Range Consistent Query Answering via Rewriting

Jef Wijsen

University of Mons, Belgium

Abstract

We explain that cautious and brave semantics in Consistent Query Answering naturally extend from Boolean queries to numerical queries, where they are known as range semantics. We then apply this extension to numerical queries that are conjunctive queries with aggregation, under primary key constraints. In particular, we highlight results from [1, 2] on computing range semantics via rewriting in aggregate logic.

Keywords

consistent query answering, primary keys, conjunctive queries, aggregation, aggregate logic

1. Motivation

The following relation gives the city and revenue of departments. Ideally, it should satisfy the constraint that *Dep* is the primary key. However, the given relation violates this constraint.

R	*Dep*	*City*	*Revenue*
	Toys	Rome	100
	Toys	Paris	100
	Shoes	Paris	80
	Shoes	Paris	90

The following query asks for the total revenue generated by departments in Paris:

$$\mathbf{sum}(r) \leftarrow R(x, \text{Paris}, r)$$

Since the relation violates the primary key constraint on *Dep*, instead of evaluating the query on it, we consider all *repairs* (i.e., maximal consistent subinstances) and return the minimum and maximum query answers over all repairs. The maximum value 190 is obtained by selecting the tuples (Toys, Paris, 100) and (Shoes, Paris, 90). The minimum value 80 is obtained by selecting the tuples (Toys, Rome, 100) and (Shoes, Paris, 80). The maximal value is computed by the following rules under the standard semantics explained later:

$$\begin{aligned} S(x, \mathbf{max}(r)) &\leftarrow R(x, \text{Paris}, r) \\ T(\mathbf{sum}(m)) &\leftarrow S(x, m) \end{aligned}$$

The first rule computes $S(\text{Toys}, 100)$ and $S(\text{Shoes}, 90)$; the second rule then yields $T(190)$. The minimal value is computed by the following rules:

$$\begin{aligned} N(x) &\leftarrow R(x, y, r), y \neq \text{Paris} \\ S(x, \mathbf{min}(r)) &\leftarrow R(x, \text{Paris}, r), \neg N(x) \\ T(\mathbf{sum}(m)) &\leftarrow S(x, m) \end{aligned}$$

The first rule identifies departments that appear in a non-Paris city. The second rule uses negation to restrict attention to departments that do not appear in any non-Paris tuple.

LINDA'26: Workshop on Logical Approaches to Handling Inconsistent Data

0000-0001-8216-273X (J. Wijsen)

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

In general, given a numerical aggregate query, we study the problem of computing its minimum and maximum answers over all repairs of an inconsistent database instance. Ideally, these values should be computable by a single query over the original instance, without explicitly enumerating repairs. We review conditions under which this is possible, as well as the differences between the computation of minimum and maximum values.

In Section 2, we briefly present the general framework of Consistent Query Answering and its natural extension from Boolean to numerical queries. In Section 3, we apply the framework to conjunctive queries with aggregation and primary keys. In Section 4, we highlight recent results.

2. Consistent Query Answering in a Nutshell

This discussion is with respect to a fixed relational *database schema*, which consists of a finite set of relation names, each with an associated non-zero *arity*. We denote the arity of a relation name R by $\mathsf{ar}(R)$. A *database instance* is a finite set of *facts* of the form $R(c_1, \ldots, c_{\mathsf{ar}(R)})$, where R is a relation name in the underlying database schema and each c_i is a *constant*. A database schema is commonly equipped with a set of *integrity constraints*. A database instance is called *consistent* if it satisfies all integrity constraints, and *inconsistent* otherwise.

In the framework of *Consistent Query Answering* (CQA), a set Σ of integrity constraints gives rise to a mapping $\mathsf{repairs}_\Sigma$ that takes a (possibly inconsistent) database instance and returns a (possibly infinite) nonempty set of consistent database instances. Each element of $\mathsf{repairs}_\Sigma(\mathbf{db})$ is called a *repair* of $\mathbf{db}$ with respect to Σ. If $\mathbf{db}$ is a consistent database instance, then $\mathsf{repairs}_\Sigma(\mathbf{db}) = \{\mathbf{db}\}$. Typically, each element in $\mathsf{repairs}_\Sigma(\mathbf{db})$ is a consistent database instance that is maximally similar to $\mathbf{db}$ according to some similarity measure. In the seminal paper [3] that introduced CQA, each repair must minimize the symmetric difference with $\mathbf{db}$.

