Datalog and Recursive Query Processing

Todd J. Green
LogicBlox Inc.
todd.green@logicblox.com

Shan Shan Huang
LogicBlox Inc.
ssh@logicblox.com

Boon Thau Loo
University of Pennsylvania
boonloo@cis.upenn.edu

Wenchao Zhou
Georgetown University
wzhou@cs.georgetown.edu
Editorial Scope

Topics

Foundations and Trends® in Databases covers a breadth of topics relating to the management of large volumes of data. The journal targets the full scope of issues in data management, from theoretical foundations, to languages and modeling, to algorithms, system architecture, and applications. The list of topics below illustrates some of the intended coverage, though it is by no means exhaustive:

- Data models and query languages
- Query processing and optimization
- Storage, access methods, and indexing
- Transaction management, concurrency control, and recovery
- Deductive databases
- Parallel and distributed database systems
- Database design and tuning
- Metadata management
- Object management
- Trigger processing and active databases
- Data mining and OLAP
- Approximate and interactive query processing
- Data warehousing
- Adaptive query processing
- Data stream management
- Search and query integration
- XML and semi-structured data
- Web services and middleware
- Data integration and exchange
- Private and secure data management
- Peer-to-peer, sensornet, and mobile data management
- Scientific and spatial data management
- Data brokering and publish/subscribe
- Data cleaning and information extraction
- Probabilistic data management

Information for Librarians

Foundations and Trends® in Databases, 2012, Volume 5, 4 issues. ISSN paper version 1931-7883. ISSN online version 1931-7891. Also available as a combined paper and online subscription.
Datalog and Recursive Query Processing

Todd J. Green  
LogicBlox Inc.  
todd.green@logicblox.com

Shan Shan Huang  
LogicBlox Inc.  
ssh@logicblox.com

Boon Thau Loo  
University of Pennsylvania  
boonloo@cis.upenn.edu

Wenchao Zhou  
Georgetown University  
wzhou@cs.georgetown.edu
# Contents

1 Introduction 2
   1.1 Contributions and Roadmap 4
   1.2 Relationship with Previous Surveys 5
   1.3 First Example: All-Pairs Reachability 6

2 Language and Semantics 9
   2.1 Language 9
   2.2 Semantics 11
   2.3 Negation 16
   2.4 Aggregation 22

3 Recursive Query Processing 27
   3.1 Bottom-up Evaluation 28
   3.2 Top-down Evaluation 31
   3.3 Magic Sets 37

4 Incremental Maintenance 47
   4.1 Counting Algorithm for Non-recursive Queries 48
   4.2 Delete and Re-Derive Algorithm (DRed) 49
   4.3 Provenance-based Incremental Maintenance 51
   4.4 Incremental Maintenance for Negation and Aggregates 51
5 Datalog Extensions 53
5.1 Beyond Stratified Negation 53
5.2 Beyond Stratified Aggregation 55
5.3 Arithmetic and Infinite Relations 57
5.4 Functors 59
5.5 States and Updates 61

6 Applications 65
6.1 Program Analysis 65
6.2 Declarative Networking 68
6.3 Data Integration and Exchange 71
6.4 Enterprise Software 76
6.5 Other Applications 79

Acknowledgements 82

References 83
Abstract

In recent years, we have witnessed a revival of the use of recursive queries in a variety of emerging application domains such as data integration and exchange, information extraction, networking, and program analysis. A popular language used for expressing these queries is Datalog. This paper surveys for a general audience the Datalog language, recursive query processing, and optimization techniques. This survey differs from prior surveys written in the eighties and nineties in its comprehensiveness of topics, its coverage of recent developments and applications, and its emphasis on features and techniques beyond “classical” Datalog which are vital for practical applications. Specifically, the topics covered include the core Datalog language and various extensions, semantics, query optimizations, magic-sets optimizations, incremental view maintenance, aggregates, negation, and types. We conclude the paper with a survey of recent systems and applications that use Datalog and recursive queries.

DOI: 10.1561/1900000017.
Mainstream interest in Datalog in the database systems community flourished in the eighties and early nineties. During this period, there were several pioneering Datalog systems, primarily from academia. Two of the more prominent ones with complete implementations include Coral \cite{99} and LDL++ \cite{20}. Some ideas from these early research prototypes made it into mainstream commercial databases. For instance, Oracle, DB2, and SQL Server provide support for limited forms of support for recursion, based on SQL-99 standards. However, a perceived lack of compelling applications at the time \cite{113} ultimately forced Datalog research into a long dormancy, and stifled its use in practice. Coral and LDL++ ceased active development in 1997 and 2000 respectively, and commercial systems did not extend a limited form of Datalog.

