A Principled Solution to the Disjunction Problem of Diagrammatic Query Representations

WOLFGANG GATTERBAUER, Northeastern University, USA

This work solves the disjunction problem of diagrammatic query representations for first-order logic: given any well-formed Tuple Relational Calculus (TRC) query, produce a diagram that (*i*) accurately encodes the query's semantics, (*ii*) preserves its relational structure (i.e., the table references, join patterns, and attribute assignments), and (*iii*) admits an unambiguous, lossless reconstruction of the original query. No prior diagrammatic representation that we are aware of satisfies all three properties once disjunctions are allowed.

CCS Concepts: • **Information systems** → **Relational database query languages**; • **Theory of computation** → **Logic and databases**.

Additional Key Words and Phrases: Visual Query Representation, Logical Diagrams, Relational Diagrams, Peirce Existential Graphs, Tuple Relational Calculus (TRC), First-Order Logic (FOL)

ACM Reference Format:
Wolfgang Gatterbauer. 2026. A Principled Solution to the Disjunction Problem of Diagrammatic Query Representations. *Proc. ACM Manag. Data* 4, 1 (SIGMOD), Article 3 (February 2026), 28 pages. https://doi.org/10.1145/3786617

☞ This pdf contains internal hyperlinks for easier reading: click any linked term to jump to the section where it is defined.

1 Introduction

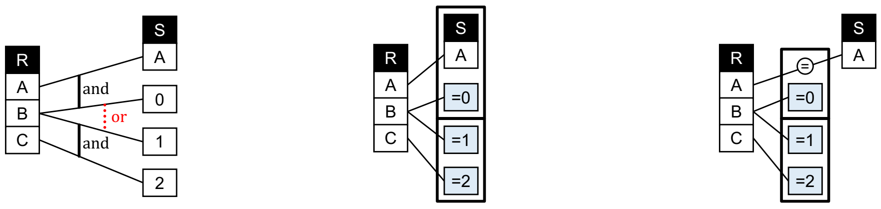
The goal of *query visualization* is to transform a relational query into a visual representation that helps a user quickly understand the intent of a query [35]. Visual representations of relational queries have been investigated since the early days of relational databases. While the history of visual query languages is long and rich [12, 31], the challenge of accurately representing complex logical constructs remains. Already in 1996, Ioannidis [46] lamented that most visual database interfaces were "*ad hoc solutions*" and that "*there are several hard research problems regarding complex querying and visualization that are currently open.*" One such fundamental problem that remains unsolved to this day is the question of how to accurately represent any logical disjunction in a graphical language. Just like conjunction, disjunction is a fundamental logical operator to combine logical statements, but it is far harder to represent graphically. We call this *the disjunction problem* of visual query representations.

The problem has vexed researchers for centuries, even for basic First-Order Logic (FOL).[1] Peirce mentions the problem already in 1896 in his influential work on Venn diagrams: "*It is only disjunctions of conjunctions that cause some inconvenience*" [58, Paragraph 4.365]. Gardner in his 1958 book 'Logic Machines and Diagrams' [28] discusses the challenging disjunction $(A \vee B) \longrightarrow (B \vee C)$ and concludes

[1] FOL is basically the same as Relational Calculus and thus equivalent in logical expressiveness to relationally complete languages.

Author's Contact Information: Wolfgang Gatterbauer, Northeastern University, Boston, MA, USA, w.gatterbauer@northeastern.edu.

ACM 2836-6573/2026/2-ART3
https://doi.org/10.1145/3786617



(a) [31] (b) Interpretation (19) (c) Interpretation (20)

Fig. 1. (a): A prototypical prior approach for representing disjunctions via edges. Edge-based approaches are incomplete as they leave quantifier scopes ambiguous (as in this example) and require symbolic annotations to encode precedence of operators (here, *or* before *and*). (b,c): Representation B has precise semantics and can pattern-represent any well-formed TRC query. For instance, it distinguishes between two interpretations of the ambiguous diagram in (a), an example discussed in detail in Section 8.1.

that "*there seems to be no simple way in which the statement, as it stands, can be diagramed*" [28, Section 4.3]. Shin in her work on the logic of diagrams writes "*any form of representation for disjunctive information–whether a sign is introduced or not–is bound to be symbolic*" [64, Ch. 3.2]. Englebretsen [26] in his review of Shin's 1995 book [63] writes: "*In her discussion of perception she shows that disjunctive information is not representable in any system.*" Thalheim states [70, 71] that "*There is no simple way to represent Boolean formulas*" and gives a challenging example (that is identical to Fig. 10a up to renaming of constants): $R.A = 1 \vee R.B = 2 \vee (R.C = 3 \wedge R.D = 4)$. A recent tutorial by the author on visual query representations [31] lists several open problems and includes Fig. 1a as challenge: $(R.A = S.A \wedge R.B = 0) \vee (R.B = 1 \wedge R.C = 2)$. A recent paper [33] states in its conclusions that "*it is not clear how to achieve an intuitive and principled diagrammatic representation for arbitrary nestings of disjunctions, such as $R.A < S.E \wedge (R.B < S.F \vee R.C < S.G)$ or $(R.A > 0 \wedge R.A < 10) \vee (R.A > 20 \wedge R.A < 30)$*".

Intuitively, the **disjunction problem** is as follows: Given any well-formed Tuple Relational Calculus (TRC) query q, construct a diagram $\varphi(q)$ s.t. the following conditions hold: ❶ *Completeness*: Every well-formed TRC query must yield a corresponding valid diagram $\varphi(q)$. ❷ *Soundness*: Translating $\varphi(q)$ back must yield a logically equivalent query $\varphi^{-1}(\varphi(q)) \equiv q$ (no loss, no ambiguity). ❸ *Pattern preservation*: The diagram $\varphi(q)$ must reference the same set of tables as q. This point is basically a compactness requirement: the size of the diagram should scale proportionally with the query size, i.e., $|\varphi(q)| = O(|q|)$. ❹ *Explicit representation of disjunction*: Disjunctions in q must be *explicitly represented* in $\phi(q)$ since disjunction is a fundamental operator in relational query languages. Additionally, an explicit disjunction symbol also enables syntactic safety to be determined directly from the diagram as is, without requiring any mental transformations of the diagram.

Our contribution. We give a principled solution to the disjunction problem of diagrammatic query representations that *unifies, generalizes, and overcomes the shortcomings* of the 3 main prior graphical approaches for disjunction proposed in the literature. Our solution, called Representation B, is a diagrammatic representation of well-formed Tuple Relational Calculus (TRC) that preserves the relational pattern and the safety of a query.[2] It is heavily inspired by Relational Diagrams [33, 34], however it generalizes Relational Diagrams: it is identical for disjunction-free

[2] In this paper, *safety* exclusively refers to *syntactic safety*, one of several syntactic criteria guaranteeing that the query is domain-independent and thus always returns finitely many answers (see Sections 2.4 and 4.2). We do not mean *semantic safety*, which is domain independence itself and is undecidable (and thus cannot be fully captured by any syntactic condition). The notion of "same relational pattern" is semantic and (slightly simplified) means that the representation uses the same number of relation variables (see Section 2.2).

queries, and it is more general and can be exponentially more concise. It also preserves the safety conditions for TRC, and it is the first to achieve 100% pattern coverage on a recently proposed textbook benchmark.

