# Computing Provenance Using the Negated Chase

Andreas Görres[1,*],
Supervised by Prof. Andreas Heuer[1]

[1]University of Rostock, Universitätsplatz 1, 18055 Rostock, Germany

Abstract
Since different challenges of data processing are interconnected, we describe them in a unified manner using a classic algorithm of database theory: the Chase. Computing the origin of query results is one of the challenges considered in this research project. Previously, the Chase has been used to calculate why-provenance of simple conjunctive queries. However, applying the Chase to more realistic scenarios requires an extension of the algorithm, for example with negation. This work reveals opportunities for the extended Chase by calculating both why- and why-not provenance of conjunctive queries with negation.

**Keywords**
Chase, negation, why-provenance, why-not provenance, data science pipeline

## 1. Introduction

When processing large amounts of data in a systematic fashion, we usually solve the arising issues with algorithms tailored towards the specific challenge. Even though this strategy leads to increased efficiency on a local level, we miss connections between the existing problems. For instance, privacy and provenance are contradictory requirements usually solved in isolation from each other.

If we describe data processing with the concept of the data science pipeline, privacy and provenance can be mapped to individual steps of the pipeline, but at the same time, they are requirements for the other steps. Therefore, we need to consider them while designing the database schema, schema evolution, cleaning the data, transforming the queries and while processing the results of data analysis. Formulating separate issues with a single consistent language makes this possible. In our research, we chose the language of the Chase algorithm.

The Chase $P(O)$ integrates a parameter $P$ into an object $O$, in the classical applications using integrity constraints as $P$ and database schemas or queries as $O$. The algorithm was introduced more than four decades ago, processing two seemingly unrelated use cases – query optimization and schema construction – in a unified way [1, 2]. In the following years, the number of its application areas increased even further, with the concept of "universal solution" connecting the different challenges [3]. Since then, intense research lead to a deeper understanding of the algorithm's properties, for example its termination behavior. Despite its success in the field of theory, software tools making use of the algorithm are – for the most part – restricted to scientific prototypes. Ultimately, we intend to use the Chase to solve practice-related use cases, in particular, issues of privacy and provenance.

In this work, we focus on our results concerning the why- and why-not provenance. Previous studies explored Chase based solutions concerning the why-provenance of simple queries. However, for more realistic scenarios, extensions of the algorithm are necessary. Unfortunately, this endangers confluence, termination and efficiency of the Chase. While we regard the Chase as a universal algorithm targeting a broad variety of objects, semantics of its extensions depends on the Chase object and therefore the use case. Privacy issues can be solved with the Chase on queries, whereas provenance computations require the Chase on database instances, which is therefore the focus of this work.

### 1.1. Data Science Pipeline

Data processing can be divided into a sequence of steps we call the data science pipeline. The initial step comprises schema evolution, data migration and data cleaning. The subsequent data analysis is abstracted as a sequence of database queries. Finally, results are interpreted and issues of privacy and provenance are tackled. In the following, we will take a closer look at the individual steps, highlighting contributions of the Chase algorithm. For data cleaning, Chase parameter $P$ are integrity constraints (e.g. key constraints) and Chase object $O$ is the database instance. The Chase might substitute null values with constants by making use of functional dependencies or insert missing tuples, e.g. to satisfy inclusion dependencies. Data exchange (and, in a similar manner, schema evolution) takes data from a source in-

*VLDB 2023 PhD Workshop, co-located with the 49th International Conference on Very Large Data Bases (VLDB 2023), August 28, 2023, Vancouver, Canada*
*Corresponding author.
andreas.goerres@uni-rostock.de (A. Görres)

stance to generate a target instance under a different schema. Again, $O$ is a database instances, whereas $P$ are rules describing a mapping from source to target schema. For data analysis, $O$ is again a database instance, while $P$ represents the database queries. Here, our interest in complex data analyses comprising e.g. statistical analysis warrants an extension of the current Chase formalism, for example with mathematical operators or negation. In the provenance step, data responsible for the analysis results is identified using provenance techniques. This way, reproducibility of those results is guaranteed. To compute provenance, we invert the Chase rules used in the previous data analysis step. Here, $O$ refers to the achieved analysis results. However, privacy guidelines might prohibit direct access to the raw data, for example when personal information is concerned. We contribute to privacy by applying Chase rules (e.g. view definitions) to the queries used in the analysis step, thereby integrating them into the latter. By combining different Chase applications, we contribute to every step of the data science pipeline.