In recent years, however, Datalog has reemerged at the center of a wide range of new applications, including data integration \cite{68, 43, 50}, declarative networking \cite{80, 77, 75}, program analysis \cite{29}, information extraction \cite{110}, network monitoring \cite{10}, security \cite{85, 60}, optimizations \cite{73}, and cloud computing \cite{15, 16}. Compared to the state-of-the-art of two decades ago, the modern systems that drives these emerging applications have significantly more mature and complete Datalog im-
plementations, and often times deploy applications that are orders of magnitude larger in code size and complexity compared to the older generation of Datalog programs.

In terms of modern academic systems, the IRIS reasoner [59] is an open-source general purpose Datalog execution engine with support for optimizations, stratified and locally stratified negation. There are also publicly available Datalog systems tailored for specific applications. These include the Orchestra system for collaborative data sharing [92], BDDDBDD [24] for program analysis, the RapidNet [101] declarative networking platforms, and the Bloom [16] platform for declarative programming in the cloud.

In the commercial world, a major development is the emergence of enterprise Datalog systems, most notably LogicBlox [4], Datomic [2], Semmle [7], and Lixto [49]. Semmle and Lixto are targeted at specific domains of program analysis and information extraction respectively, while LogicBlox and Datomic aim to provide a general platform for developing enterprise software.

The revival of Datalog in the new generation of applications is driven by the increasing need for high-level abstractions for reasoning about and rapidly developing complex systems that process large amounts of data, and are sometimes distributed and parallel. Datalog provides a declarative interface that allows the programmer to focus on the tasks (“what”), not the low-level details (“how”). A common thread across these systems is the use of the Datalog language as a declarative abstraction for querying graphs and relational structures, and implementing iterations and recursions. Its clear and simple syntax with well understood semantics aims to achieve the best of both worlds – having a rich enough language to support a wide range of applications, yet at a high and concise level that makes rapid prototyping easy for programmers without having to worry about low level messy details related to robustness and parallelism. The high-level specifications also make code analysis easier, for applying optimizations and for reasoning about transactions and safety.
1.1 Contributions and Roadmap

This survey paper aims to provide an accessible and gentle introduction to Datalog and recursive query processing to readers with some basic background in databases (in particular, SQL and the relational model). Given the wide range of research literature on Datalog spanning decades, we identify a “practical” subset of Datalog based on recent advances in the adoption of Datalog. In particular, our survey aims to cover the following:

- **Language.** Core Datalog syntax and semantics. (Chapter 2)

- **Query processing.** Recursive query processing techniques for executing Datalog programs efficiently, using the bottom-up and top-down evaluation strategies, such as the well-known semi-naïve [22] and Query/Subquery (QSQ) [67] evaluation strategies. (Chapter 3)

- **Incremental maintenance.** Extensions to query processing techniques in the previous chapter, to include mechanisms for incrementally updating the materialized views of a Datalog program, as the input data changes, without having to recompute the entire Datalog program from scratch. (Chapter 4)

- **Common extensions.** Each application domain takes the core Datalog language and then further customizes and extends the core language and implementation techniques to meet its particular needs. Here, we discuss extensions to incorporate negation, aggregation, arithmetic, uninterpreted functions, and updates, as well as the query processing techniques to handle these extensions. (Chapter 5)

The survey concludes in Chapter 6 with a brief survey of recent applications of Datalog, in the domains of program analysis, declarative networking, data integration and exchange, enterprise software systems, etc.
1.2. Relationship with Previous Surveys

Our survey serves as an entry point into several other survey papers and books on Datalog. We briefly mention some of them:

- Bancilhon et al. [23] surveys and compares various strategies for processing and optimizing recursive queries in a greater depth compared to our survey.

- Ceri et al. [32] presents the syntax and semantics of Datalog along with evaluation and optimization techniques for efficient execution. Extensions to the Datalog language, such as built-in predicates and negation are also discussed.

- Ramakrishnan and Ullman [100] provides a high-level overview of the Datalog language, query evaluation and optimizations, and more advanced topics on negation and aggregation in a few pages. This should be viewed as a “quick-starter” guide for someone exposed to Datalog for the first time.

- Textbooks [12, 27, 33, 118, 36] cover some topics (e.g. language, semantics, magic sets) in greater detail than our survey. Abiteboul et al. [12] in particular is a widely used textbook geared towards a database theory audience.

Overall, our survey is broader than Bancilhon [23], which focuses primarily on query processing, and Ramakrishnan and Ullman [100], which surveys Datalog systems (which are now more than a decade old) with a brief discussion on query processing and optimizations. We cover a breath of topics similar to the surveys [32, 88], but provide significantly more details on systems issues related to query processing, incremental maintenance, and modern applications. Compared to all of the above surveys, we provide a more systems approach in presentation of classical topics, and discuss only extensions relevant to modern applications.