Our approach in a nutshell. We proceed in two steps: First, we introduce a rigorously defined representation that *replaces join and selection predicates with anchor relations* (a relational encoding of built-in predicates and constants) and rewrites disjunctions using De Morgan's laws. This solves the disjunction problem because it gives a dedicated anchor to each predicate, which can then be placed *in any level of nesting* below the scopes of the table attributes it references. While complete, it is practically unsatisfying due to its visual clutter, the lack of a dedicated disjunction symbol and the fact that it cannot preserve direct safety conditions of TRC. Second, we then substitute the anchor relations with prior visual formalisms (while keeping the formal semantics of anchor relations) and add a box-based visual shortcut for disjunction that brings back the direct safety conditions. Our representation allows disjunctions *at any nesting level*, while prior box-based approaches restrict disjunctions to be at the root.

Outline. Section 2 provides the background and problem definitions. Section 3 classifies prior approaches for representing disjunctions and discusses the challenges. Section 4 develops our notation for Tuple Relational Calculus (TRC), its safety conditions, and the notion of pattern expressiveness based on an Abstract Syntax Tree (AST) representation of TRC. These ASTs are in a 1-to-1 correspondence to our later introduced diagrammatic representations. Section 5 gives our preliminary solution to the disjunction problem with anchor relations, and Section 6 replaces the anchor relations with *De Morgan-fuse boxes*, leading to Representation B. Section 7 discusses our perceptual choices and shows how our fuse boxes unify and generalize prior approaches for disjunction. Section 8 presents our solutions to the challenging queries from the introduction and shows 100% pattern coverage over a database textbook benchmark. The full version [32] includes many more details, proofs, illustrating examples, and an extensive analysis and discussion of prior solutions for disjunctions (including screenshots from original work).

Delineation. While prior work has shown that diagrammatic representations can help users understand relational structures faster and more reliably [33, 49, 55], we do not make any claim that our choice of representation is easier to understand than any other representation. This paper is *not about usability*, but about *feasibility and expressiveness*, similar to other work in our community studying what can be done or not [11, 14, 17, 27, 50]. In that area, we do make a strong claim in this paper. We claim to give the *first pattern-preserving diagrammatic representation* of TRC and thus the *first complete solution to the disjunction problem* (Theorem 7). We also claim that our solution can admit an exponentially more concise representation than prior work (Proposition 10).

2 Formal Background and Problem Statements

We discuss diagrammatic (visual) query representations, define notions of *a logical diagram* and *relational patterns*, define our two problems (the disjunction problem and the direct safety problem), and classify prior approaches for representing disjunctions.

2.1 Diagrammatic vs. textual representations

We use *diagrammatic representation* synonymously with one that is visual, graphical, or non-symbolic (in contrast to *textual* or symbolic), and define logical diagrams as follows:

Definition 1 (Logical Diagram). *A logical diagram is a graphical representation of a logical formula in which the topological relationships between its elements represent logical relationships between the elements of the formula.*

Topological relationships are *spatial relations that remain invariant under continuous deformations*, such as connectivity, containment, and adjacency. Intuitively, in order for a representation to be called diagrammatic, it needs to show joins (i.e. the relationships between tables) as edges between the respective table attributes, and it cannot contain non-atomic logical sentences that require symbolic interpretation of logical connectives, such as "$A=1 \vee A=2$". Our definition captures the essence of many prior definitions of diagrams, for example: "*Diagram: a simplified drawing showing the appearance, structure, or workings of something; a schematic representation*" [57]. "*Diagram: a graphic design that explains rather than represents; especially: a drawing that shows arrangement and relations (as of parts)*" [53]. "*The relationships established between two sets of elements constitute a diagram*" [9, p. 129]. "*Logic diagram: a two-dimensional geometric figure with spatial relations that are isomorphic with the structure of a logical statement*" [28, p. 28]. "*A diagram is an arrangement of marks on a virtual page (...) that represents a set of ideas and their relations*" [75].

Notice that while the relationships between elements are captured diagrammatically, the elements themselves are represented as text. This is because relation names and attribute names do not constitute relational information themselves. For example, the string "Sailor" is still used to represent the name of a relation called "Sailor" (instead of an icon with domain-specific interpretation) and similarly with an attribute named "name". However, the fact that "name" is one of the attributes of a relation named "Sailor" constitutes a relationship (see [31] for an illustration). This *separation of information carried by individual elements via text from relational information that can be read off diagrammatically* is a key motivation of diagrammatic representations and visualizations in general. See for example Scott McCloud's beautiful work on understanding comics [52] that shows that using text to convey part of the information frees up the image to focus on other content, and vice versa. In the case of diagrams, it is the topology that focuses on the relations between elements rather than the individual elements themselves. See also Hearst's recent discussion [43] of the complex interactions when combining text with visualizations and many references therein.

2.1.1 Constitutive vs. enabling features. De Toffoli [72] distinguishes between "constitutive" and "enabling" features of a notational system. *Constitutive features* have precise mathematical meaning and are essential to interpret the notation correctly (e.g. topological relationships). *Enabling features* facilitate interpretation but are not essential (e.g. colors or relative sizes). We find this distinction helpful and use it when discussing non-essential features of our representation (especially in Section 7.1). Similar distinctions were made many times throughout history, e.g. by Manders [51] who contrasts "co-exact" attributes of a diagram (in essence, topological attributes) from "exact" attributes (geometric attributes that are unstable under perturbations, like size and shape), and by Green and Petre [40] who contrast "formal semantics" from "secondary notation" (such as color and grouping of related statements) that "*have no place in the formal semantics*" yet "*could make a substantial difference in the readability*."[3]

2.2 Relational patterns and disjunctions

Our goal is to develop an unambiguous diagrammatic representation of logical formulas that preserves "their structure", even in the presence of arbitrary disjunctions. To formalize this goal, we use the notion of a query's relational pattern [34], which gives precise meaning to a query's relational structure. We define a *table reference* as any quantified reference to a base table within a query expression q. For example, the SQL query "SELECT ... FROM R as R1, R as R2 WHERE ..." has

[3] Although the term "constitutive" may be difficult to grasp at first, and although we considered alternative pairs such as (formal/secondary), (core/auxiliary), or (defining/supporting), we decided in the end to follow De Toffoli's terminology rather than introduce new terms, in order to avoid further proliferation of concepts [81]

two table references "R" to the same base table R.[4] The *(table) signature* $\mathcal{S}$ of a query q is the ordered sequence of all its table references (in the order they appear syntactically within the query).[5] To construct the *dissociated query* q', we treat each table reference in $\mathcal{S}$ as referring to a distinct base table (i.e., a fresh input relation), resulting in a new dissociated signature $\mathcal{S}'$. For the SQL example above, the dissociated query becomes: "SELECT ... FROM R1 as R1, R2 as R2 WHERE..." We then define the relational pattern of q as the function computed by q' from its dissociated signature $\mathcal{S}'$ (a tuple of table references) to an output relation.

