data sources and mappings, showing the original base data, tuples and derivations. Figure 3.2 shows how Table 3.1 from our running example can be represented using a schema mapping expressed in Datalog. The schema mapping is as follows:

\[
m : \text{E}(\text{name, salary, phone}) :\text{A}(\text{name, salary})
\]
Querying this model is basically showing elements of the graph to the user with the results. Queries can be asking for common provenance, specific sources, or specific tuples. The model itself is an encoding of provenance semirings, thus it can be used by specifying different sum and product operations to represent a number of different things, including derivability, probability, lineage, trust, and weight/cost. In the derivability example, which is the closest to the notion of how provenance, all base nodes are given a value (true), a derivation node is mapped to the product (AND) of the annotation of all source nodes, a tuple node whose derivations are all annotated is mapped to the sum (OR) of all mappings that lead to it. The language ProQL has two main functions: (1) return a projection of the graph (a subgraph) for specific tuples or paths with bindings for distinguished variables and (2) return how to evaluate a subgraph or bindings as an expression for a specific semiring.

I will focus on the first function since it is more demonstrative of how provenance queries work. In (1) the user specifies in the FOR clause which variables to bind to relations or derivations. The WHERE clause is for conditions. The INCLUDE PATH clause specifies which nodes and paths to include in the output graph. The RETURN clause specifies bindings to be returned as result tuples. Figure 2 shows shows one data sources, two derivations A and E, and schema mapping m between A and E. The query has a variable $x$ bound to E and a variable $y$ bound to A, the INCLUDE path says to include all paths between A and E. It returns the part of the graph in Figure 3.2 and the tuples bound to $x$ as result tuples.

\[
\text{FOR } [E \ x] \leftarrow + \ [A \ y] \\
\text{INCLUDE PATH } [x] \leftarrow + \ [y] \\
\text{RETURN } x
\]

In (2) two new clauses are added. The EVALUATE semiring OF clause has the user specify what function they want to evaluate for the output (e.g. derivability, trust). The second clause is the ASSIGNING EACH clause, in this one the user specifies values for leaf nodes (e.g. base tuples or nodes without derivations) in a fashion similar to switch statements in programming languages, a second one can used to define a function to give values for mappings.
In the described implementation, the storage model is as follows. Provenance information is encoded in relations. Tuples in relations are identified by keys. Mappings are stored in relations with the keys for source and target relations. To execute queries: (1) schema mappings are converted into a provenance schema graph which contains one node for each relation and one node for each mapping, edges are added according to mappings’ target or source keys. (2) Match the paths in the query against the provenance schema graph to identify mapping and relation nodes. (3) Create a Datalog program that contains the mapping and relation nodes identified as well as any source relations to use. (4) Execute this program in SQL on mappings and tuples. For indexing, the authors use access support relations (ASRs), which intuitively amount to storing joins between provenance relations paths in materialized views or relations.

In [19], the authors conduct a thorough evaluation of graph projection and annotation queries. The system performs well and the indexes set up show speedup in the execution. This is pretty much new territory so it’s nice to see a performance evaluation.

Discussion

Provenance semirings can be used to model anything that uses annotations: lineage, probability, uncertainty. This allows Orchestra to cover provenance with different semantics and possibly different types such as how and why provenance. This is due to the fact that semirings can still represent the simpler notion of why provenance. ProQL can also express complex queries in different context. Last but not least, a full performance evaluation with optimizations such as support for indexes is refreshing to see.

While Orchestra excels in many ways, there are a few challenges when looking at generalizing its findings to generic provenance applications. Most importantly, the graph model lends itself well to the collaborative data sharing application but can prove to be difficult to generalize to other areas. The semantics for the semiring operations, addition and multiplication, are tied to the nodes in the graph. Similarly, the query language ProQL is similar graph query languages. As a result of tying everything to the graph, the provenance representations can be difficult to
understand and the queries themselves are complex as a result. As a result, it may not be worthwhile to try and implement this approach in any other context.

Orchestra provides a great basis for a provenance enabled data sharing context where schema mappings are prevalent. However, I am looking for a generic system implementation that would generalize to most datasets. For example, if I apply this approach to the GLEI system (described in Chapter 2) which does not fit this use case, the results would be a small graph and the queries would become needlessly complex.

3.2.5 Perm

The main idea behind Perm [16] is to generate and query provenance information using easily optimizable SQL code. It has support for complex SQL queries (nested, correlated sub-queries and aggregation) as well as storage of provenance information in the same relation as the data.

The authors define the following principles for an ideal provenance system and proceed to build Perm accordingly: (1) support for different types of provenance (why, where and how) with sound semantics, (2) provenance generation for SQL including complex queries, (3) support for complex queries over provenance data, (4) efficient generation and querying for large databases by only generating the necessary provenance information. Perm contributes to the four requirements as follows: (1) it supports the 3 types of provenance why, where, and how natively without using a different model. (2) Perm supports ASPJ queries and set operations (union, intersection, difference) and it is the only system to support nested and correlated sub-queries that I have seen. (3) Perm uses a relational representation of provenance and full SQL queries over provenance information. (4) The user

<table>
<thead>
<tr>
<th>Results</th>
<th>Employee</th>
<th>Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>phone</td>
<td>p1</td>
</tr>
<tr>
<td>Omar</td>
<td>1234</td>
<td>Omar</td>
</tr>
<tr>
<td>Omar</td>
<td>1234</td>
<td>Omar</td>
</tr>
<tr>
<td>Ali</td>
<td>1239</td>
<td>Ali</td>
</tr>
</tbody>
</table>

Table 3.7: Q1 results in Perm
chooses when and what provenance to generate, a sub query within the provenance query would generate the provenance information. Hence generation and querying are both in the same extended SQL query, that is subsequently rewritten into a single SQL query which utilizes the DBMS optimizer for non-nested queries. For nested queries, it uses novel un-nesting techniques.

When a user asks a query Q and asks for its provenance in the same query, Perm returns the provenance of results of Q to the user with the results of Q. For example, the results of Q1, using PI-CS provenance, are represented with their provenance information in Table 3.7. Perm represents provenance types differently. PI-CS provenance is represented as witness lists, the input tuples that were used to derive the output. This is represented by a relational table that includes the result tuples’ attributes and values appended by all attributes from contributing tables, and NULLs from tuples/attributes that do not contribute to the results. For result tuples with multiple tuples in their witness lists, duplicates are added to the table as in Table 3.7. All relations’ attributes carry a prefix and the name of the original relation and attribute (e.g. prov_Employee_name). C-CS provenance is just a variation of PI-CS provenance where only tuples that had information copied from them are shown in the results, hence it is a tuple-granularity where provenance. For transformation provenance, the information is represented by wrapping XML tags around the parts of the query that do not contribute to the result for each result tuple. The PI-CS representation is not compact enough but it can easily be queried for interesting information. The duplication can also cause a problem by hiding original results’ multiplicities. However, this is still an exciting new approach to representing provenance in a relation, and it shows how the provenance information is associated with the result tuples explicitly.