### 1.2. Research Data Management

One of the real world application areas motivating our study is research data management. Many of our use cases originate from our long term co-operation with the Institute for Baltic Sea Research (IOW). Here, reproducibility of published research results is a requirement for good scientific conduct. However, the schema of recorded data changes over time. Thus, while tracing back results to the responsible data, we need to account for schema evolution. After identifying involved tuples, scientists provide them – and not the entire data set – to an external reviewer.

### 1.3. Negation in Data Analysis

We interpret data analysis as a series of database queries, which in turn are expressed as Chase rules. In particular, we are interested in statistical analysis of scientific data which often contains negation. On the one hand, negation might be explicit part of the analysis algorithm, for example in the form of set difference. On the other hand, negation can be implicit part of basic aggregate functions, for example the maximum function. However, negation is not part of the standard Chase.

### 1.4. Contribution

Using the Chase algorithm, we solve interconnected challenges of the data science pipeline in a coordinated manner. In this work, we study how previous results concerning provenance calculation are affected if we extend the Chase with negation. While the Conditional Chase we describe here has been studied in previous theoretical works [4], connecting it to Chase negation is rather uncommon. Furthermore, this extension allows the calculation of instance-based why-not provenance. While the computation of this provenance is already possible with specialized algorithms, our Chase based solution is directly integrated into a framework solving a multitude of other data science challenges, for example schema evolution and query transformation.

## 2. State of the Art

The Chase is a fixpoint algorithm incorporating a set of rules, the Chase parameter, into a Chase object. In this work, the Chase parameter is a query or transformation rule encoded as a tuple generating dependency (tgd), while the Chase object is a database instance. Tgds are logical implications of the form $\phi(\vec{x}, \vec{y}) \rightarrow \exists \vec{z} : \psi(\vec{x}, \vec{z})$. Here, $\phi$ and $\psi$ are sets of relational atoms over the depicted variables. If the tgd encodes a query, we often existentially quantify $\vec{y}$, while at the same time prohibit existentially quantified variables $\vec{z}$ in the rule head (its right-hand side). The inversion of a tgd is the logical implication $\psi(\vec{x}, \vec{z}) \rightarrow \exists \vec{y} : \phi(\vec{x}, \vec{y})$.

If there is a homomorphic mapping from a Chase rule's body (the left-hand side) to the Chase object and the rule is not satisfied yet, the rule head is materialized under the homomorphism, generating a set of new tuples. Each existentially quantified variable of the head contributes to a fresh marked null values $\eta_i (i \in \mathbb{N})$. The Chase continues until a fix point is reached, however, termination is not guaranteed if existentially quantified variables in the rule head are allowed and the rules are cyclic. Still, several termination tests guarantee Chase termination even on cyclic rule sets. The Conditional Chase described later in this work for instance terminates on richly acyclic rule sets in polynomial time [4].

In general, provenance can be seen as meta-data describing a production process [5]. For our generalizing research approach, why- and why-not data provenance are of particular interest. While there are tools (e.g. [6]) computing more detailed provenance information, they are not applicable to other challenges of the data science pipeline.

Why-provenance provides tuples – the witnesses – justifying a certain result. Quite often, for example if tuples become indistinguishable after projection, there are alternative sets of tuples witnessing the same result. Thus, a witness basis is the set of all sets of witnesses. Alternatively, inverting the query might provide a single generic representation for alternative witness sets (compare the relaxed Chase-inverse in [7]).

Why-not provenance explains the absence of expected result tuples. This can take three different forms:

Instance-based explanations, query-based explanations and refinement-based explanations [5]. Since the Chase algorithm allows neither tuple deletion nor update of constants, we restrict ourselves to explanations based on the insertion of tuples into the database. Similar to our approach, the algorithm described in [8] computes why-not provenance using conditions and c-tables. However, those concepts are not used in the context of the Conditional Chase discussed in this work. Most importantly, this solution is restricted to provenance and isolated from other challenges of the data science pipeline.