Definition 2 (Relational pattern [34]). *Given a query q with table signature $\mathcal{S}$. The* relational pattern *of q is the function defined by its dissociated query $q'(\mathcal{S}')$.*

Intuitively, the dissociated query defines a logical function from a tuple of relation values (one per table reference in q) to an output relation. The ordering of these inputs matters, just as the signature of a function determines the order of its arguments. For our SQL example above, the dissociated query describes a function mapping any database instance with two tables R1 and R2 to an output table. Notice that by abstracting away from language-specific syntax and focusing instead on table references (relations being the one language element common to all relational query languages), the dissociated query *provides a semantic, language-independent characterization of a query's internal structure*, hereafter referred to as its relational pattern. This abstraction makes it possible to compare the pattern expressiveness of different languages. Two queries q_1 and q_2 are said to have the same relational pattern (they are *pattern-isomorphic*) if their dissociated queries are logically equivalent up to renaming and reordering of schema elements and constants (see [34, Def. 10]). If q_1 and q_2 are logically equivalent and have the same relational pattern, we say that q_2 is a *pattern-preserving* representation of q_1 (and vice versa), or that q_2 *pattern-represents* q_1.

Prior work on relational patterns [33] proves that there exist syntactically safe queries in relational calculus (RC) that have logically equivalent queries in relational algebra (RA), yet have no pattern-preserving equivalent in RA. Concretely, the paper presents a safe TRC query with 3 table references and shows that every logically equivalent RA query must have at least 4 table references, thereby having a different relational pattern (the 3 table references in TRC have no bijection to the 4 table references in RA). This implies that neither RA nor its associated evaluation trees can fully capture the set of relational patterns expressible in RC. Note that this result (that RC supports more relational patterns than RA) even applies for syntactically safe queries, and is therefore distinct from the classical observation that unsafe RC queries are not expressible in RA at all [1].

2.3 Problem 1: The disjunction problem

These notions allow us to reformulate conditions ❶-❸ from our intuitive problem formulation stated in Section 1 as follows:

[4] Notice that the following 4 terms denote distinct concepts: *relation variable*, *relation value*, *range variable*, and *table reference*. Date and Darwen [20, 21] introduced the schema-level term *relation variable* (or relvar) to denote the identifier of a base table, distinguishing it from the *relation value*, which represents the current state of the relvar (i.e., its set of tuples). In contrast, *range variables* are query-specific. For example, in the SQL fragment "FROM R as R1, R as R2", the aliases R1 and R2 are range variables that both iterate over the same relation R. Similarly, in the TRC expression "$\exists r_1 \in R, \exists r_2 \in R$", the tuple variables r_1 and r_2 range over the same relation R. In both examples, the two range variables refer to the same base relation R. However, because they are bound separately within the same query, they represent distinct *table references* (a distinction that becomes essential when reasoning about relational patterns).

[5] Replacing the term "table signature" with "relational signature" seems natural. However, for consistency, this would also require renaming "table reference" to "relational reference" and "base table" to "base relation". However, "base table" is an established term that we did not want to change.

Definition 3 (Disjunction problem). *The disjunction problem of diagrammatic query representations is to find a pattern-isomorphic diagrammatic representation for any valid First-Order Logic (FOL) formula, with precise and unambiguous bidirectional translations.*

2.4 Problem 2: Preserving direct safety

Our solution to the first problem has no dedicated symbol for disjunctions and expresses them indirectly with De Morgan's law. Although not strictly necessary, there is a reason why we have the disjunction operator: replacing it with double-negations in logic and language can complicate understanding. Citing from [8] on disjunctions: "*...the fewer connectives we have, the harder it is to understand our sentences.*" For diagrams too, having an *explicit symbol* for disjunction could avoid otherwise nested negations as in Fig. 5a.

Another case for having an explicit disjunction symbol is that safety restrictions are defined via syntactic criteria. Ullman's safety criteria [76, Section 3.8] are *not invariant under equivalence* and are applied directly to the formula as written. They do not require any transformation and are thus also referred to as *syntactic safety* [37]. As minimum example, the query $\{q(A) \mid \neg(\neg(\exists r \in R[q.A = r.A]))\}$ is not Ullman-safe, but its logically equivalent formula $\{q(A) \mid \exists r \in R[q.A = r.A]\}$ is. As a consequence, rewriting a formula using De Morgan's laws can turn a safe query into an unsafe one. We give here a simplified example by Ullman [76, Example 3.29]:

Example 1 (Union of queries). Consider two unary tables $R(A)$ and $S(A)$ and the TRC query:

$$\{q(A) \mid \exists r \in R[q.A = r.A] \lor \exists s \in S[q.A = s.A]\} \tag{1}$$

This query is Ullman-safe [76]. We can remove the disjunction by using De Morgan. However, the resulting query is now not Ullman-safe due to the outer negation ($\neg$) operator [76]:

$$\{q(A) \mid \neg(\neg(\exists r \in R[q.A = r.A]) \land \neg(\exists s \in S[q.A = s.A]))\} \tag{2}$$

Several alternative notions of safety have been proposed that apply increasingly powerful sets of syntactic rewrite-rules before determining safety (the full version [32] gives an overview). We call "*direct safety*" a syntactic criterion that determines safety directly from the formula as written, without any rewrite (or mental transformation), similar to Ullman's. This requires an explicit visual device for disjunction. Condition ❹ from Section 1 can thus be reformulated as:

Definition 4 (Direct safety). *A diagrammatic query representation has direct safety if it allows deciding a formula's safety directly from the diagram without any prior transformation.*

Notice that syntactic safety is different from semantic safety of queries. Semantic safety of a query (the finiteness of its output on every database) is undecidable and can thus not be inferred using syntactic criteria (see [6, Chapter 8]).