Rewriting queries into provenance generating queries, in Perm, works as follows: (1) Define the semantics declaratively of the provenance approach and define a relational representation. (2) From the declarative definition come up with algebraic rewrites to transform a query into a “provenance-generating” query, (3) Translate the relational algebra rewrites into SQL rewrites that use the optimizer. This is explained in detail for the 3 types of provenance PI-CS, C-CS, and transformation provenance in [16], and I also go into further detail describing our approach which is based on Perm in Chapter 4.
Perm data manipulation language adds the following extensions to SQL: (1) the PROVENANCE keyword in SELECT clause is added if the user wants the provenance of the query, (2) ON CONTRIBUTION allows the user to specify the type of provenance (PI-CS is the default), (3) BASE RELATION allows the user to limit how far back to go for results (e.g. Adding BASERELATION to an intermediate aggregation result would only use that in provenance representation, rather than go all the way back to the base), this feature is not to be confused with propagation of provenance data. Perm also allows users to add and query provenance information generated manually or from other systems to relations as long as it abides to the structure used by Perm and is stored in additional attributes, this is done by adding PROVENANCE keyword to the FROM clause to the provenance attributes (e.g. FROM PROVENANCE(name, employee, date)).

Discussion
Perm seems to be built on a solid foundation by defining sound principles inspired by previous provenance systems. It has support for aggregation and nested queries which is a novel contribution to the provenance space. Everything is done in SQL with natural extensions, which gives it the ability to utilize the query optimizer. It also has support for multiple types of provenance which I would argue is a necessity at this point, since the different semantics serve users differently, this can be seen in the discussion of the different types of provenance in Section 3.1 Furthermore, it notably features a very intuitive representation of provenance that users can browse through or query using SQL. Since Perm is an optimized lazy approach, it avoids the trade-off of time vs. space. It also allows the users to store provenance from other approaches or possibly provenance that is manually input or generated.

There are a few problems that can arise from the representation of provenance in Perm. The first is the verbosity of output. This is something that cannot be helped since the system opts for a complete representation of witness lists. The second problem comes from null values for provenance attributes. Null values in provenance attributes mean that those attributes did not contribute to the results. Although such missing values are semantically meaningful, problems can arise if the data itself contained missing values. As an example on this, applying the
Following query to Table 3.1 and Table 3.2:

```
SELECT PROVENANCE *
FROM SELECT name FROM Employee UNION
SELECT name FROM Project
```

The results are shown in Table 3.8. As described earlier, the provenance of the results contain all attributes from all contributing relations. The values for attributes that did not contribute to a result is left empty.

The last problem is inherent in all lazy approaches, Perm trades off propagation of provenance for smaller space. Perm stores provenance information in the same table as the data that show the immediate ancestor(s). While Perm can be classified as eager in that provenance can be calculated immediately without users asking for it, the distinction remains that eager systems usually offer propagation of annotations like we saw with DBNotes [9].

Perm’s biggest strength is its ability to support provenance operations over a fully relational DBMS with good performance. The problems outlined above either arise in very specific scenarios or can be mitigated using different provenance dissemination techniques. The effects of verbose output can be reduced by dividing the data and showing the users small manageable subsets of it with provenance information. Missing values from data can be made distinct from those that refer to non-contributing tables by filling in a different value for the latter. I am aware that this could prove tricky in certain scenarios so the option can be left to the user on how to fill these non-contributing cells. The last problem, even though Perm only stores information about immediate ancestors, graphs of derivation history can be constructed by tracing the source tables all the way to the origin. This

<table>
<thead>
<tr>
<th>Results</th>
<th>Employee</th>
<th>Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>p_name</td>
<td>p_salary</td>
</tr>
<tr>
<td>Omar</td>
<td>Omar</td>
<td>20k</td>
</tr>
<tr>
<td>Omar</td>
<td>Omar</td>
<td>20k</td>
</tr>
<tr>
<td>Ali</td>
<td>Ali</td>
<td>25k</td>
</tr>
<tr>
<td>John</td>
<td>John</td>
<td>22k</td>
</tr>
</tbody>
</table>

Table 3.8: Perm query results with missing values
problem can be resolved by enabling the users to browse derivation history graphs of data.

3.2.6 Other approaches

As part of this survey, I have looked at other systems briefly that do not warrant a full subsection or are outside the scope of the thesis. The first is similar to [11] in a lot of ways so I am not going into further details on it. The second is related to workflow provenance, since I am not covering workflow provenance systems, I will briefly write about this approach here.

The database system Trio [26] supports lineage as part of its components and is built on the principles of lineage of views in data warehouses [11]. Trio supports lineage, along with uncertainty as first class citizens that can be queried and stored. Trio features a lineage table which contains inverse queries for every tuple in a materialized view, this table can either be queried directly or using a proposed TriQL query language. The lineage inverse queries generate the provenance on request.

The other approach is referred to as a Graph model for workflow and database provenance [1]. This is an interesting venture into bridging the gap between workflow systems and database provenance. The two fields have remained separate for a while with little effort beyond suggesting and spotting similarities [25]. The motivation is that while workflow provenance is developed and used very often to describe the computation steps performed in constructing intermediate and final results, it lacks unified semantics. Each system can have its own semantics for provenance information making it difficult to compare approaches. In database research, contribution semantics are some of the most researched aspects of provenance. Furthermore, where provenance and lineage lend themselves to graph representations while how and why provenance can be more difficult to represent as graphs. The graphical model is explained in [1], expressed in a data flow language based on nested relational calculus. This work sets the groundwork for possibly establishing a connection between scientific workflow and database provenance.
3.3 Summary

Table 3.9 shows a summary of the models and systems I looked at in this survey, broken down by basic features to make them easier to compare. I can look at and explain each property before I go further into comparing them. Provenance types, query languages, models, storage and computing provenance have been explained already or are self explanatory. It is worth noting that querying provenance refers to querying provenance information and the operations available to do that, all approaches have query languages to query data but not all support querying the provenance data itself. Browsing provenance refers to any other facilities other than query languages, a system should ideally have both to support different types of users. Provenance granularity refers to the granularity at which the system supports finding the provenance of a piece of data (table, tuple, attribute). Provenance data coupling is borrowed from [15] and is defined when I talk about related work in section 3.4. However, in that paper the authors only categorize one system (lineage in warehouses), here I take it a step further and look at five different systems.

