

# Cardinality Estimates for Recursively Defined Relations in Datalog

Christian Fattebert<sup>1</sup>, Zhekai Jiang<sup>1,\*</sup>, Christoph Koch<sup>1</sup>, Reinhard Pichler<sup>2</sup> and Qichen Wang<sup>3</sup>

<sup>1</sup>EPFL, Switzerland

<sup>2</sup>TU Wien, Austria

<sup>3</sup>Nanyang Technological University, Singapore

## Abstract

Cardinality estimates are crucial for cost-based query optimization. However, in datalog, where relations can be defined recursively, appropriate techniques for cardinality estimates are missing. In this work, we present a framework for adorning datalog rules that allows us to compute (asymptotic) bounds on the result sizes of recursive datalog programs.

This is an extended abstract of a paper published at PODS 2026 [1].

## Keywords

datalog, adornments, cardinality estimates

## 1. Introduction

Datalog engines are starting to show their potential in a wide range of data-intensive applications. With ever-growing data sizes, cost-based optimization is essential for fast, scalable execution in such engines. However, accurate cost-based optimization relies heavily on the ability to estimate the cardinalities of intermediate results generated during query evaluation [2]. In traditional join processing, there have been significant research efforts on cardinality estimation [3] and cost-based optimization [4]. But estimating result sizes is significantly more challenging when recursion is involved, which is an inherent feature of datalog. In particular, there is currently no reliable method for computing size bounds on the final outputs of recursively defined relations, except for a few very specific datalog programs [5]. Without such bounds, optimizers are often forced to rely on heuristics, as is the case for the popular datalog engine Soufflé [6]. Such methods often lead to suboptimal execution plans or higher runtime overhead for complex workloads.

In this paper, we present techniques for computing (asymptotic) bounds on the result sizes of recursive datalog programs. More specifically, we introduce a framework for adorning datalog rules (and specifically IDB atoms) with conjunctive queries over the input database, the union of which subsumes the result of the IDB predicate, thus acting as a form of upper bound. We refer to such adorned programs as *EDB-bounded datalog programs*. We give a simple illustration with a variant of the well-known transitive closure program:

**Example 1.1.** Consider the datalog program  $\Pi$  with the following rules:

$$\begin{aligned} R_1 : TC(\bar{x}, \bar{y}) &\leftarrow e(\bar{x}, \bar{y}), \\ R_2 : TC(\bar{x}, \bar{y}) &\leftarrow TC(\bar{x}, \bar{z}), e(\bar{z}, \bar{y}), \end{aligned}$$

where  $e$  is an EDB relation, each of  $\bar{x}$ ,  $\bar{y}$ , and  $\bar{z}$  is a vector of  $m$  variables, and  $e$  and  $TC$  hence have arity  $2m$ . This program is unbounded, as the depth of recursion depends on the size of the input. Therefore,

*Datalog 2.0 2026: 6th International Workshop on the Resurgence of Datalog in Academia and Industry, September 7, 2026, Klagenfurt, Austria*

\*Corresponding author.

✉ christian.fattebert@epfl.ch (C. Fattebert); zhekai.jiang@epfl.ch (Z. Jiang); christoph.koch@epfl.ch (C. Koch); reinhard.pichler@tuwien.ac.at (R. Pichler); qichen.wang@ntu.edu.sg (Q. Wang)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

there does not exist a finite non-recursive program (or, finite union of conjunctive queries (UCQ)) that is equivalent to  $\Pi$ . However, assuming that the number of tuples in  $e$  is  $N$ , we can in fact see that the number of tuples in the final output of TC is bounded by  $N^2$ , even though the arity  $2m$  may be greater than 2. This is because, at maximum, we choose (1) one tuple from  $e$  and assign the first  $m$  attributes to  $\bar{x}$ , and (2) one tuple from  $e$  and assign the last  $m$  attributes to  $\bar{y}$ .