3 Prior work and approaches for disjunctions

3.1 Existing Visual query representations and diagrammatic reasoning systems

Visual query languages for writing queries have been investigated since the early days of databases and a 1997 survey [12] has already over 150 references, with examples such as Query-By-Example (QBE) [82] and Query By Diagram (QBD) [4, 13]. Today, many commercial and open-source database systems have rudimentary graphical SQL editors, such as SQL Server Management Studio (SSMS) [68], Active Query Builder [3], QueryScope from SQLdep [60], MS Access [54], and PostgreSQL's pgAdmin3 [59]. Also, new direct manipulation visual interfaces are being developed, such as DataPlay [2] and SIEUFERD [7]. More recently, visual query representations have been proposed for the inverse functionality of understanding queries, with notable examples VisualSQL [47],

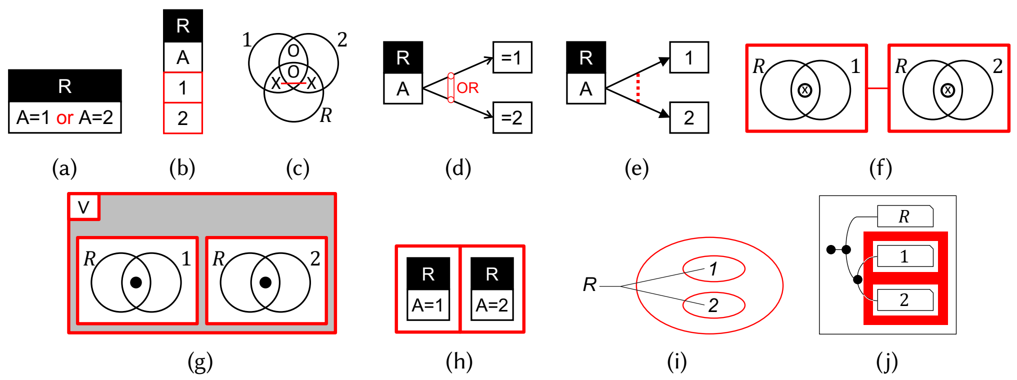


Fig. 2. Section 3.2: This summary shows 5 conceptual approaches for representing disjunctions applied to the deceptively simple problem of representing $R.A\!=\!1 \lor R.A\!=\!2$: text-based (a), form-based (b), edge-based (c-e), box-based (f-h), and De Morgan-based (i,j).

QueryVis [19, 49], and SQLVis [55]. Some predict a future user interaction where users speak to voice assistants, and those then "visualize back" what they understood [35].

Seemingly disconnected from these developments, the diagrammatic reasoning community [22, 39, 44] studies diagrammatic representations that have sound and complete inference rules. Most noteworthy is Shin's influential work [63] that proves that a slight modification of Venn-Peirce diagrams constitutes a sound and complete diagrammatic reasoning system for monadic FOL. Many variants of diagrammatic reasoning systems have since been proposed at the annual Diagrams conference [48]. However, neither of these proposals can represent general polyadic predicates (the maximum are dyadic relations [69, Table 1], many of them are either not sound or not complete [69, Table 2], and neither of them allow pattern-isomorphic representations, even for the fragments of logic they cover (most often, they can't handle arbitrary disjunctions).

Recent VLDB and ICDE tutorials [30, 31] surveyed diagrammatic representations within and outside the database community and listed diagrammatic representations of disjunction as open problem.

3.1.1 Relational Diagrams. Relational Diagrams [33] are a relationally complete and unambiguous representation of safe TRC. They use UML notation for tables and their attributes and represent negation scopes with hierarchically nested dashed rounded rectangles that partition the canvas into *zones* (compartments). Join predicates are shown with directed arrows and optional labels on the edges (an important detail for us later). However, this representation (like all prior diagrammatic representations) cannot accurately represent relational patterns involving disjunctions. To represent disjunctions, Relational Diagrams first *require a transformation that duplicates binding atoms* (i.e. add new table references), which changes the relational pattern (Example 3 illustrates this transformation). The intuitive reason is that a negation like $\neg(R.A = S.A \land \ldots)$ does not apply to either of the tables R nor S, but the equality predicate as a whole. This predicate is represented by a line, which cannot be easily confined to a negation scope.

We were inspired by that work, yet develop a diagrammatic representation called Representation B that can represent *all relational patterns of TRC* (i.e. it is *pattern-complete* for TRC) and is backward compatible with Relational Diagrams (every Relational Diagram has an identical representation as Representation B, but not vice versa). Interestingly, we achieve this generalization by mostly redefining existing visual notations and giving them a stricter semantic interpretation. Our representation does not only preserve the table signature (the relation variables), but also the join and selection predicates, and thus all atoms from a given TRC query.

3.2 Prior incomplete approaches for disjunction

We summarize here 5 conceptual approaches for representing disjunctions that we found throughout the literature. We illustrate with query $\exists r \in R[r.A=1 \vee r.A=2]$ and use a standardized and often simplified notation that focuses on the key visual elements. For the interested reader, full version [32] contains the original figures that led us to this classification.

3.2.1 Text-based disjunctions. Logical formulas can be kept as text (Fig. 2a). This approach is not diagrammatic, and we list it only for completeness. Example uses are the condition boxes by QBE [82] and handling of simple disjunctions by DataPlay [2].

3.2.2 (Vertical) form-based disjunctions. QBE [82] allows filling out two separate rows with alternative information (Fig. 2b). Recent visual query representations such as SQLVis [55] adopt this approach for simple disjunctions, such as our running example. However, this formalism does not allow disjunctions between join predicates and selection predicates (e.g., $\exists r \in R[r.A=1 \vee \exists s \in S[r.A=s.A]]$) or nested disjunctions. Datalog [15] expresses disjunctions with repeated rules and each rule has a new relation variable. Thus, it does not preserve the pattern: $Q :\!- R(1). Q :\!- R(2)$.

3.2.3 Edge-based disjunctions. Around 1896, Charles Sanders Peirce [58] extended Venn diagrams [77] with edges connecting "O" and "X" markers to express disjunctions. An "O" means the partition is empty (false). An "X" means there is at least one member (true). An edge between two markers means that at least one of these statements is true (Fig. 2c). Connecting disjunctive predicates via edges of various forms was suggested repeatedly, e.g. by VQL [56], VisualSQL [47] (Fig. 2d), and QueryViz [19] (Fig. 2e). Edges were mostly proposed for disjunctive filters within the same table and cannot represent more complicated formulas, such as (19) and (20) discussed in Section 8.1.

3.2.4 Box-based disjunctions. Peirce proposed another solution to disjunctions [58]: He put unitary Venn diagrams into rectangular boxes and interpreted adjacent boxes as alternatives, i.e. disjuncts. Shin [63] adds back lines between boxes (Fig. 2f).[6] Spider diagrams [45] remove the lines between boxes and place them in a larger "box template" with explicit $\vee$ labels (Fig. 2g). Relational Diagrams [33] represent a union of queries via adjacent "union cells" (Fig. 2h). All of these prior box-based approaches represent disjunctions as unions of well-formed diagrams. Ours is the first to allow and give a well-defined semantics to disjunctions of logical expressions deeply nested *within diagrams*.

3.2.5 De Morgan-based disjunctions. We use the term for representations that use only symbols for negation and conjunction, and apply negation in a nested way in accordance with the logical identity $A \vee B \equiv \neg(\neg A \wedge \neg B)$. Peirce's beta existential graphs [58] use closed curves to express negation and juxtaposition for conjunctions (Fig. 2i). String diagrams [10, 42] are a variant that represent bound variables by a dot at the end of lines (Fig. 2j). These prior De Morgan-based approaches cannot represent arbitrarily nested logical conditions, such as the ones from the introduction, since predicates (often expressed by lines) lacked stable anchor points for negation. Our anchor relations solve this problem by providing stable references for negation scopes (see Fig. 5a).