In this chapter I have looked extensively at the types of provenance, models and query languages in several database systems. My findings lead me to believe that the ideal system should have the following properties (the first three agree with the principles outlined in [16]):

1. Support for multiple types of provenance. Why, where and how and also transformation provenance can provide different information, and support different tasks for different users.

2. Support for a query language similar to SQL with intuitive extensions that allow for generating and querying provenance data. Moreover, the system should allow SQL queries over provenance data.

3. The model should be relational, as lineage in warehouses [11], DBNotes [9] and Perm [16] prove the relational model is sufficient for most cases. Furthermore, using the relational model will make it possible to use existing optimizations for provenance queries.

4. Storage of provenance data gives different trade-offs to consider. Loosely coupled provenance to the data or without coupling has the advantage that
querying provenance information would become simpler without data items carrying around the ancestors’ attributes. Tightly coupled information however, keeps track of tuples and their respective provenance information more easily. Tightly coupled approaches are the only ones to do that. Ideally a system should be able to support both storage methods for different granularities.

5. The system should allow for a hybrid approach for computing provenance, lazy generation of provenance with propagation like eager systems. All of this while keeping the user in the loop to determine the level of propagation. The goal is to avoid scalability concerns, “in one public database (the Gene Ontology) we often observed 10MB of lineage (provenance) data for a single tuple” [22].

6. While data provenance semantics and systems are being developed to be as comprehensive as possible, there is little regard for the user side of things. Very few systems offer facilities for the user to visualize/browse through provenance information. I argue that this is necessary to facilitate the understanding of provenance data.

7. Provenance systems should support provenance information at different granularities, tuple granularity is sufficient in most cases. However, attribute granularity can help when I am trying to explain changes at an attribute level (data cleaning tasks). Table granularity can also help with space concerns.

In the realm of theoretical models, thinking beyond the sub-problem of explaining the origin of tuples in the result of a query is limited, there is a need for thinking about what data to store, how to store it, propagate it, present it and explain it to users. Now that the theoretical models are set in place, these problems should come to the forefront of provenance research.

3.4 Related work

A lot of the content in this survey is inspired and influenced by some of the numerous surveys in data provenance. In [8] the survey is dedicated to comparing
why, where and how provenance in terms of how they relate to each other, as well as how the different approaches fit into these notions. They also compare systems and develop simple examples using SQL or datalog that facilitate comparisons. Similarly, I attempt to present examples using SQL queries and simple relations that are easy to grasp.

I take inspiration for Table 3.9 mainly from [25]. In [25], this survey compares four different scientific workflow systems and Trio [26]. It looks into why they record provenance, how they represent it, and how they use it. The goal is to provide a taxonomy of provenance approaches. The approaches are compared based on multiple criteria. Provenance information can have different “subjects”, it can be collected directly about the data (data-oriented) or derived indirectly from the process and how the data came to be (process-oriented). It can also be collected on different granularities, although most of the research on database provenance is tuple-based. Provenance methods can be either eager or lazy, this is a popular classification that I have seen in multiple works. The eager methods involve using annotations which can be rich and can carry more information, which have implications on storage and scalability. The lazy methods involve inversion, like reversing a SQL query. In terms of representations, There is no standard across disciplines, though most approaches explored in this survey store it as XML, the database models I looked at in my survey all store provenance information in relations. Storing and updating provenance information is another issue. Storing the annotations with the data can make it difficult to query. Involving the user and leaving the storage of provenance to users to do manually can prevent completeness. As for dissemination of provenance data, this information is usually presented to users as relations they can query and browse through.

In [15] similar to [25] the authors attempt to categorize provenance approaches based on the following features: provenance model, query and manipulation functionality, storage model, and recording strategy. The goal is to uncover research problems and suggest a sort-of unification of these approaches across different disciplines by looking at things from a more conceptual level. This motivates me as well to look into simplified, broken-down versions of systems to make them easier to compare. In this categorization scheme there is a number of novel ideas. In the provenance model category, they are looking at transformation and source prove-
nance and further subcategories that look at exactly what kind of data is being modeled. In terms of storage, systems are categorized into: (1) tightly coupled, where provenance information is stored with the data, (2) loosely coupled, provenance information is stored with the data but logically separated, and (3) no coupling, provenance information is stored in independent repositories. I adopt this idea in my results in Table 3.9. The authors also suggest a number of provenance-based manipulation operations: (1) a merge operation to merge transformations, (2) a split operation to split complex transformations, (3) recreate source data from results. Which so far, I have not seen implemented in a database provenance system. This paper uncovers some interesting details on provenance systems. However, the categorization scheme, in an attempt to predict the future, shows some features that have not and may not be implemented in any systems. Therefore in my comparison I only look at what is already there and leave prediction and potential problems to the future work section.

Finally, the primer in [14] classifies provenance information into data, transformation and auxiliary provenance. The last category I have yet to see in any database system, although storing information about the data types, users or environments can be valuable in certain contexts. In [14] the author also looks at the relationship between provenance, audit logs, and temporal databases.

3.5 Further background

The focus of this chapter is on models and provenance systems. For the future, I plan to look further into how systems handle things like query rewrites, and the algorithms that facilitate these operations.

I look further into scientific workflow provenance in Chapter 6 and how it is explained to users to see if there are any lessons applicable to a database context. Scientific workflow systems seem to provide a greater emphasis on user experience in general. However they lack the unified semantics of database systems. The survey at [25] suggests a close relationship between workflow and database provenance that is worth exploring. Therefore studying the relationship between the two starting with [1] will prove worthwhile. It would also be interesting to add a few scientific workflow systems into this comparison to see how they fare in issues of
scalability and usability.