In this paper, we generalize this idea by deriving size bounds based on a notion of width analogous to edge-cover numbers in conjunctive queries. For example, following our method, the program  $\Pi$  above would be rewritten to the following program  $\Pi'$ :

$$\begin{aligned} R'_1 : & \text{TC}_{\text{TC}(\bar{x}, \bar{y}) \leftarrow e(\bar{x}, \bar{y})}(\bar{x}, \bar{y}) \leftarrow e(\bar{x}, \bar{y}), \\ R'_2 : & \text{TC}_{\text{TC}(\bar{x}, \bar{y}) \leftarrow e(\bar{x}, \_), e(\_, \bar{y})}(\bar{x}, \bar{y}) \leftarrow \text{TC}_{\text{TC}(\bar{x}, \bar{y}) \leftarrow e(\bar{x}, \bar{y})}(\bar{x}, \bar{z}), \quad e(\bar{z}, \bar{y}), \\ R'_3 : & \text{TC}_{\text{TC}(\bar{x}, \bar{y}) \leftarrow e(\bar{x}, \_), e(\_, \bar{y})}(\bar{x}, \bar{y}) \leftarrow \text{TC}_{\text{TC}(\bar{x}, \bar{y}) \leftarrow e(\bar{x}, \_), e(\_, \bar{y})}(\bar{x}, \bar{z}), e(\bar{z}, \bar{y}). \end{aligned}$$

In  $\Pi'$ , each atom of the IDB predicate TC is adorned, shown as subscripts, by a rule with only EDB atoms in the body. Intuitively, this shows the “provenance” of attributes in the tuples involved. More importantly, the rule in the adornment of the head atom of each rule subsumes the results that will be produced by the rule. Therefore,  $\Pi'$  is equivalent to  $\Pi$  in the sense that the set of tuples that will be in the final output of the IDB relation TC in the original program  $\Pi$  is exactly the same as the union of the two adorned variants of TC in  $\Pi'$ , i.e.,  $\text{TC}_{\text{TC}(\bar{x}, \bar{y}) \leftarrow e(\bar{x}, \bar{y})}$  and  $\text{TC}_{\text{TC}(\bar{x}, \bar{y}) \leftarrow e(\bar{x}, \_), e(\_, \bar{y})}$ . In such a case, we say that the edge-cover width of TC is 2, and we can then easily tell that the size of the IDB relation TC is bounded by  $O(N^2)$ .

Our approach is thus based on rewriting—or, more precisely, adorning—datalog programs with non-recursive queries that serve as upper bounds on the IDB relations defined by the program. As a central result, we show that our program rewriting technique can be used to find the minimal integral edge-cover width (which immediately yields an asymptotic bound on output sizes) across all EDB-bounded datalog programs equivalent to a given input program. In other words, determining the worst-case sizes of the IDB relations of a datalog program, or any datalog program equivalent to it, is decidable. This is somewhat surprising, as the equivalence of recursive datalog programs is undecidable [7].

## 2. EDB-Adorned and EDB-Bounded Datalog Programs

We now introduce our framework to provide bounds for datalog programs via adornments added to IDB atoms. These adornments themselves are datalog rules whose body consists of EDB atoms only, and they *subsume* the adorned IDB predicate. Formally, we get the following definitions.

**Definition 2.1 (EDB-adorned datalog).** An EDB-adorned atom is an atom of the form  $q_\rho(\bar{t})$ , such that  $q$  is an IDB predicate of the given schema of arity  $\alpha$ ,  $\bar{t} = (t_1, \dots, t_\alpha)$  is a tuple of the same arity, such that each  $t_i$  is either a variable or a constant, and the adornment  $\rho$  is a safe datalog rule  $A \leftarrow B_1, \dots, B_\mu$  satisfying the following properties: (1)  $A$  is an atom with predicate symbol  $q$ , (2) every body atom  $B_i$  has an EDB predicate and each argument in  $B_i$  is either a variable or the underscore “\_”, and (3)  $q(\bar{t})$  is an instance of  $A$ —that is, there exists a substitution  $\sigma$  such that  $\sigma(A) = q(\bar{t})$ .