[6] We slightly simplified here Shin's proposal. The conclusion is the same, and our appendix gives the full details.

4 Tuple Relational Calculus (TRC)

We give a succinct and necessary background on TRC. The full version [32] discusses different formalisms for TRC, different notions of safety, and includes a proof of the correctness of our safety definition.

4.1 Well-formed Tuple Relational Calculus (TRC)

Our key claim is that our representation has a direct and, hence, pattern-preserving mapping from *any* safe TRC query. To prove that, we are extra careful in defining our notions. Our formalism is heavily inspired by the sections on relational calculus of several textbooks [1, 24, 61, 62, 65, 76], yet it is geared towards an explicit mapping to a visualization. Example 2 gives an end-to-end example.

TRC formulas. Given a relational vocabulary $\mathbf{R} = \{R_1, R_2, \ldots\}$, a *(well-formed) TRC formula* can contain 3 types of *atoms*:

(1) *Binding atoms* of the form $r \in R$, where r is a range variable (i.e., a variable ranging over tuples of relation R).[7]
(2) *Join predicates* of the form $r.A\,\theta\,s.B$, where r and s are range variables, A and B are attributes of r and s, respectively, and $\theta \in \{<, \leq, =, \neq, >, \geq\}$ is a comparison operator. The expressions $r.A$ and $s.B$ are the left and right operands of the comparison.
(3) *Selection predicates* of the form $r.A\,\theta\,c$, where r, A, and θ are as before, and c is a constant. Here, $r.A$ is the attribute operand and c is the constant operand.

A *range variable* r (also called a *tuple variable*) is said to be *free* unless it is bound by a quantifier/binding block of the form $\exists r \in R$ or $\forall r \in R$. TRC formulas are built inductively from atoms using the following 3 families of formation rules:

(1) *Atoms*: An atom is a formula.
(2) *Logical connectives*: If $\varphi_1, \varphi_2, \ldots, \varphi_k$ are formulas, then so are $\neg(\varphi_1)$, $(\varphi_1 \rightarrow \varphi_2)$, $(\varphi_1 \vee \varphi_2 \vee \cdots \vee \varphi_k)$, and $(\varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_k)$.
(3) *Quantifier/binding blocks*: If φ is a formula and $R_1, \ldots, R_k$ are relations from $\mathbf{R}$, then $\exists r_1 \in R_1, \ldots, r_k \in R_k[\varphi]$ and $\forall r_1 \in R_1, \ldots, r_k \in R_k[\varphi]$ are also formulas.[8] This means that all bindings of range variables (e.g., $r \in R$) are introduced by an explicit quantifier, and those range variables are available within the scope defined by the brackets: $\exists r \in R, s \in S[\langle \text{scope of } r \text{ and } s \rangle]$.[9]

We assume the usual operator precedence $(\neg) > (\wedge) > (\vee) > (\rightarrow)$ and can omit parentheses if this causes no ambiguity about the semantics of the formula. Quantifier scopes are always explicit using brackets $[\ldots]$, so precedence does not apply to quantifiers. Notice that our convention is different from the common informal convention in FOL, where a quantifier is often understood to extend as far right as possible unless parentheses intervene. For example, applying our convention to Domain Relational Calculus, we would write $\exists x[\phi] \wedge \psi$ instead of $(\exists x.\phi) \wedge \psi$, and $\exists x[\phi \wedge \psi]$ instead of the ambiguous notation $\exists x.\phi \wedge \psi$. WLOG, no range variable can be bound more than once, and no variable occurs both free and bound in a subformula.

TRC queries. A (well-formed) Boolean query (or sentence) is a TRC formula without free variables. A (well-formed) non-Boolean query is an expression $\{q(\mathbf{H}) \mid \varphi\}$ where q is the only

[7] While we use "binding atom" exclusively for atomic expressions of the form $r \in R$, they are always introduced by quantifiers and together form a quantifier/binding block. The reason is that we treat $r \in R$ not as a guard or Boolean membership atom (sometimes written as $R(r)$ [24]), but rather the entire quantifier/binding block $\exists r \in R$ similar in spirit to a generator $r \leftarrow R$ in a list comprehension framework that *binds* the range variable r to each row in R (see [36] for details).

[8] Notice that in our notation, *two binding atoms can share the same quantifier.* Thus, we write $\exists r \in R, s \in S[\varphi]$ instead of $\exists r \in R, \exists s \in S[\varphi]$. If quantifiers alternate, we write $\exists r \in R\,[\forall s \in S[\varphi]]$, instead of $\exists r \in R, \forall s \in S[\varphi]$. We also allow $\exists r \in R\,[\exists s \in S[\varphi]]$, but prefer to write the logically equivalent maximally scoped $\exists r \in R, s \in S[\varphi]$.

[9] We also allow a body-less quantifier/binding if a range variable is never used: $\exists r_1 \in R_1, \ldots, r_k \in R_k$ and $\forall r_1 \in R_1, \ldots, r_k \in R_k$. For example, $\exists r \in R$ is a valid sentence that is true if R is not empty.

free tuple variable of φ, and the *header* $\mathbf{H} = (A_1, \ldots, A_k)$ is a list of attributes (or components) of q specifying the output schema. The set builder notation defines the answer as the set of tuples $(q.A_1, \ldots, q.A_k)$ that satisfy φ.

Abstract Syntax Tree (AST). The *Abstract Syntax Tree (AST)* of a TRC query is a tree-based representation that encodes a unique logical decomposition into subexpressions. It abstracts away certain syntactic details from the parse tree and gives a unique reversal of the inductively applied formation rules. The *leaves* (inputs) are formed by atoms. Inner nodes belong to 3 families:

(1) The root for non-Boolean queries $\{q(\mathbf{H}) \mid \varphi\}$ is formed by a *query* node. Its two children are *output* for the output relation $q(\mathbf{H})$ and *formula* for φ. The root of Boolean queries is formed by a *formula* node.
(2) $\exists r_1 \in R_1, \ldots, r_k \in R_k[\varphi]$ and $\forall r_1 \in R_1, \ldots, r_k \in R_k[\varphi]$ are represented by $\exists$ and $\forall$ quantifier nodes, respectively. Their two types of children are one or more *binding atoms* and zero or one *formula* φ.
(3) *Logical connectives* are nodes that have either one child ($\neg$), two children ($\rightarrow$), or $k \geq 2$ children ($\wedge, \vee$), and that form the root of a formula.

We require that no $\neg$ is the child of another $\neg$ node (we can always cancel double negations by $\neg\neg\varphi \equiv \varphi$) and that the polyadic connectives ($\wedge, \vee$) are "flattened" [1, Sect. 5.4], i.e. they can have more than 2 children, yet no child of an $\wedge$ is another $\wedge$ (analogously for $\vee$). Similarly, quantifier nodes ($\exists, \forall$) can't have a quantifier node of the same type as a grandchild, i.e. a $\exists$ quantifier node can't have another $\exists$ quantifier node as child of its formula child.