The notion of “approximate lineage” [22] in uncertain databases is worth exploring as well. The idea of “working models” [13] is similarly inspiring. The theoretical focus is shifting from complete provenance information mainly due to the scalability for large databases. Taking in the user’s perspective on what is useful and sufficient would be a great contribution to this field of research and a first step into making sufficient models for provenance that are not necessarily as performance intensive.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Query language</td>
<td>SQL</td>
<td>DQL</td>
<td>pSQL</td>
<td>ProQL</td>
<td>Extended SQL</td>
</tr>
<tr>
<td>Model</td>
<td>Relational</td>
<td>Semi structured</td>
<td>Relational with annotations</td>
<td>A data sharing graph</td>
<td>Relational</td>
</tr>
<tr>
<td>Storage</td>
<td>No storage</td>
<td>-</td>
<td>Relational</td>
<td>Relational</td>
<td>Relational</td>
</tr>
<tr>
<td>Provenance granularity</td>
<td>Tuple</td>
<td>-</td>
<td>Attribute</td>
<td>Tuple</td>
<td>Tuple, attribute or relation</td>
</tr>
<tr>
<td>Provenance data coupling</td>
<td>No storage</td>
<td>-</td>
<td>Tightly coupled</td>
<td>Loosely coupled</td>
<td>Loosely coupled</td>
</tr>
<tr>
<td>Computing provenance</td>
<td>Lazy</td>
<td>Lazy</td>
<td>Eager and lazy</td>
<td>Eager</td>
<td>Lazy or eager (no propagation)</td>
</tr>
<tr>
<td>Querying provenance</td>
<td>None</td>
<td>-</td>
<td>Query annotations using pSQL as first class citizens</td>
<td>Query the graph using ProQL</td>
<td>Query the results using SQL as relational tables</td>
</tr>
<tr>
<td>Browsing provenance</td>
<td>None</td>
<td>-</td>
<td>Visualize and explain provenance</td>
<td>None</td>
<td>Browse tables</td>
</tr>
</tbody>
</table>

Table 3.9: Provenance systems comparison table
Chapter 4

Implementation

In this chapter I describe my work on a system implementation based on my findings in Section 3.3. From looking at the different system implementations, I came up with a number of desirable features that would help me shape my own customized system. Other approaches simply rely on the user to query and explore the results via different data manipulation languages. The models can be complex and some query languages can be unintuitive making things more difficult than using a relational DBMS. For that end, I developed a system based on relational DBMS approaches. My approach differs from others in a number of areas highlighted in section 4.1.

One of the conclusions that I made based on my finding is that provenance contains a large amount of information that could have a serious implication on storage and complexity of manipulation. Not to mention that it could be prohibitive to users. My goal is to design a system that takes the users’ perspective from all of this. See what information is valuable to users over others. I attempt with my provenance explorer to show the information to the users in way that is manageable. This leads me to design a system where the user can easily and efficiently browse through provenance information.

Moreover, using my system I can study and experiment with provenance meta-data, and evaluate the different provenance dissemination techniques and how they serve the user. My goal is to develop novel provenance browsing techniques to allow users to explore and understand provenance meta-data in the context of rela-
In this chapter I look at the entire process of developing a provenance system. I start by listing the design principles and goals of the system and how I achieve them in Section 4.1. I list and justify the subset of SQL operations I choose to work with in Section 4.2. I explain the provenance semantics I use in Section 4.3 with how they are implemented. I then give an overview of the system architecture in Section 4.4. I end by presenting a number of potential improvements to the system as future work items in Section 4.6.

4.1 Design principles

The system is built on the principles detailed in the related works chapter. As a reminder I list them briefly here and provide details on how I tackled each one:

1. Support for multiple types of provenance. I start by supporting the lineage type PI-CS provided in [16], in future work I intend to offer the user the ability to use Where provenance semantics as well.

2. A query language similar to SQL. I utilize the SQL language for querying data and provenance information alike. I also offer provenance information explicitly via an API.

3. The model should be relational. My approach can work over any relational DBMS.

4. Storage of provenance data. I give the option of either tightly coupled or uncoupled storage with different trade-offs for each for different granularities and tasks.

5. The system should allow for a hybrid approach, between lazy and eager, for computing provenance. My system provides lazy provenance computing, however in the provenance browser I give the user the ability to dig deeper into ancestors and browse provenance information.

6. Making provenance simpler for the user with different dissemination techniques. I provide provenance explaining and provenance visualizations that make it easier for the user to look at an overview whenever they want.
7. Provenance systems should support provenance information at different granularities. I provide tuple and table granularities, and the storage of provenance information changes with the level of granularity.

While there are certainly systems that attempted the first 3 items, my approach differs from systems like Perm [16] as follows:

- The system is implemented as an extra layer rather than modifying an existing DBMS. This allows me to implement it over any relational DBMS with minimal modifications.

- I provide different levels of granularity depending on the user task. There are instances where table granularity would be sufficient, such as looking at an overview of where the data comes from.

- I develop an API that allows users to extract data or subsets of it. The main goal of this API is to support the visualizations and analysis components, and support custom user visualizations.

- I provide a provenance explorer as a first class component of my system.

### 4.2 SQL support

The goal of my system is to study the usability of provenance data, and develop novel provenance dissemination techniques. My goal is not to develop a comprehensive provenance system, I believe this has already been done. Furthermore, the implementation of a provenance database management system is not a trivial task. To that end, I define a subset of SQL that I deem sufficient for my purposes and leave the implementation of other operators on a need-to-use basis. As it stands, I currently support the following subset of SQL operations in my system:

- Projection with renaming operations.

- Selection queries with comparison conditions.

- All join types; natural join, left join, right join, cross join, inner join with conditions where they are needed.
• Aggregation, all aggregation functions work. I focus on the following three in my tests: count, sum and average. Aggregation can also have selection in the HAVING clause.

I do not currently support the following operations. However, the system can easily be extended to do so.

• SET operators: UNION, DIFFERENCE and INTERSECTION. I can implement these operators on a need-to-use basis.

• LIMIT, ORDER BY, and OFFSET operators. These operators cannot be converted into relational algebra, and I believe they can be ignored when computing provenance (the same goes for TOP and similar operators) since they do not really change the query and just modify the output. I utilize other methods using the API to simulate their effects.

• The system does not currently support sub-queries. I handle sub-queries by defining views then querying them, making it easier to reason about and track data changes.

Currently the system has sufficient capabilities to answer queries from the GLEI datasets, even when joined with other datasets (such as offshore entities or FFIEC). The focus right now is on ASPJ (aggregate-select-project-join) queries with comparison conditions, such as:

SELECT * FROM glei, offshore
WHERE glei.entityID = offshore.entityID AND glei.country = "United States"

AND

SELECT count(LEI) AS count, Country, City FROM glei
GROUP BY Country, City

The reason for this subset is that most of the requirements for the GLEI system center around aggregation, for example: What are the countries in which the most registrations occur? Which LOUs have the highest number of registrations?