Even though [9] formalizes both evolution rules and conjunctive queries using tgds, only basic analysis operations can be realized this way. Furthermore, only non-recursive queries are actually inverted using the Chase.

# 3. Implementation

With the Chase implementation ChaTEAU, different applications of the Chase are combined in a in a single toolkit [9]. Currently, extensions like generalized negation on database instances and queries under the conditional semantics discussed in this work are integrated into the software.

# 4. Negation on Instances

While we regard the Chase as a universal algorithm handling different kinds of parameters and objects in a unified manner, semantics of negation still depends on the Chase object. For the Chase on instances, negation in a Chase rule requires the absence of a certain set of tuples. In contrast, negation in Chase rules applied to queries requires the presence of an explicitly negated set of atoms in the query. As a consequence, we differentiate between negation as a negated boolean subquery (from integrity constraint to database instance), and negation as a boolean subquery with inverted direction (from query to integrity constraint). In this work, we will focus on the first case, since it is more relevant the use case data provenance. After finding a term mapping $h$ for all variables $\vec{x}$ of the positive body of an integrity constraint, we select the atoms $\phi(\vec{x}, \vec{y})$ of a negative body $\neg\exists\vec{y} : \phi(\vec{x}, \vec{y})$ (that is, a negated conjunction of relational atoms). If the result of the boolean query $\phi(\vec{x}, \vec{y}) \rightarrow ()$ is $()$, we reject $h$, otherwise, we continue with the next negative body or complete the Chase step (e.g. generate some tuples).

In this work, we will focus on semi-positive negation, that is, negation of base relations. Otherwise, we can no longer guarantee a single generic witness base and calculations become rather complicated.

# 5. Conditions and Certain Answers

Standard Chase (without negation) operates under certain answers semantics. Interpreting (marked) null values as unknown, but existing values, we consider only results justified under any valuation of null values with concrete constants. In general, we treat null values as constants unequal to any constant in the database instance. However, this naive interpretation is insufficient for negation under certain answers semantics. Consider rule $r_1$ which is triggered if null value $\eta_1$ equals constant $c$. Under naive interpretation, $\eta_1$ is not equal to $c$ and no result is generated. Let there be a second rule $r_2$ which would be blocked by $r_1$'s result. Clearly, $r_2$ should not be triggered under certain answers semantics since there is a valuation of null values ($\eta_1 \mapsto c$) not justifying the generation of this tuple. Therefore, some tuples are only generated under certain conditions. For this paper, we define conditions as conjunctions of logical comparisons $x_1\theta x_2$ ($\theta \in \{=, \neq\}$, $x_i \in \mathrm{CONST} \cup \mathrm{NULL}$), with $CONST$ being the set of all constants of the domain and NULL being the set of all marked null values. In the previous example, the blocking tuple would be generated under condition $\eta_1 = c$, while the result of $r_2$ is generated under condition $\eta_1 \neq c$. If conditions of equivalent tuples form a tautology (e.g. $\eta_1 = c$ and $\eta_1 \neq c$), those tuples exist under certain answers semantics.

# 6. Provenance

## 6.1. Why Provenance

As mentioned before, we can use the standard Chase algorithm (without negation) to calculate the why-provenance of query results. The notion of why-provenance used here corresponds to the relaxed Chase-inverse found in [7].

For this, we invert the Chase rules that created the result by switching the rule's head and body. Attribute values not passed to the result correspond to existentially quantified variables of the inverted Chase rule. Applying the inverted query to the result generates the minimal witness basis [9]. This set of tuples is sufficient to create the original query result. However, it is usually smaller than the complete instance and contains marked null values in places irrelevant for the query.

Extending this established algorithm with semi-positive negation is often without consequences. However, the same witness might play different roles during query execution. Consider query $q$ on instance $I$:

$$I = \{R(1), R(2), S(2)\}$$
$$q : R(x) \wedge R(y) \wedge \neg S(y) \rightarrow (x, y).$$

If we ignore the semi-positive negation of $S$, the witness basis is $\{R(1), R(2)\}$. Indeed, even if we materialize

the image of $q$'s negative body, we only learn that $S(1)$ cannot exist, but we learn nothing about $S(2)$. Consequently, we can justify the existence of all previously published results $\{(1,1),(2,1)\}$, but we could additionally justify the result $(2,2)$. The absence of this expected result from the published result could be explained using why-not provenance. However, the instance-based why-not provenance presented in this work is restricted to insertions – clearly, there is no way to generate $(2,2)$ by inserting additional tuples into the database.