Maximally scoped TRC. We call a TRC formula *maximally scoped* if no $\exists$ quantifier node is the child of an $\wedge$ node. This is WLOG, as existential quantifiers can always be pushed before an $\wedge$ node, as in: $\exists r \in R[\varphi_1] \wedge \exists s \in S[\varphi_2] \equiv \exists r \in R, s \in S[\varphi_1 \wedge \varphi_2]$.

Safety of Boolean queries. While Boolean queries in Domain Relational Calculus can be unsafe, e.g. $\exists x[\neg(R(x))]$, in our definition of well-formed TRC, all relation variables (other than the output) are bound to a base table. It follows that well-formed Boolean TRC queries are always safe, and hence, domain-independent.

Proposition 5 (Boolean TRC safety). *Every well-formed Boolean TRC according to Section 4.1 is domain-independent.*

4.2 Explicit safety of TRC

Recall that (syntactic) safety *syntactically restricts* the well-formed TRC queries s.t. safe queries are guaranteed to be domain-independent (and thus have only finitely many answers), and this subset can express all possible finite queries [73]; *direct safety* (Definition 4) requires that such safety can be recognized directly from the representation without transformation. We will next define a syntactic safety condition called "*explicit safety*" that fulfills the direct safety conditions.

We call the *base partition* of an AST the maximal subtree reachable from the root without crossing a negation ($\neg$), implication ($\rightarrow$), or universal quantifier ($\forall$). We call *base disjunction* any disjunction that appears in the base partition. We say that a non-Boolean TRC query $\{q(\mathbf{H}) \mid \varphi\}$ is *explicitly safe* if it is well-formed and the following 4 conditions hold on φ:

(1) Every attribute A of the header $\mathbf{H}$ is assigned in φ to either (i) an attribute B of an existentially quantified table $\exists r \in R$ via an *equijoin predicate* $q.A = r.B$, or (ii) a constant c via an *equiselection predicate* $q.A = c$. In both cases, we call this equality predicate an *assignment predicate* for $q.A$.
(2) Every assignment predicate is in the base partition of the AST.
(3) Under base disjunctions: If an assignment predicate for $q.A$ occurs under a base disjunction $\vee$ in the AST, then all child subformulas of that $\vee$ node have exactly one free tuple variable, and it is the same variable with the same attributes.

```
QUERY
├─ OUTPUT: q(A,B)
└─ OR
   ├─ AND
   │  ├─ PREDICATE: q.A = 0
   │  └─ OR
   │     ├─ QUANTIFIER ∃
   │     │  ├─ BINDING: r ∈ R
   │     │  └─ PREDICATE: q.B = r.B
   │     └─ PREDICATE: q.B = 1
   └─ QUANTIFIER ∃
      ├─ BINDING: r2 ∈ R
      └─ AND
         ├─ PREDICATE: q.A = r2.A
         ├─ PREDICATE: q.B = r2.B
         └─ QUANTIFIER ∀
            ├─ BINDING: s ∈ S
            └─ QUANTIFIER ∃
               ├─ BINDING: r3 ∈ R
               └─ AND
                  ├─ PREDICATE: r2.A = r3.A
                  └─ PREDICATE: r3.B = s.B
```

Fig. 3. Example 2: AST for TRC query with nested disjunctions.

(4) Outside base disjunctions: If an assignment predicate for $q.A$ is not under any base disjunction in the AST, then it is the unique predicate involving $q.A$.

Conditions (1) and (2) ensure that every output attribute is explicitly defined by an assignment predicate in the base partition. Conditions (3) and (4) regulate how these definitions interact with disjunctions: under a base disjunction, each branch must define the same output schema independently; outside disjunctions, each output attribute has a single assignment predicate. Together, these 4 conditions for explicitly safe TRC queries ensure that, once a single branch is chosen at each base disjunction, every output attribute is assigned to exactly one table column (or constant) in the base partition of the AST. Equivalently: each base disjunction induces a choice of one branch; after fixing all such choices, each $q.A$ has exactly one assignment in the base partition. Figure 3 illustrates that with the purposefully involved Example 2.

Notice that condition (4) of our explicit safety definition enforces a convenient normal form which simplifies both reading and diagrammatic interpretation and is without loss of generality. Any otherwise well-formed TRC query can be transformed into this normal form without changing its relational pattern: whenever an output attribute appears in multiple predicates, all but one that fulfills condition (1) can be replaced by equivalent join predicates between the corresponding range variables. The resulting query is logically equivalent, pattern-preserving, and explicitly safe (see the full version [32] for a more detailed discussion).

Also notice that maximally scoping a TRC expression by pushing existential quantifiers before all $\wedge$ nodes does not affect the base partition of the AST, and thus neither explicit safety.

Example 2. Consider the following safe non-Boolean TRC query:

$$\begin{aligned}\{q(A,B) \mid\; & (q.A=0 \wedge (\exists r \in R[q.B=r.B] \vee q.B=1)) \\ & \vee (\exists r_2 \in R[q.A=r_2.A \wedge q.B=r_2.B \\ & \quad \wedge \forall s \in S[\exists r_3 \in R[r_2.A=r_3.A \wedge r_3.B=s.B]]])\}\end{aligned}$$

Figure 3 shows its AST. Notice how the 4 safety conditions are fulfilled. In particular, both child subformulas "$\exists r \in R[q.B=r.B]$" and "$q.B=1$" of the lower nested disjunction have $q(B)$ as free tuple variable. However, both child subformulas of the earlier disjunction in the tree have $q(A,B)$ as free tuple variable.

5 A preliminary solution with anchor relations

This section develops an extension to Relational Diagrams that solves the disjunction problem and gives the resulting representation the same pattern expressiveness as TRC. The approach relies on anchor relations (i.e. unary and binary relations that represent constants and built-in predicates). The approach is simple in that it does not require any novel visual syntactic devices (it uses even a smaller visual vocabulary than Relational Diagrams). However, it is practically unsatisfying due to its additional visual clutter, and the fact that it *cannot preserve the explicit safety conditions* of TRC. We address these problems in the subsequent section.

Here, we first discuss two important fragments of TRC (Section 5.1), then discuss our idea of anchor relations (Section 5.2), before we prove it to be pattern-isomorphic to TRC (Section 5.3).

5.1 TRC$^{\neg\exists\wedge}$ is an atom-preserving fragment of TRC, but not of safe TRC

We first show that universal quantifiers $\forall$, material implications $\rightarrow$, and disjunctions can be replaced in TRC by using only the symbols for negation $\neg(\ldots)$, existential quantification $\exists$, and conjunction $\wedge$ *without changing the relational pattern.*[10]. While it is standard textbook knowledge that the connectives $\neg, \wedge$ are truth-functionally complete [8, Sections 7.4], we show the slightly more general statement that we can preserve *all atoms* in the AST.