A *Boolean query* is a mapping that takes a database instance and returns an element of $\{0, 1\}$, ordered as $0 < 1$. We write $\mathsf{eval}_q(\mathbf{db})$ for the value returned by a query q on a database instance $\mathbf{db}$. In the CQA framework, we are typically interested in the aggregation of the query results over all repairs:

$$agg\left(\{\!\{\mathsf{eval}_q(\mathbf{r}) \mid \mathbf{r} \in \mathsf{repairs}_\Sigma(\mathbf{db})\}\!\}\right), \tag{1}$$

where agg is an aggregate operator. Two common choices for agg are as follows:

(a) By letting $agg = \mathbf{min}$, the value returned is 1 if every repair satisfies q, and 0 otherwise. This corresponds to *cautious* (or *skeptical*) semantics in the literature [4].

(b) By letting $agg = \mathbf{max}$, the value returned is 1 if some repair satisfies q, and 0 otherwise. This corresponds to *brave* (or *credulous*) semantics.

A *numerical query* is a mapping that takes a database instance and returns an element of a totally ordered numerical domain. Since the argument of agg in (1) can be infinite, it is common to replace **min** and **max** with, respectively, the greatest lower bound (**glb**) and least upper bound (**lub**):

$$\mathsf{glb\_eval}_{\Sigma,q}(\mathbf{db}) := \mathbf{glb}\left(\{\!\{\mathsf{eval}_q(\mathbf{r}) \mid \mathbf{r} \in \mathsf{repairs}_\Sigma(\mathbf{db})\}\!\}\right) \tag{2}$$

$$\mathsf{lub\_eval}_{\Sigma,q}(\mathbf{db}) := \mathbf{lub}\left(\{\!\{\mathsf{eval}_q(\mathbf{r}) \mid \mathbf{r} \in \mathsf{repairs}_\Sigma(\mathbf{db})\}\!\}\right) \tag{3}$$

The cautious and brave semantics discussed above are derived from Equations (2) and (3), respectively, by interpreting the Boolean domain $\{0, 1\}$ numerically. The study of numerical queries in the CQA framework was first introduced in [5, 6].

The above definitions readily extend to queries with free variables. Let $q(x_1, x_2, \ldots, x_n)$ be a query with free variables $x_1, \ldots, x_n$. For every tuple of constants $(c_1, c_2, \ldots, c_n)$, the query obtained by substituting each free occurrence of x_i with c_i is a query without free variables, whose semantics is defined as above. In many cases, the computation of these semantics is independent of the actual constants chosen for the x_i. In such cases, free variables can be treated as distinct constants, without requiring a separate treatment.

3. CQA for Primary Keys and Conjunctive Queries with Aggregation

We will now instantiate the general framework of CQA by restricting the integrity constraints and the numerical queries.

Integrity constraints For the integrity constraints, each relation name is further associated with a *primary key arity*, denoted $\mathsf{ar}_{\mathsf{PK}}(R)$, which is a natural number satisfying $1 \leq \mathsf{ar}_{\mathsf{PK}}(R) \leq \mathsf{ar}(R)$. A database instance is consistent if it does not contain two distinct facts with the same relation name, say $R(a_1, \ldots, a_{\mathsf{ar}(R)})$ and $R(b_1, \ldots, b_{\mathsf{ar}(R)})$, such that for every $i \in \{1, 2, \ldots, \mathsf{ar}_{\mathsf{PK}}(R)\}$, $a_i = b_i$. Informally, this means that the attributes $1, \ldots, \mathsf{ar}_{\mathsf{PK}}(R)$ constitute the primary key of R in standard database terminology. Since we assume a fixed key arity for each relation name, these primary key constraints are part of the database schema.

Numerical aggregation queries Certain positions of a given relation name R may be designated as *numerical*. In a fact $R(c_1, \ldots, c_{\mathsf{ar}(R)})$, the value c_i must belong to the underlying numerical domain whenever position i is designated as numerical. We assume that these numerical constraints are always satisfied. The numerical queries we consider take the form:

$$agg(r) \leftarrow R_1(\vec{x}_1), \ldots, R_n(\vec{x}_n), \tag{4}$$

where

- each $\vec{x}_i$ is a tuple of arity $\mathsf{ar}(R_i)$ consisting of variables and constants;
- *self-join freeness:* $i \neq j$ implies $R_i \neq R_j$;
- agg is an aggregate operator. An aggregate operator over a numerical domain maps each non-empty multiset of values from that domain to a single numerical value; its value on the empty multiset is defined either as a numerical value or a designated non-numerical constant. Aggregate operators include, but are not limited to, the standard operators **sum**, **count**, **max**, **min**, and **avg**. In the general framework, we make no assumptions about their computational complexity.
- r is either a numerical constant or a numerical variable. If r is a numerical variable, then it occurs in at least one $\vec{x}_i$, and only in numerical positions.

The atom on the left-hand side of $\leftarrow$ is called the *head*, and the atoms on the right-hand side form the *body*, often denoted by B. We write $\mathbf{vars}(B)$ for the set of variables occurring in B. We say that a valuation θ over $\mathbf{vars}(B)$ *satisfies* B with respect to a database instance if, for every atom $R_i(\vec{x}_i)$ in B, it holds that $R_i(\theta(\vec{x}_i))$ is a fact of the database instance.