In chapter 2 I covered all the requirements for the GLEI watch system.
Query:
SELECT count(id) AS count FROM employee GROUP BY department

<table>
<thead>
<tr>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

Table 4.1: Results of an aggregation query without provenance

<table>
<thead>
<tr>
<th>count</th>
<th>provenance_employee_name</th>
<th>provenance_employee_phone</th>
<th>provenance_employee_id</th>
<th>provenance_employee_department</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Omar</td>
<td>604111</td>
<td>1234</td>
<td>CS</td>
</tr>
<tr>
<td>2</td>
<td>John</td>
<td>604101</td>
<td>1254</td>
<td>ECE</td>
</tr>
<tr>
<td>2</td>
<td>Ali</td>
<td>604003</td>
<td>2354</td>
<td>CS</td>
</tr>
</tbody>
</table>

Table 4.2: PI-CS example results of an aggregation query with provenance

4.3 PI-CS Provenance type

The provenance type implemented is part of the Perm Influence contribution semantics (PI-CS) [16]. It is based on lineage with modifications that make it work for more SQL query types. This type of contribution semantics or provenance type gives back the results with their corresponding provenance information or witness lists. It works by defining a relational algebra set that maps SQL operations to relational operators. The main difference between Perm relational algebra and standard algebra is the inclusion of bag and set semantics projection operators. These bag projection operators are used in all provenance generating queries.

Provenance information under PI-CS is represented as witness lists, the input tuples that were used to derive the output. This is represented by a relational table that includes the result tuples’ attributes and values appended by all attributes from contributing tables, and NULLs from tuples/attributes that do not contribute to the results. For result tuples with multiple tuples in their witness lists, result tuples are duplicated and are added to the table. Table 4.2 shows an example of the results to an aggregation query.

The authors also define a set of relational algebra rules that convert a regular relational algebra expression into a provenance generating expression, I look at the subset of those rules that I implemented in the next subsection.
4.3.1 Relational algebra provenance rules

The rules to convert a relational algebra query into a provenance generating query are based on the work of [16]. Table 4.3 shows the rules to convert a relational algebra expression into a provenance generating expression based on the semantics of PI-CS. The $A^+$ symbol represents all attributes from all contributing tables. R1 shows a simple relation access converted into a provenance generating expression, in this rule a table is converted into select all attributes and all attributes renamed into provenance information format using the $n()$ function. In R2 the SELECT operation is straight forward with changing the table into a provenance table. A PROJECT operation in R3 is converted to project both the required attributes $A$ and the $A^+$ as well. Aggregation in R4 is trickier since information cannot be propagated through this operation, so a left join between the original query and a second query that selects all attributes along with the group by attribute. The two tables are joined over the group by attribute $G$, agg refers to the aggregation functions. R5 shows the JOIN operations, $A^+$ in this case would mean all attributes in all contributing tables, different join types work the same way with or without the C condition.

Table 4.4 contains examples of each rule applied to a SQL query.

Table 4.3 shows the rules that were implemented in my system. (R5) is used for all possible join types by simply substituting the JOIN symbol in place of JOIN.

For formal proofs that these rules work the reader can refer to [16].
Table 4.4: Examples of relational algebra conversion rules

<table>
<thead>
<tr>
<th>Rule number</th>
<th>Original query</th>
<th>Provenance query</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>SELECT * FROM R</td>
<td>SELECT * FROM R, R.a AS prov_R.a, R.b AS prov_R.b, R.c AS prov_R.c</td>
</tr>
<tr>
<td>R2</td>
<td>SELECT * FROM R WHERE a &gt; 1</td>
<td>SELECT * FROM R, R.a AS prov_R.a, R.b AS prov_R.b, R.c AS prov_R.c</td>
</tr>
<tr>
<td>R3</td>
<td>SELECT a, b FROM R</td>
<td>SELECT a, b FROM R, R.a AS prov_R.a, R.b AS prov_R.b, R.c AS prov_R.c</td>
</tr>
<tr>
<td>R4</td>
<td>SELECT sum(a) AS sa FROM R GROUP BY b</td>
<td>SELECT orig.sa, prov_R.a, prov_R.b, prov_R.c</td>
</tr>
<tr>
<td>R5</td>
<td>SELECT * FROM R JOIN T ON R.b = T.b</td>
<td>SELECT * FROM R JOIN T ON R.b = T.b, T.d AS prov_T.d, T.b AS prov_T.b</td>
</tr>
</tbody>
</table>

4.3.2 Provenance attributes renaming function

The $n()$ function shown in R1 table 4.3 is responsible for renaming generated provenance attributes. The naming scheme I developed renames a provenance attribute into the following: Provenance_OriginalTableName_AttributeOriginalName. Provenance is a keyword to distinguish provenance attributes from regular attributes. Adding the original attribute name makes it possible to identify where it comes from in the original table. Table 4.2 shows an example of how this works.

4.4 System architecture

![Diagram](image)

**Figure 4.1:** The idea behind the system is to take a query, parse it and process it to produce a provenance generating query.

The plan is to implement a system based on the findings of the survey on database provenance, mainly inspired by [16]. The idea behind the system is to take a query, parse it to produce tokens of each context (SELECT, FROM, WHERE, GROUP BY.. etc contexts). The parser is non-validating, any errors in the SQL
query is reported back by the DBMS. The tokens are translated to a relational algebra expression. The relational algebra expression is converted into a provenance generating expression. Then it is translated back into SQL and executed by the DBMS. It is worth noting that in case the user does not want to see the individual tuples and their corresponding provenance, I can have the results of a query and provenance information decoupled to show the results’ provenance information in a different table. The current approach is based on PI-CS but the system is extensible enough to include other provenance semantics (such as where provenance).

I describe the query translation and converting in the next two subsections.

### 4.4.1 Query translation

Translating a query from SQL to relational algebra and vice versa is straightforward. The SQL query is parsed into a tree structure allowing for non-sequential access to nodes. Each SQL query is translated into the following format:

$$\sigma_H(\alpha(\Pi(\sigma_C(R_1 \bowtie R_2 \bowtie \ldots R_n)))$$

The $\sigma_H$ refers to the HAVING clause of a query.

Translating the relational algebra expression back to SQL format is also straightforward, each symbol (node in a tree) is translated into a SQL query clause with its respective operators. For sub-queries, recursively constructing sub-queries first then going back to the main query. Currently, sub-queries are only used in translating provenance generating aggregation queries for now.