An EDB-adorned datalog rule  $R$  is a datalog rule in which the head and every body atom is either an EDB atom or an EDB-adorned IDB atom. A datalog program with only EDB-adorned rules is called an EDB-adorned datalog program.

The *intended meaning* of the adornment  $\rho$  of an EDB-adorned IDB atom  $q_\rho(\bar{t})$  is to indicate how instances of the atom  $q_\rho(\bar{t})$  could be directly derived from EDB atoms via  $\rho$ . The rule  $\rho$  thus corresponds to iterated inlining, and the EDB atoms in the rule body of an adornment constitute restrictions on the tuples that can possibly be derived by the adorned IDB predicate.

**Definition 2.2 (EDB-bounded datalog).** Given an EDB-adorned datalog program  $\Pi$ , let  $R$  be a rule in  $\Pi$  with head predicate  $q_\rho$ . We say that rule  $R$  is bounded by the adornment  $\rho$ , if for every EDB  $D$ , the

following property holds: If an atom  $q_\rho(\bar{t})$  is derived by rule  $R$  when evaluating  $\Pi$  over  $D$ , then  $q(\bar{t})$  is also derived when evaluating the single-rule program consisting of the rule  $\rho$  over  $D$ .

We call  $\Pi$  an EDB-bounded program if every rule of  $\Pi$  is bounded by the adornment of its head predicate.

We now present Algorithm 1, which takes a datalog program  $\Pi$  as input and constructs an *equivalent* (to be defined below) EDB-bounded datalog program  $\Pi'$ . Intuitively, our algorithm iteratively constructs a new adorned rule by taking a rule  $R$  from  $\Pi$  and produces an adorned rule  $R'$  by simultaneously inlining already derived adorned rules into  $R$ . The adornment of the head atom of  $R'$  is obtained by collecting the rule bodies of the adornments of the inlined rules plus the EDB atoms of  $R$ , followed by some simplification of the rule body: more specifically, we first replace all variables occurring in the body but not in the head of the adornment by “\_” (i.e., a fresh existentially quantified variable) and then we remove *redundant* body atoms. A body atom  $A$  is redundant if there exists another body atom  $B$ , s.t.  $A$  and  $B$  have the same predicate and, on every position, they either have the same term or  $A$  has “\_”. This process is continued until a fixpoint is reached.