### 6.2. Why-not Provenance

Similarly to why-provenance, why-not provenance can be computed using the witness basis generated by the Chase. This witness basis (in the context of why-not provenance known as "'generic"' witness basis) includes the query's materialized negative bodies. Since we are interested in witnesses without representation in the database, we insert artificial representatives for each (positive) witness into the database. We interpret the witness basis as a query returning the tuple identifiers and apply it to the database. Every generated result tuple corresponds to one possible why-not explanation. We select results referring to a minimum of artificial representatives and return the respective representatives as the why-not explanation. If a witness mapped to its own artificial representative and a witness mapped to a tuple from the original database share an existentially quantified variable, this variable is mapped to a marked null value (from the artificial tuple) and a constant (from the original tuple). The Conditional Chase allows this mapping under the condition that null value and constant are equal. If those equality conditions are consistent, we interpret them as term mappings and substitute null values from explanation tuples with constants from the original database instance. A more detailed description of the algorithm outlined above can be found in [10].

## 7. Future Work

Currently, we only consider semi-positive negation. Otherwise, two queries triggering each other could lead to nested negation in the generic witness basis, which resolves to a disjunction of generic witnesses. However, the use cases motivating our work are not restricted to semi-positive negation or even stratifiable rule sets, so an extension of our framework is necessary.

In this work, the Chase object is a database instance. However, other steps of the data science pipeline require a query to be the Chase object. While Chase semantics are very similar for both types of objects, semantics of negation differ distinctively.

## 8. Conclusion

The Chase algorithm solves a multitude of data science challenges in a unified manner. Provenance, for instance, can be calculated while keeping track of schema evolution. However, real world applications require extensions of the algorithm. In this work, we explain computation of why- and why-not provenance of conjunctive queries with semi-positive negation using the extended Chase. While we chose the Conditional Chase to realize certain answers semantics with negation, this decision was advantageous for the explanation of why-not provenance even in the absence of negation.

## Acknowledgments

This work was supported by a scholarship of the Landesgraduiertenförderung Mecklenburg-Vorpommern.

## References

[1] A. V. Aho, Y. Sagiv, J. D. Ullman, Efficient optimization of a class of relational expressions, ACM Trans. Database Syst. 4 (1979) 435–454.

[2] D. Maier, A. O. Mendelzon, Y. Sagiv, Testing implications of data dependencies, ACM Trans. Database Syst. 4 (1979) 455–469.

[3] A. Deutsch, A. Nash, J. B. Remmel, The chase revisited, in: PODS, ACM, 2008, pp. 149–158.

[4] G. Grahne, A. Onet, On conditional chase termination., AMW 11 (2011) 46.

[5] M. Herschel, R. Diestelkämper, H. Ben Lahmar, A survey on provenance: What for? what form? what from?, VLDB J. 26 (2017) 881–906.

[6] P. Senellart, L. Jachiet, S. Maniu, Y. Ramusat, Provsql: Provenance and probability management in postgresql, Proc. VLDB Endow. 11 (2018) 2034–2037.

[7] R. Fagin, P. G. Kolaitis, L. Popa, W. C. Tan, Schema mapping evolution through composition and inversion, in: Schema Matching and Mapping, Data-Centric Systems and Applications, Springer, 2011, pp. 191–222.

[8] M. Herschel, M. A. Hernández, Explaining missing answers to SPJUA queries, Proc. VLDB Endow. 3 (2010) 185–196.

[9] T. Auge, M. Hanzig, A. Heuer, Prosa pipeline: Provenance conquers the chase, in: ADBIS (Short Papers), volume 1652 of *Communications in Computer and Information Science*, Springer, 2022, pp. 89–98.

[10] A. Görres, A. Heuer, Computing Provenance Using the Negated Chase, Technical Report CS 01-23, Institut für Informatik, Universität Rostock, 2023. Extended version of this work.