We adopt the following standard semantics: if $\theta_1, \ldots, \theta_\ell$ enumerate all the valuations over $\mathbf{vars}(B)$ that satisfy B, then the query returns $agg(\{\!\{\theta_1(r), \ldots, \theta_\ell(r)\}\!\})$, where ℓ may be 0. Note that the argument of agg is in general a (possibly empty) multiset, because it is possible that $\theta_i(r) = \theta_j(r)$ for $i \neq j$. Moreover, since r is a number or a variable restricted to numerical positions, each $\theta_i(r)$ will be a number. We write AGG[sjfCQ] for the class of queries of the form (4). A query in AGG[sjfCQ] is called an agg-query if agg is its aggregate operator.

A subtle point is that the definitions of $\mathsf{glb\_eval}_{\Sigma,q}$ and $\mathsf{lub\_eval}_{\Sigma,q}$ in Section 2 assume that numerical queries return values in a totally ordered numerical domain. These definitions must be revisited for agg-queries when $agg(\{\!\{\}\!\})$ evaluates to a designated non-numerical constant. In particular, following [1], if $q = agg(r) \leftarrow B$, we may define $\mathsf{glb\_eval}_{\Sigma,q}(\mathbf{db})$ to return $\bot$ if $agg(\{\!\{\}\!\})$ is non-numerical and some repair of $\mathbf{db}$ admits no valuation satisfying B.

To see that queries of the form (4) can capture Boolean queries, consider the aggregate operator **or** such that $\mathbf{or}(\{\!\{\}\!\}) = 0$ and $1 = \mathbf{or}(\{\!\{1\}\!\}) = \mathbf{or}(\{\!\{1, 1\}\!\}) = \ldots$. Define $q_{boole} := \mathbf{or}(1) \leftarrow R_1(\vec{x}_1), \ldots, R_n(\vec{x}_n)$. Then $\mathsf{eval}_{q_{boole}}(\mathbf{db})$ returns 1 if there exists a valuation that satisfies the body of the query, and 0 otherwise.

Implicit integrity constraints If q is of the form (4), we write $\mathsf{glb\_eval}_{\mathsf{PK},q}(\mathbf{db})$ as a convenient notation for $\mathsf{glb\_eval}_{\Sigma,q}(\mathbf{db})$ where the set Σ consists of the primary key constraints associated with each R_i.

4. Rewritability into First-Order Logic with Aggregation

As motivated in Section 1, we consider the following problem: given a query q in AGG[sjfCQ], express $\mathsf{glb\_eval}_{\mathsf{PK},q}$ and $\mathsf{lub\_eval}_{\mathsf{PK},q}$ in a target query language whenever this is possible. Our target language, denoted AGG[FOL], extends first-order logic with aggregate operators as in [7, Definition 8.24]. Starting from a non-recursive datalog program, AGG[FOL] allows replacing rules of the form $S(\vec{x}, r) \leftarrow B$ with $S(\vec{x}, agg\,(r)) \leftarrow B$, where r is a numerical constant or variable. Examples appear in Section 1. We adopt the following standard semantics. Let $\vec{a}$ be a tuple of constants of the same arity as $\vec{x}$. Let $\theta_1, \ldots, \theta_\ell$ enumerate all valuations θ over $\mathbf{vars}(B)$ that satisfy B and such that $\theta(\vec{x}) = \vec{a}$. If $\ell \geq 1$, then the rule derives $S(\vec{a}, s)$, where $s = agg\,(\{\!\{\theta_1(r), \ldots, \theta_\ell(r)\}\!\})$. If $\ell = 0$, no fact is derived. Note that the semantics of a rule of the form $S(agg\,(r)) \leftarrow B$ differs from that of the numerical query $agg\,(r) \leftarrow B$: on database instances that admit no valuation satisfying B, the rule derives no S-fact, whereas the numerical query returns $agg\,(\{\!\{\}\!\})$, which is either a numerical value or a designated non-numerical constant.

We now highlight some significant results about **sum**-queries. We take the standard assumption that $\mathbf{sum}\,(\{\!\{\}\!\}) = 0$. It is worth noting that **count**-queries $\mathbf{count}(r) \leftarrow B$ can be expressed as $\mathbf{sum}\,(1) \leftarrow B$, and are thus also covered by these results. We refer to [1, 2, 8] for extensions of these results beyond **sum**-queries.

Theorem 1 ([1, 2])**.** *The following problems are decidable in quadratic time assuming the numerical domain is* $\mathbb{Q}_{\geq 0}$*:*

- *Given a* **sum***-query* q *in* AGG[sjfCQ]*, decide whether* $\mathsf{glb\_eval}_{\mathsf{PK},q}$ *is expressible in* AGG[FOL].
- *Given a* **sum***-query* q *in* AGG[sjfCQ]*, decide whether* $\mathsf{lub\_eval}_{\mathsf{PK},q}$ *is expressible in* AGG[FOL].