### 4.4.2 Query converting

Queries are parsed, tagged with their respective types, SPJ or ASPJ, detailing what operations are contained in them, and finally passed to the translator. The translator would convert the query into a relational algebra expression tree according to the query translation scheme in the previous subsection. The tree is then passed to the converter. The converter would transform the query into a provenance generating relational expression tree. To access the complete list of attributes for contributing tables, the system queries the database catalog and gets the list. The list is then injected into the relational expression. In cases where the query transformation
Figure 4.2: Converted ASPJ query

requires completely changing the query structure, such as aggregation, the query tree is split and then reconstructed with the sub-queries. Finally the provenance generating relational algebra tree is passed back to the translator and is translated back to SQL and executed in the underlying DBMS.

Figure 4.2 shows an example of a rewritten aggregation query. The query is restructured into a left join of the original query with a new rewritten query over the group by variables G. The system accesses the database catalog to get the attribute names \( A^+ \) for the GLEI table. The same catalog access operation is used to prevent provenance attributes from appearing in query results that do not require provenance, or prevent them from propagating to new tables.
4.5 User interface

Figure 4.3: Provenance system user interface

Figure 4.3 shows a screen-shot of the main page of my system. The user has the ability to write a query and execute it. Select whether or not to generate provenance information and the type of provenance. In case provenance information is requested, the query results are shown with and without provenance. The provenance generating query is also shown to the user. The user can also view tree visualizations of the original query and how it was transformed. Statistics on the execution time of both regular and provenance queries are also provided.

The user can also go into the visualization component to look at provenance information more closely.

4.6 Future work

The system is designed to be extensible and efficient. However, there are still a number of ways in which the system could be improved. The following are some items I intend to address:

- The current approach is lazy, however I would also like to study propagation
of provenance meta-data throughout a transaction using the same provenance semantics.

- Complete the implementation for all SQL operators to make the system applicable to other case studies and to conduct more experiments.

- Implement other provenance types, Where provenance, How provenance and transformation provenance for different granularities.

- Experiment with capturing different information such as the user or the operation that caused a change in the data.
Chapter 5

Experimental Evaluation

5.1 Performance and overhead

I ran a number of experiments to determine the overhead of my provenance system. I am expecting a difference in performance between running regular queries and provenance generating queries. The difference in performance could be attributed to the extra time taken to process queries and the extra time taken to access the database catalog to acquire table columns for provenance generating queries, or it could be attributed to simply executing longer, more complex queries. I ran my experiments to determine the factors that contribute the most to this overhead.

5.1.1 Experimental setup

Macbook Pro running OS X 10.10.5 with 2.2 GHz Intel Core i7 and 4 GB 1333 MHz Memory. The underlying database management system is PostgreSQL version 9.4. The database driver is Psycopg2 since most of the code is in python.

5.1.2 Data

Currently, as far as I know, there is no real benchmark for provenance systems. Benchmarks such as TPC-H by the Transaction Processing Council [10] can only be used with a complete system, they are geared heavily towards systems with support for subqueries. The goal of such benchmarks is also to stress test the
system, which is not my interest at the moment.

Therefore I ran the experiments on synthetic data instead. The data is generated using the faker (https://pypi.python.org/pypi/fake-factory) python package. The data is composed of tables that can be joined over common attributes and reduced by aggregating by an attribute. I vary the size of tuples of tables between 100 to 100,000. My goal is to measure the time overhead for a number of queries and determine where most of it comes from. The data is generated in a way that ensures that the performance results can be generalized to our typical workloads.

5.1.3 Queries

I ran the following query types: SPJ queries and ASPJ queries. The SPJ queries perform a join on a common attribute, the join type is varied to measure all possibilities. The ASPJ queries aggregate results according to the most common attribute in tuples. I also made sure when writing the queries and generating the data that the queries’ results size in tuples would scale with the size of tables. The queries like the tables are representative of our typical workloads. I ran every single query 100 times on each table size and claculated the mean of the durations.

5.1.4 Results

Figure 5.1 and Figure 5.2 show the results of each query for the varying number of sizes of the input tables. Figure 5.1 shows different results for different queries. The important thing to keep in mind is that the cost of processing the query and accessing the database catalog is trivial compared to the cost of the query execution at larger databases, making my approach of developing this system valid. This is proven by showing the performance of my system without the overhead of query processing and catalog access in Figure 5.2.

In terms of space overhead, the generation of provenance data can have dramatic implications on the size of the results. This is especially true in the case of aggregation queries. To demonstrate, Figure 5.3 shows the different size of results for three different queries running over the GLEI data (I go into detail describing the GLEI data in Chapter 2). The queries are as follows:

q1: SELECT LEI, Address, LegalName, LegalForm FROM GLEI WHERE
Figure 5.1: A line plot of the queries running with overhead of my system measured

Country = 'United States'
Agg q1: SELECT count(LEI) AS count FROM GLEI GROUP BY Country
Agg q2: SELECT count(LEI) AS count FROM GLEI GROUP BY Country, City

This is not something we can generalize, aggregation queries would give different results based on the aggregation group by variables or the data. The implications however are clear that provenance information can be prohibitive with such datasets. The simple SELECT query does not give the full picture either. While the number of tuples remains constant, the number of attributes in the results is much bigger, going from the 4 requested attributes to a full list of attributes from the two contributing tables. This is also true for aggregation queries but the large number of tuples overshadows the increase in attributes size.

Therefore it is absolutely essential to give the user full control over when to generate provenance data. It is also necessary to summarize the results or at least
break them down so the user does not get overwhelmed by the sheer volume of information.

5.1.5 Discussion

The results demonstrate the minimal overhead of query processing for the query types I ran. The results also show that the Perm [16] approach still maintains low overhead even when implemented as an extra layer over the DBMS. I also verified that the only operation causing the overhead is the extra database catalog accesses since the cost of query processing is trivial (linear) at large table sizes (> 10k). However, the results of the size in tuples of the provenance information also raise serious questions about the usability of such information reveals an interesting direction of research where provenance information should be reduced somehow. I propose my provenance explorer component in the next section as part of the
solution where I attempt to give a summarized overview of the data itself so the user can easily navigate through the large volumes of provenance information.
Chapter 6

Provenance Explorer

Provenance information can increase data size exponentially. This can have serious implications not just on storage but also on usability. Furthermore, I would like to enable users to browse through provenance information without being overwhelmed by the size of it. I would also like them to be able to explore smaller portions of the data with its provenance information. Therefore, I believe that the best approach would be to provide an overview or a summary of the data that adheres to visualisations’ limitations and also shows a big picture overview that they can drill down and isolate portions of it and examine the provenance information. This is what I attempt with my provenance explorer component.