---

**Algorithm 1:** Construction of an EDB-bounded datalog program from a given program

---

```

input      : A datalog program  $\Pi$ 
output     : An EDB-bounded datalog program  $\Pi'$ 
1 begin program
2 Initialize  $\Pi'$  as an empty datalog program ;
3 repeat
4   foreach rule  $R$  in  $\Pi$  do
5     W.l.o.g. assume that  $R$  has the form  $q(\bar{s}) \leftarrow (\bigwedge_{i=1}^{\lambda} p_i(\bar{s}_i)) \wedge \mathcal{E}$ , where  $p_1(\bar{s}_1), \dots, p_\lambda(\bar{s}_\lambda)$ 
       are IDB atoms, and  $\mathcal{E}$  is a conjunction of EDB atoms ;
6     foreach  $(\rho_1, \dots, \rho_\lambda)$ , such that for all  $i = 1, \dots, \lambda$ , there exists a rule  $R_i$  in  $\Pi'$  with head
       atom  $p_{i\rho_i}(\bar{v}_i)$  with adornment  $\rho_i$  do
7       Apply a variable renaming to all head atoms  $p_{i\rho_i}(\bar{v}_i)$  to obtain  $p_{i\rho'_i}(\bar{v}'_i)$  so that  $R$  and
          all head atoms in all  $R_i$  are variable-disjoint ;
8       Let  $\rho'_i$  be of the form  $p_i(\bar{v}'_i) \leftarrow \mathcal{B}_i$  ;
9        $\sigma \leftarrow \text{some mgu}\{(\bar{s}_i, \bar{v}'_i) \mid 1 \leq i \leq \lambda\}$  ;
10      if no  $\sigma$  exists then
11        | Continue ;
12       $\rho := \text{the rule } q(\sigma(\bar{s})) \leftarrow (\bigwedge_{i=1}^{\lambda} \sigma(\mathcal{B}_i)) \wedge \sigma(\mathcal{E})$  ;
13      In the body of  $\rho$ , replace non-head variables by _ and delete redundant atoms;
14       $R' := \text{the rule } q_\rho(\sigma(\bar{s})) \leftarrow (\bigwedge_{i=1}^{\lambda} p_{i\rho_i}(\sigma(\bar{s}_i))) \wedge \sigma(\mathcal{E})$  ;
15      if  $\Pi'$  does not contain an identical rule (up to variable renaming) then
16        | Add  $R'$  to  $\Pi'$ 
17 until fixpoint of  $\Pi'$  is reached;
18 return  $\Pi'$ 
19 end program.

```

---

We illustrate Algorithm 1 by revisiting the transitive closure program  $\Pi$  from Example 1.1.

**Example 2.3.** Consider the modified transitive closure program  $\Pi$  and the EDB-adorned rules  $R'_1$ ,  $R'_2$ , and  $R'_3$  in Example 1.1. We now verify that these are precisely the rules generated by Algorithm 1:

Rule  $R'_1$  is obtained from  $R_1$  via the special case  $\lambda = 0$  (i.e.,  $R_1$  contains no IDB atoms in the body) in Algorithm 1. Rule  $R'_2$  is obtained by choosing  $R_2$  from program  $\Pi$  and unifying the IDB atom  $\text{TC}(\bar{x}, \bar{z})$  with the head  $\text{TC}_{\text{TC}(\bar{x}, \bar{y}) \leftarrow e(\bar{x}, \bar{y})}(\bar{x}, \bar{y})$  of the adorned rule  $R'_1$ . Due to the simplicity of this example, we skip the variable renaming and mgu computation. Collecting all EDB atoms from the adornment of  $R'_1$

plus the EDB  $e(\bar{x}, \bar{y})$  from the rule body of  $R_2$  followed by the replacement of non-head variables by “\_” yields the adornment  $\text{TC}(\bar{x}, \bar{y}) \leftarrow e(\bar{x}, \_), e(\_, \bar{y})$  of the head predicate of  $R'_2$ .

In the outer foreach-loop of Algorithm 1, we now choose again rule  $R_2$  from  $\Pi$  and unify the IDB atom  $\text{TC}(\bar{x}, \bar{z})$  in its body with the head of rule  $R'_2$ . We thus get the rule  $R'_3$  shown in Example 1.1. Actually, in the first place, we get the adornment  $\text{TC}(\bar{x}, \bar{y}) \leftarrow e(\bar{x}, \_), e(\_, \bar{y}), e(\bar{z}, \bar{y})$  for the head predicate of rule  $R'_3$ . However, replacing the non-head variable  $\bar{z}$  by “\_” and then eliminating one of the redundant atoms  $e(\_, \bar{y})$  yields the rule  $R'_3$  shown in Example 1.1.

In the next iteration of the the outer foreach-loop of Algorithm 1, the only possible choice of a rule from  $\Pi$  is again rule  $R_2$ . But we now unify the IDB atom  $\text{TC}(\bar{x}, \bar{z})$  in its body with the head of rule  $R'_3$ . It is easy to verify that Algorithm 1 then produces a copy of rule  $R'_3$ . Hence, Algorithm 1 has reached the fixpoint, and  $\Pi' = \{R'_1, R'_2, R'_3\}$  is the desired EDB-adorned program. Clearly,  $\Pi$  and  $\Pi'$  are equivalent. Moreover, the adornments in  $\Pi'$  yield the upper bound  $O(N^2)$  on the size of the IDB-relation  $\text{TC}$ .