Lemma 6. *Given a TRC formula φ with universal quantification, implication, or disjunction. Then there exists a logically equivalent TRC formula φ' that (i) is pattern-isomorphic to φ, (ii) does not use universal quantifiers, implications, or disjunction, (iii) uses the identical set of atoms, and (iv) can be found in polynomial time in the size of φ.*

We call "Existential-Negation-Conjunctive TRC" (TRC$^{\neg\exists\wedge}$) the fragment of TRC that only uses the connectives $\{\neg, \exists, \wedge\}$.[11] We call "Existential-Negation TRC" (TRC$^{\neg\exists\wedge\vee}$) the variant of that fragment that additionally allows disjunction $\vee$.

5.1.1 Comparison with the non-disjunctive fragment. The *non-disjunctive fragment* of Relational Diagrams includes an extra condition requiring all predicates to be "guarded" (each predicate needs to contain a "local attribute" whose relation is quantified within the scope of the last negation). This condition leads to a reduction in logical expressiveness, which the authors fixed by adding a union operator as a new visual element. It also leads to cases where expressing a query requires a *different table signature*. This is in contrast to TRC$^{\neg\exists\wedge\vee}$ and TRC$^{\neg\exists\wedge}$ which are only syntactic restrictions of non-leaf nodes of the AST.

Example 3. Consider the following TRC query that is a variation on relational division. It returns values from $R.A$ that co-occur in R with either $S.B$ or $S.C$, for all tuples from S with $S.A > 0$:

$$\begin{aligned}\{q(A) \mid \exists r \in R[q.A = r.A \wedge (\forall s \in S[s.A > 0 \;\rightarrow \\ (\exists r_2 \in R[(r_2.B = s.B \;\vee\; r_2.C = s.C) \wedge r_2.A = r.A])])]\}\end{aligned} \tag{3}$$

[10] The lemma follows from a straightforward application of the standard transformations, yet may not be immediately obvious. Take as an example the single connective NOR ($\downarrow$) which is also truth-functionally complete [8, Sections 7.4], yet replacing connectives (NOT, AND, OR) with NOR *would not be pattern-preserving*: $\neg(\varphi) \equiv \varphi \downarrow \varphi$. As example, consider the query $\{q(A) \mid \exists r \in R[q.A = r.A \wedge \neg(\exists s \in S[r.B = s.B])]\}$. Replacing $\neg$ with $\downarrow$ would lead to a different relational pattern $\{q(A) \mid \exists r \in R[q.A = r.A \wedge (\exists s \in S[r.B = s.B] \downarrow \exists s \in S[r.B = s.B])]\}$

[11] We have consulted a long list of standard textbooks on logic [8, 25, 38], online resources, and LLMs, yet have not found a standardized, shorter, non-ambiguous terminology for this rather natural fragment, despite being often implicitly used.

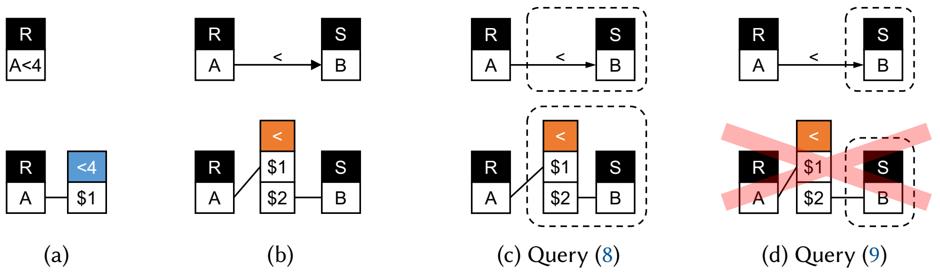


Fig. 4. Sections 5.2.1 and 5.2.2: (a) Unary (blue) and (b) binary (orange) anchor relations are added to the visual vocabulary of Relational Diagrams in order to make the resulting diagrammatic representation system pattern-complete for TRC. Example 4 (c), (d): the placement of operator labels does not matter for Relational Diagrams (upper diagrams). However, it matters when replacing the labels with new relations (lower diagrams).

Replacing $\forall$ and $\rightarrow$ leads to an expression in TRC$^{\neg\exists\wedge\vee}$:

$$\{q(A) \mid \exists r \in R[q.A = r.A \wedge \neg(\exists s \in S[s.A > 0 \wedge \neg(\exists r_2 \in R[(r_2.B = s.B \ \vee \ r_2.C = s.C) \wedge r_2.A = r.A])])]\} \tag{4}$$

Further replacing $\vee$ leads to a query in TRC$^{\neg\exists\wedge}$:

$$\{q(A) \mid \exists r \in R[q.A = r.A \wedge \neg(\exists s \in S[s.A > 0 \wedge \neg(\exists r_2 \in R[\neg(\neg(r_2.B = s.B) \ \wedge \ \neg(r_2.C = s.C)) \wedge r_2.A = r.A])])]\} \tag{5}$$

Notice that the above three expressions are not only pattern-isomorphic (we do not change the signature), but also all the selection predicates (e.g. "$S.A > 0$") and the join predicates (e.g. "$r_2.C = s.C$") are identical. Only the logical connectives ($\neg, \vee, \wedge, \rightarrow$), quantifiers and parentheses have changed. In other words, *all atoms* are identical, and hence all leaves in their ASTs are identical as well.

This is in contrast to the non-disjunctive fragment required by Relational Diagrams. Concretely, [33] suggests to use De Morgan with quantifiers to distribute the quantifier over the disjuncts and transforming into a conjunction: $\neg(\exists r \in R[A \vee B]) = \neg(\exists r \in R[A] \vee \exists r \in R[B]) = \neg\exists r \in R[A] \wedge \neg\exists r \in R[B]$. This leads to a disjunction-free query, yet also a different signature (3 occurrences of table R) and thus a different relational pattern (also shown in Fig. 5b):

$$\{q(A) \mid \exists r \in R[q.A = r.A \wedge \neg(\exists s \in S[s.A > 0 \wedge \neg(\exists r_2 \in R[r_2.B = s.B \wedge r_2.A = r.A]) \wedge \neg(\exists r_3 \in R[r_3.C = s.C \wedge r_3.A = r.A])])]\} \tag{6}$$

5.1.2 *Safety is preserved by TRC$^{\neg\exists\wedge\vee}$, but not by TRC$^{\neg\exists\wedge}$.* Recall that explicit safety is a syntactic criterion, and applying De Morgan can render a safe query unsafe and vice versa (recall Example 1). Thus removing disjunction from the vocabulary makes it impossible to represent all logical queries *while preserving explicit safety*. It is easy to see that the safe query from Example 1 cannot be expressed in TRC$^{\neg\exists\wedge}$ while preserving safety: The output needs to be restricted to the union of $R(A)$ and $S(A)$. In the absence of disjunction $\vee$ that can only be achieved with De Morgan, which renders the base partition empty, and the resulting query unsafe. In contrast, TRC$^{\neg\exists\wedge\vee}$ *preserves explicit safety* since all transformations for removing $\forall$ and $\rightarrow$ from a safe TRC query must happen *outside the base partition* of its AST, and thus no assignment predicate changes during the transformation.