Provenance data is represented as annotations or extra attributes attached to the results. However, data often goes through a series of steps to arrive at the final results. Therefore, a more intuitive approach is to show provenance as a connected graph of the derivation history. I use this graph to summarize the trip that a data item took to arrive at the query results. This summary gives an overview of provenance that the user can use to navigate the lineage or derivation history of a portion of the dataset or the entire database.

Data size can overwhelm the user but it can also affect visualisations. In information visualization, visual idioms are limited in terms of size of inputs. Visualizing large graphs or large trees is a challenge and a different research problem than what I am trying to solve. Therefore, my focus is on visualisations that can effectively show an overview without going further into showing a large number of
data items.

In this chapter I look at my provenance explorer component. In Section 6.1 I briefly look at the provenance browsers implemented in other systems. In Section 6.2, I identify a set of tasks users would like to do when interacting with my system. Section 6.3 outlines the visualization principles I followed to design my system. Section 6.4 gives an overview of the currently available functions. In Section 6.5 I look at how the provenance explorer is implemented. To show how a user would use the provenance explorer I present a case study with the GLEI Watch system and data. I explore the results in Section 6.6.

6.1 Provenance browsers

While database provenance propose very limited utilities for visualizing and guiding the user through provenance information, provenance browsers are evident in the field of work-flow provenance. While they bear different semantics that are not as unified as those in database provenance, they still offer insight into how to visualize such data and make it more manageable for the user.

In MyGrid [27] [2], provenance information is captured automatically in provenance logs when a workflow is executed. The logs contains such things as the services invoked, their parameters, and the data products used and derived. Semantic annotations are applied to work-flows and services manually. These annotations are carried over to end data products and used to link relevant pieces together. Using the semantic web browser users can browse through the semantically linked provenance information and reason about provenance related problems. The interface is like a web page with hyper-links to relevant information.

In [2], the authors present a provenance browser that shows the users the process that generated a piece of data. The provenance is represented as a dependency graph of the workflow. The graph can be queried using QLP a graph query language. The provenance is recorded by a scientific workflow system called Kepler [20]. In Kepler, processes that change the data (or actors) take in XML files as input and alter them. These processes and data changes are recorded as provenance traces. The browser enables the users to navigate or write QLP queries to look at different aspects of the workflow graph.
In [9] a database provenance system, a simple visualization of the provenance of a value can be displayed to the user. This visualization shows where the value came from and traces it to the original source. It also shows the transformations or queries that resulted in the final results. I looked at DBNotes in more detail in Chapter 3.

Those systems and others offer insight into what it takes to visualize or just offer different dissemination techniques than simply asking queries and looking at results. However the complexity of the problem in database provenance comes from the fact that provenance information is huge, and most of it is stored in tables, either coupled or decoupled from the data items themselves. In my case it is stored as witness lists with its data. I will look at my approach next and how I handle such a challenge.

**Figure 6.1:** The interface of the provenance browser from [2]
6.2 User tasks

I define two categories of tasks and try to identify tasks that the user would perform using the provenance explorer.

6.2.1 Data related tasks

Since the amount of provenance information is too big for users to browse through, and the data itself can be too large to handle, I offer an initial visualization of aggregated and/or filtered overview of the data. Through this visualization the users can perform the following tasks:

- Aggregate or filter the data by writing SQL queries.
- Filter or search the data further with its respective provenance information.
- Navigate and select individual data items.
- Edit the data, annotate it, or export subsets of it to files along with relevant provenance information.
Figures 6.3 and 6.4 show some of these functions as implemented in the explorer.

**Figure 6.3:** The data visualizations with search, filter and sort options

### 6.2.2 Provenance information tasks

Interacting with the visualization of the data the users can also examine the provenance information and perform any of the following:

- Click on a piece of data to see the provenance information of that piece.
- Drill down to the original source of the data.
- Ask for the provenance to be explained. An overview of the process of generating the provenance information through rewriting queries will be displayed as tree visualizations.
- See where they are in the derivation history of an item through the provenance graph.
- Browse through provenance using a provenance graph that shows all ancestors of a subset of the data.
Figure 6.4: Selecting an individual data item to see its provenance

These tasks can be seen in Figures 6.8, 6.6 and 6.7 showing screenshots from the system.

Figure 6.5: The provenance data tables with the provenance graph
Figure 6.6: Drilling down to the ancestor of the table in Figure 6.5. Clicking on the provenance graph lets the user navigate the derivation history of the displayed provenance table, it also works as a navigation tool letting the user know where they are in the lineage of a data item.

Figure 6.7: A visualization of the query and the rewritten provenance query. Explaining provenance to the users can be done by showing the user the queries and a visualization of the original, and provenance generating relational expressions.
6.3 Visualization principles

My design of the provenance explorer component is guided by identifying a number of user tasks seen in Section 6.2 as well as a set of visualization principles. Here I list a few of those principles that I adhered to to improve the effectiveness of my visualizations:

- Overview first, zoom and filter, then details on demand [24]. This principle is essential for dealing with large scale information. The main goal of my visualizations is to show the user a summarized view of the data (usually an aggregation). The volume of data is usually large enough without introducing at least double the information with provenance information. Having the user browse through an overview made by the user by specifying an aggregation or filtering query. The user can then zoom in and look at specific tuples and look at its provenance information. Provenance information can also exported in small sets in different formats (JSON or CSV).

- Eyes beat memory [21]. Navigating through data tables and drilling down can confuse the user as to where they are in the system. Keeping track of where the user is via visual elements beats having to remember where they are. Hence I try to show a persistent overview, in the form of the provenance graph, that highlights which level the user is currently browsing. This can help keep track of where the user is at all times and limit the cognitive load imposed by trying to remember where they are.

- Multiple views are most effective when explicitly linked[23]. The user is given the option to utilize multiple views at once. Highlighting one element in a certain view (The bar chart for example) would highlight the same item on a different view (country on a map for example).

- I use aggregation and filtering to reduce the data. Visual idioms have limitations in terms of the number of items they can display [21]. Reducing guarantees better scalability for large datasets and more efficient visualizations.
6.4 Overview

The task at hand is to visualize provenance information that is stored in relational tables side by side with their respective tuples. The data itself can be large and overwhelming and the provenance information multiplies the size. The user needs to see the information divided into smaller subsets, and the provenance information should be hidden until the user asks for it.