1.3 First Example: All-Pairs Reachability

We begin with a high level introduction to the Datalog language and its basic evaluation strategy. As our first example, we consider a Datalog program that computes all-pairs reachability, essentially a transitive closure computation in a graph for figuring out all pairs of nodes that are connected (reachable) to each other.

\[
\begin{align*}
    r1 & \text{ reachable}(X,Y) :- \text{link}(X,Y). \\
    r2 & \text{reachable}(X,Y) :- \text{link}(X,Z), \text{reachable}(Z,Y). \\
    \text{query}(X,Y) & :- \text{reachable}(X,Y).
\end{align*}
\]

The above two rules, named as \textit{r1} and \textit{r2}, derive the \textit{reachable} nodes (i.e. \textit{reachable}(X,Y)) using facts about directly linked nodes (i.e. \textit{link}(X,Y)). Here, we use capital letters \(X\) and \(Y\) to signify that they are variables in the domain of all the nodes. The output of interest in this program, as denoted by the special predicate \textit{query}(X,Y), is the set of derived \textit{reachable} facts. The input graph in this case can represent a network of routers, and forms a basis for implementing network routing protocols [80], web crawlers [81], and network crawlers [79].

Rule \textit{r1} expresses that node \(X\) is reachable from \(Y\) (i.e. \textit{reachable}(X,Y)) if they are directly linked. Rule \textit{r2} is a bit more interesting, as it specifies the \textit{reachable} relation in terms of itself: \((X,Y)\) are reachable from one another if \(X\) has a direct link to a node \((Z)\) that is reachable to \(Y\). We refer to rules such as \textit{r2} as \textit{recursive} rules, since the \textit{reachable} relation appears in both the rule body (right of “ \(:=\)”) and head (left of “ \(:=\)”). Rule \textit{r2} is also a \textit{linear} recursive rule, since \textit{reachable} appears only once in the rule body.

\[\text{Figure 1.1: Example graph used for reachability computation.}\]

We illustrate the execution of Datalog rules by evaluating the \textit{reachable} rules over the graph shown in Figure 1.1, which depicts a network consisting of three nodes and four direct \textit{links}. Thus, there are
1.3. First Example: All-Pairs Reachability

(initial base tuples)  (iteration 1)  (iteration 2)  (iteration 3)

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

Reachable

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

New tuples derived in the current iteration that are not known in prior iterations are shaded.

Figure 1.2: Tuples derived by the All-pairs Reachability program for each iteration. New tuples derived in the current iteration that are not known in prior iterations are shaded.

Four initial entries (tuples) in link: link(a, b), link(b, c), link(c, c), and link(c, d).

Intuitively, rule evaluation can be understood as the repeated application of rules over existing tuples to derive new tuples, until no more new tuples can be derived (i.e. evaluation has reached a fixpoint). Each application of rules over existing tuples is referred to as an iteration. This evaluation strategy is often times referred to as the naïve evaluation strategy.

The evaluation of the reachability rules over the network in Figure 1.1 reaches a fixpoint in three iterations, as shown in Figure 1.2. In iteration 1, rule r1 takes as input the initial link tuples, and uses that to generate 4 reachable tuples. These tuples essentially represent all pairs of nodes reachable within one hop. In the next two iterations, all reachable tuples generated in previous iterations are used as input to rule r2 to generate more reachable tuples that are two and three
hops apart. Iteration 4 (not shown in the figure) derives the same set of tuples as iteration 3, and hence, a fixpoint is reached. Given that no two nodes are separated by more than 3 hops, the recursive query completes in 4 iterations.

As an optimization, instead of using all derived facts as input to rules at each iteration, one can suppress the evaluation that uses only tuples already learned in prior iterations when computing new tuples the next iteration. For instance, when generating new facts in iteration 3, rule \texttt{r2} will not evaluate for inputs \texttt{reachable(b,c)} and \texttt{link(a,b)}, since they have already been used in iteration 1. The intuitive description above corresponds loosely to the \textit{semi-naïve} evaluation strategy, which will be described in greater detail in Chapter \ref{chapter3}.

Note that the above approach is a \textit{bottom-up} evaluation technique, where existing facts are used as input to rule bodies to derive new facts. A fixpoint is reached when no new facts are derived. This is also known as a \textit{forward-chaining} style of evaluation. An alternative approach used in Prolog \cite{112} uses a goal-oriented \textit{backward-chaining} approach, starting from the goal (i.e. query), and then expanding the rule bodies in a top-down fashion.

A top-down approach allows for an evaluation strategy that focuses only on facts necessary for the goal. However, a bottom-up evaluation approach used in Datalog allows us to draw upon a wealth of query processing and optimization techniques to draw upon for doing the computations efficiently even when datasets are too large to fit in main memory. Moreover, as we show in Section \ref{section3.3} query optimization techniques can optimize Datalog programs for bottom-up evaluation, to avoid deriving facts not relevant to answering queries.
References

References


[24] BDD-Based Deductive DataBase. \url{http://bddbddb.sourceforge.net/}


[28] Martin Bravenboer and Yannis Smaragdakis. Doop website, \url{http://doop.program-analysis.org/}.

References


References


References


References


References