Moreover, if the answer to either problem is positive, the corresponding expression in AGG[FOL] *can be constructed effectively.*

The proof of Theorem 1 relies on a deep result by Hella et al. [9] stating that adding aggregation to first-order logic preserves Hanf-locality of queries.

Informally, the following theorem (Theorem 2) states that the criterion for deciding whether $\mathsf{glb\_eval}_{\mathsf{PK},q}$ is expressible in AGG[FOL] coincides with the one from the Boolean case [10], where it is known as the "acyclicity of the attack graph." Interestingly, the same criterion also arises in [11], where queries are interpreted over relations with tuples annotated by elements of a naturally ordered positive semiring. The condition $r \neq 0$ excludes trivial queries $\mathbf{sum}(0) \leftarrow B$ which always return 0 under our assumption that $\mathbf{sum}\,(\{\!\{\}\!\}) = 0$.

Theorem 2 ([1])**.** *Consider two queries in* AGG[sjfCQ] *with the same body* B*:*

$$q : \mathbf{sum}(r) \leftarrow B$$
$$q_{boole} : \quad \mathbf{or}(1) \leftarrow B$$

If $r \neq 0$ *and the numerical domain is* $\mathbb{Q}_{\geq 0}$*, then* $\mathsf{glb\_eval}_{\mathsf{PK},q}$ *is expressible in* AGG[FOL] *if and only if* $\mathsf{glb\_eval}_{\mathsf{PK},q_{boole}}$ *is expressible in* FOL*.*

The following proposition implies that the criterion for deciding whether $\mathsf{lub\_eval}_{\mathsf{PK},q}$ is expressible in AGG[FOL] does not coincide with that of the Boolean case (and hence, by Theorem 2, does not coincide with that for $\mathsf{glb\_eval}_{\mathsf{PK},q}$).

Proposition 1. *Let*

$$q : \mathbf{sum}(1) \leftarrow R(\underline{x}, y), S_1(\underline{y}, x), S_2(\underline{y}, x)$$
$$q_{boole} : \quad \mathbf{or}(1) \leftarrow R(\underline{x}, y), S_1(\underline{y}, x), S_2(\underline{y}, x)$$

Then $\mathsf{lub\_eval}_{\mathsf{PK},q_{boole}}$ *is expressible in* AGG[FOL]*, whereas* $\mathsf{lub\_eval}_{\mathsf{PK},q}$ *is not.*

References

[1] A. Amezian El Khalfioui, J. Wijsen, Computing range consistent answers to aggregation queries via rewriting, Proc. ACM Manag. Data 2 (2024) 218:1–218:19.

[2] A. Amezian El Khalfioui, J. Wijsen, Computing consistent least upper bounds in aggregate logic, in: ICDT, LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2026, pp. 4:1–4:21.

[3] M. Arenas, L. E. Bertossi, J. Chomicki, Consistent query answers in inconsistent databases, in: PODS, ACM Press, 1999, pp. 68–79.

[4] T. Eiter, G. Gottlob, On the computational cost of disjunctive logic programming: Propositional case, Ann. Math. Artif. Intell. 15 (1995) 289–323.

[5] M. Arenas, L. E. Bertossi, J. Chomicki, Scalar aggregation in FD-inconsistent databases, in: ICDT, Lecture Notes in Computer Science, Springer, 2001, pp. 39–53.

[6] M. Arenas, L. E. Bertossi, J. Chomicki, X. He, V. Raghavan, J. P. Spinrad, Scalar aggregation in inconsistent databases, Theor. Comput. Sci. 296 (2003) 405–434.

[7] L. Libkin, Elements of Finite Model Theory, Texts in Theoretical Computer Science. An EATCS Series, Springer, 2004.

[8] A. Amezian El Khalfioui, Computing Range Consistent Answers to Conjunctive Queries with Aggregation, Ph.D. thesis, UMONS - University of Mons [Faculté des Sciences], Mons, Belgium, 18 September 2025. URL: https://hdl.handle.net/20.500.12907/53232.

[9] L. Hella, L. Libkin, J. Nurmonen, L. Wong, Logics with aggregate operators, J. ACM 48 (2001) 880–907.

[10] P. Koutris, J. Wijsen, Consistent query answering for self-join-free conjunctive queries under primary key constraints, ACM Trans. Database Syst. 42 (2017) 9:1–9:45.

[11] P. G. Kolaitis, N. Pardal, J. Virtema, J. Wijsen, Rewriting consistent answers on annotated data, Proc. ACM Manag. Data 3 (2025) 110:1–110:26.