Thus I offer two components to facilitate browsing through the provenance information: The data visualization and the provenance graph. In the data visualization, the data itself is summarized and the user is presented with a manageable overview. In this overview, the user can click on any data item and see the provenance results, this way the user is only looking at a subset of the information at a time. The provenance graph gives an interactive overview of the derivation history of a piece data and lets the user see where they are in that history. I go into more detail on both in the next subsections.

6.4.1 Data visualization

I enable the visualization of the dataset through the following process: Load the dataset or query the database for results. Filter or aggregate the dataset to get a manageable subset. Visualize the data using one of the following idioms: Treemap, for hierarchical data or aggregated data. A graph for networks data. Geographical map for data containing geographical locations. Bar or line charts for quantitative data. This visualization step is completely optional, data that does not fit any of the above criteria can simply be displayed as a table.

The user can then proceed to examine the visualization or table of data and zoom or select a subset of it. The user can click on any data item to see the provenance results. The user can also drill down on any tuple in the provenance to see its provenance all the way to the original source. Optionally the user can also see the provenance of an entire table. This could be a reasonable approach for smaller datasets.
6.4.2 The provenance graph

The derivation history of a piece of data can contain multiple tables and multiple transformations. The provenance graph representation of this history comes naturally and is similar to previous efforts, such as [9] [2]. The interaction with this representation is to allow the user to browse through the history and display information based on where they click. The graph also works as a navigation guide, letting users know where they are in the derivation history of a data item.

Figure 6.8 shows a provenance graph of the GLEI dataset. The provenance graph shows the original source of the data all the way to the final query results. The graph helps users keep track of where they are in the provenance levels and also enables users to browse through it. The graph persists throughout the visualization and offers an overview that is always there. It is worth noting that the graph only shows the derivation history of data, it does not offer any information on dependencies or relationships between tables or views.

6.5 Provenance Explorer implementation

The provenance explorer is a web based interface to the system. It is powered by my API and D3 [5] visualizations. D3 allows for scalable, efficient, and interactive visualizations. It enables me to attach data to visual elements in the HTML DOM (Document Object Model). I export the data or query results as JSON objects that I later map as DOM elements. I also attach information such as table names and attribute names to HTML elements. This mapping enables me to query and present provenance information for each data item. This enables the user to click on a data item to request its provenance information.

To construct the provenance graph I derive the table names in the history of a piece of data from provenance attributes in tables. Provenance attributes are named
as follows: provenance_tableName_attributeName. The tableName part of the attribute gives the table names to add to the graph. Drilling down to the provenance tableName gives its respective provenance tableName. Constructing the graph is done through queries to the database catalog collecting relevant table names. This is done in the back-end system and the output is sent to the explorer as a JSON tree.

The user can also change a data item to update it. However, the provenance information can be preserved in cases where the original data owners may not want them to be altered.

I also offer the users services that grant the ability to export a subset of the data with its provenance for further analysis or custom visualizations. This data is offered in JSON, CSV or XML format.

6.6 GLEI case study

I looked at the GLEI Watch System and data in detail in Chapter 2. My interest is in the GLEI system as an example of a curated database. Users can look at the data, clean it, modify it or annotate it as they see fit. The provenance information should be preserved as it comes from the original system. Maintaining provenance information would guarantee changes to the data can be verified by future users. I am also interested in guaranteeing to users that they can trust the data presented by the GLEI Watch system, convincing them that it has not been altered in ways that lead to errors and that no important data items were dropped or erased. Allowing the users to inspect the original and intermediate sources can ensure that.

In addition to the tasks highlighted in Section 6.2, the GLEC Watch system is intended to answer the following queries:

- What are the cities in which the most registrations occur?
- What are the cities with the highest number of registrations?
- Which LOUs have the highest number of registrations?
- What is the pattern of registrations over time by country and by city and by registration authority?
• What is the proportion of types of legal entities (legal forms) being registered by country, by city and by registration authority?

• What companies have been registered?

• Where are the companies registered?

Providing an overview of the data on the shape of a geographical map is the natural way to represent this information and answer some of those queries. Most of the queries are also aggregation queries. I offer the user an overview of the data as a geographical map of countries that contain registration numbers.

I also offer a linked view of a bar-chart showing the number of registrations in each country. The barchart and map views are linked. Hovering over a bar-chart item highlights the country on the map. Clicking on a country shows the provenance results. It explains the number of companies in that country by displaying provenance information of the aggregation operation. The users can drill down further all the way to origin of any data item.

Figure 6.9: A world map with aggregated count of registrations in a country

Alternatively, to answer queries 2, 3, and 4 the user could opt for a count aggregation by city, LOU data source, or legal form. In such a case, the user could opt for a treemap representation of this aggregation data. The provenance infor-
Figure 6.10: A barchart showing aggregated count of registrations in a country

Figure 6.11: A treemap with aggregated count of registrations in a country

The other queries posed involve detecting patterns and trends in data. Using line-charts to show patterns in time-series data is standard practice. Enhanced by provenance information the user can see where every data item comes from and trace it all the way to the origin.

The purpose of applying the provenance explorer to the GLEI data is to ensure that the users can trace every change that has been applied to the data. The
GLEI watch system already guarantees freely available data, visualisations and total transparency. Provenance information adds an extra layer of transparency showing the users every step that was taken by the system owners to alter the data.
Chapter 7

Conclusions

In this thesis, I have looked extensively at provenance research in the database field. My findings are summarized as a classification scheme of provenance approaches. I identify the most desirable features in a provenance system and list them out.

I also developed a system based on my findings on top of a relational DBMS. I show the minimal overhead of my approach, making it ideal for the experiments I carry out and a good fit for the GLEI Watch system.

I used this system to develop a provenance explorer, a system to visualize query results and explore provenance information. I used the provenance explorer on a curated dataset of financial data to demonstrate how it works and give a case study to show its uses. I believe that working on a solution that makes provenance information more manageable for users fills a gap in this area and is a valid contribution to this field of research.

There is still a lot to be done in this field before a single approach is deemed complete. The size of the provenance data is still too large to handle, and scalability is a real concern. I feel that solutions in the vein of my provenance explorer are a step in the right direction. I intend to extend my system and my visualizations to support more general use cases. I also intend to validate my system through a thorough quantitative user study in the future. Other ideas for extensions include assisting the user in formulating queries that summarize the data, limiting the attributes to use in provenance generation, and extending query rewrites to support other things such as guaranteed querying of only trusted sources or including only
trusted sources in provenance information.
Bibliography