### 3. Properties of the Programs Constructed by Algorithm 1

Let  $\Pi$  be a datalog program and let  $\Pi'$  be an EDB-bounded datalog program. We say that  $\Pi$  and  $\Pi'$  are equivalent, if for every EDB  $D$  and for every IDB predicate  $q$  in  $\Pi$ , the following condition holds:

$$\llbracket q(\Pi, D) \rrbracket = \bigcup_{\rho} \llbracket q_{\rho}(\Pi', D) \rrbracket$$

That is, the set of tuples in the IDB relation  $q$  that can be derived by program  $\Pi$  from the database  $D$  coincides with the union over all sets of tuples in any adorned IDB relation  $q_{\rho}$  that can be derived by program  $\Pi'$  from the database  $D$ . We can show (for details, see [1]) the following:

**Theorem 3.1.** *Given an arbitrary datalog program  $\Pi$ , let  $\Pi'$  be the EDB-adorned program obtained from  $\Pi$  according to Algorithm 1. Then  $\Pi'$  is an EDB-bounded program equivalent to  $\Pi$ .*

Recall that the (integral) edge-cover width of a set of variables (of a CQ) is the minimum number of atoms (from the CQ) it takes to cover all variables. We thus define the edge-cover width  $\text{ew}(q_{\rho})$  of an adorned predicate  $q_{\rho}$  as the edge-cover width of the head variables in  $\rho$ , when considering the body of  $\rho$  as the CQ. Moreover, for an EDB-bounded program  $\Pi'$ , we define the edge-cover width  $\text{ew}(\Pi')$  of  $\Pi'$  as the maximum  $\text{ew}(q_{\rho})$  over all predicates in  $\Pi'$ . By the definition of EDB-bounded datalog rules, we get the following upper bound on the sizes of IDB relations:

**Theorem 3.2.** *Let  $\Pi$  be an EDB-bounded datalog program, and suppose that we evaluate  $\Pi$  over a database instance  $D$  of size  $|D| = N$ . Let  $k = \text{ew}(q_{\rho})$  denote the edge-cover width of  $q_{\rho}$ . Then every IDB relation  $q_{\rho}$  has size  $|q_{\rho}| \leq N^k$ .*

Moreover, we can show that the edge cover width of the EDB-adorned program  $\Pi'$  obtained from a datalog program  $\Pi$  by Algorithm 1 is in fact the minimum attainable over all EDB-adorned programs  $\Pi'$  equivalent to  $\Pi$ . More specifically, we have the following result:

**Theorem 3.3.** *Let  $\Pi$  be a datalog program and let  $\Pi'$  be the EDB-adorned program produced by Algorithm 1. For a predicate  $q$  in  $\Pi$ , let the integral edge-cover width  $\text{ew}(q)$  of  $q$  be defined as the maximum  $\text{ew}(q_{\rho})$  over all adorned predicates  $q_{\rho}$  in  $\Pi'$ . Then, for every IDB predicate  $q$  in  $\Pi$ ,  $\text{ew}(q)$  for the program  $\Pi'$  is minimal over all EDB-bounded programs equivalent to  $\Pi$ .*

We note that the number of possible adornments of a predicate  $q$  in the EDB-bounded program computed by Algorithm 1 is independent of any concrete database. In other words, this number is only a function of  $\Pi$ . Together with Theorem 3.2, we thus immediately get:

**Corollary 3.4.** *For an IDB relation  $q$  in a datalog program  $\Pi$ , let  $\Pi'$  be the equivalent EDB-bounded datalog program constructed by Algorithm 1. Then,  $|q| \leq f(\Pi) \cdot N^{\text{ew}(q)}$ , where  $f(\Pi)$  is the number of IDB relations of the form  $q_{\rho}$  in  $\Pi'$  (i.e., the number of different adornments  $\rho$  that  $q$  takes in  $\Pi'$ ).*

This raises the natural question as to how we can determine this number of adornments each IDB relation  $q$  in  $\Pi$  can take? Actually, we have shown in [1] that counting the number of adorned predicates of the form  $q_p$  in the EDB-bounded datalog program  $\Pi'$  generated by Algorithm 1 is #P-hard. Hence, we cannot expect an exact closed-form formula for this number without explicitly generating  $\Pi'$ . Nevertheless, in [1], we have derived the following upper bound on  $f(\Pi)$  in Corollary 3.4:

$$f(\Pi) \leq (\text{ar}(q) + |\Pi|)^{\text{ar}(q)} \cdot 2^{\#\text{EDBs} \cdot ((\text{ar}(q)+1)^{\text{ear}} - 1)},$$

where  $\#\text{EDBs}$  denotes the number of EDB predicates and  $\text{ear}$  denotes the maximum arity of the EDB predicates. Clearly, this upper bound is, in general, highly exponential. However, as we have seen in Example 1.1, this number may, in practice, be much smaller.

## 4. Conclusion

In this paper, we have presented the framework of EDB-adorned and EDB-bounded datalog, which has allowed us to obtain a fixed-parameter tractable bound (considering the program  $\Pi$  as parameter) on the size of recursively defined IDB relations. Actually, in [1], we studied a refinement of hypertree decompositions [8] to *free-connex* decompositions. This allowed us to derive also an upper bound on the time required for evaluating a datalog program over a given database. We also study the generalization of our results from integral edge-covers and hypertree decompositions to fractional edge-covers and fractional hypertree decompositions.

## Acknowledgments

Qichen Wang is supported by the Ministry of Education, Singapore, through the Academic Research Fund Tier 1 grant (RS32/25), and by the Nanyang Technological University Startup Grant. Part of this work was completed while Qichen Wang was at EPFL. Reinhard Pichler is supported by the Vienna Science and Technology Fund (WWTF) [10.47379/ICT2201, 10.47379/VRG18013, 10.47379/NXT22018].

## Declaration on Generative AI

The authors have not employed any Generative AI tools.

## References

- [1] C. Fattebert, Z. Jiang, C. Koch, R. Pichler, Q. Wang, Size bound-adorned datalog, Proc. ACM Manag. Data 4 (2026) 97:1–97:27. Full version at <https://doi.org/10.48550/arXiv.2603.15425>.
- [2] V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, T. Neumann, How good are query optimizers, really?, Proc. VLDB Endow. 9 (2015) 204–215.
- [3] W. Cai, M. Balazinska, D. Suciu, Pessimistic Cardinality Estimation: Tighter Upper Bounds for Intermediate Join Cardinalities, in: Proc. SIGMOD 2019, ACM, 2019, pp. 18–35.
- [4] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, T. G. Price, Access path selection in a relational database management system, in: Proc. SIGMOD 1979, ACM, 1979, pp. 23–34.
- [5] S. Seshadri, J. F. Naughton, On the Expected Size of Recursive Datalog Queries, J. Comput. Syst. Sci. 51 (1995) 137–148.
- [6] H. Jordan, B. Scholz, P. Subotić, Soufflé: On Synthesis of Program Analyzers, in: Proc. CAV 2016, Springer, LNCS 9780, 2016, pp. 422–430.
- [7] O. Shmueli, Equivalence of DATALOG queries is undecidable, J. Log. Program. 15 (1993) 231–241.
- [8] G. Gottlob, N. Leone, F. Scarcello, Hypertree Decompositions and Tractable Queries, J. Comput. Syst. Sci. 64 (2002) 579–627.