5.2 Anchor relations reduce the visual vocabulary but extend the pattern expressiveness of Relational Diagrams

We next add unary and binary anchor relations to the vocabulary of Relational Diagrams and show that this addition enables the resulting visual representation to represent all patterns of TRC. *Anchor relations* are (possibly infinite) relations encoding built-in predicates (externally defined relations [36]). For example, the anchor relation for $<$ is the (infinite) set of all pairs (x, y) such that $x < y$." These anchor relations can serve as stable anchors for negation scopes and thus permit a direct translation from $\text{TRC}^{\neg\exists\wedge\vee}$ to such extended Relational Diagrams.[12] Furthermore, by expressing predicates as relations, we *do not have to introduce any new visual elements*. However, we lose the ability to encode safety and it becomes visually more complex. We address both issues later.

5.2.1 Constants represented as unary anchor relations. For each constant c and each arithmetic comparison operator $\theta \in \{=, <, \leq, >, \geq, \neq\}$, we allow a new unary relation $\boxed{\theta c}$ descriptively named "θc" that contains the (possibly infinite) subset of the domain that fulfills that condition. Each selection predicate "$R.A\,\theta\,c$" is then represented as equijoins with a different occurrence of that relation.[13] WLOG, we adopt Ullman's notation [76] for the ordered, unnamed perspective and name the column \$1 (for first attribute). Figure 4a shows the representation for the selection predicate "$R.A < 4$". Notice that our color choice of a blue background for unary anchor relations is not "constitutive" but "enabling" (Section 2.1.1).

5.2.2 Join predicates represented as binary anchor relations. For each arithmetic comparison operator $\theta \in \{=, <, \leq, >, \geq, \neq\}$, we allow a new binary relation $\boxed{\theta}$ descriptively named "θ" that contains the subset of the (possibly infinite) cross-product of the domain that fulfills that arithmetic comparison. Each join predicate "$R.A\,\theta\,S.B$" is then represented as two equijoins of R and S with an instance of such a relation. We again adopt Ullman's notation and name the columns \$1 and \$2 (for first and second, respectively). Figure 4b shows the representation for the join predicate "$R.A < S.B$". Notice that our choice of orange background color for binary anchor relations is again only "enabling".

5.2.3 Correct placement of anchor relations. The placement of edge labels representing built-in predicates in Relational Diagrams [33] is not important since the correct interpretation is guaranteed by the "guard" of each predicate (i.e. the innermost nested relations). This freedom of placement disappears for our anchor relations, as we illustrate next.

Example 4. According to [33], the two upper Relational Diagrams in Figs. 4c and 4d have the same meaning. The negation scope only applies to the enclosed table and the position of the label "<" does not alter the semantics. They both assert: "There exists a value in $R.A$ s.t. there is no value in $S.B$ that is bigger", i.e.

$$\exists r \in R[\neg(\exists s \in S[r.A < s.B])] \tag{7}$$

[12] An anchor is a visual element in a diagram that provides a stable reference for logical operations. An anchor makes it possible to unambiguously represent and compose otherwise hard-to-visualize logical constructs like negation and disjunction. We originally called those relations "built-in relations" in reminiscence of built-in predicates like < in SQL [76]. We now prefer the term "anchor relations" as the term also applies to higher-arity, non-built-in predicates such as "R.A+S.B>T.C" and arithmetic predicates as in "SELECT A+B as C FROM R". Interesting recent work by Guagliardo et al. [41] calls these infinite relations "external predicates" and describes a general framework for inferring safety for queries that use them.
[13] When clear from the context, we write the table name instead of a table variable.

After replacing the built-in predicate $r.A < s.B$ with an anchor relation $\boxed{<}$, whether it is placed inside or outside the negation scope (bottom in Figs. 4c and 4d) changes the diagram's meaning:

$$\exists r \in R[\neg(\exists s \in S, j \in \text{“}<\text{”}[r.A = j.\$1 \wedge j.\$2 = s.B])] \tag{8}$$

$$\exists r \in R, j \in \text{“}<\text{”}[\neg(\exists s \in S[r.A = j.\$1 \wedge j.\$2 = s.B])] \tag{9}$$

Query (8) is identical to (7), but query (9) states something far more permissive: "There exists a value in $R.A$ s.t. there exists a bigger value that is not in $S.B$." For example, assume the database is $R(1)$ and $S(2)$. Then variant (8) is false (as expected), whereas variant (9) is true for the assignment: $R.A = \text{“}<\text{”}.\$1 = 1$ and $\text{“}<\text{”}.\$2 = 3$ (since the value 3 does not exist in $S.B$).

Achieving a correct translation (the expected interpretation) is straightforward: we require each anchor relation to be placed in exactly the negation scope that it appears in the TRC expression.

Example 5 (Example 4 continued). Our representation replaced the exact position of the join predicate with the anchor relation, which nests it in the correct negation scope:

$$\exists r \in R[\neg(\exists s \in S[\exists j \in \text{“}<\text{”}[r.A = j.\$1 \wedge j.\$2 = s.B]])]$$

5.3 Relational Diagrams with anchor relations are pattern-complete for TRC

We are ready to state our first of two main results of this paper.

Theorem 7 (Full pattern expressiveness). *There is an algorithm that translates any well-formed TRC query into Relational Diagrams extended with anchor relations. That representation has an unambiguous logical interpretation (there is another algorithm that translates that diagram back into TRC) and has the same atoms as the original TRC query and thus has the same relational pattern.*

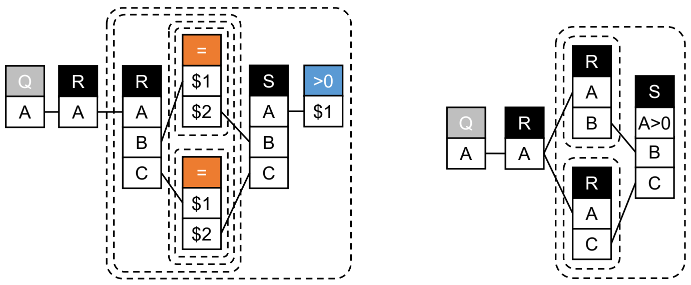
The proof consists of constructive, pattern-preserving translations in both directions between TRC and Relational Diagrams with anchor relations. We next give the translation in one direction and provide the other direction in the full version [32]. Both directions together provide a solution to the disjunction problem.

5.3.1 Translation from TRC$^{\neg\exists\wedge}$ to Relational Diagrams with anchor relations. We next give the straightforward 4-step translation from well-formed TRC$^{\neg\exists\wedge}$ queries into Relational Diagrams extended with anchor relations. This translation is heavily inspired by the 5-step translation given in [33] for Relational Diagrams, however it differs in crucial steps: By showing that our translation preserves all atoms (which includes the assignment predicates) for *all* well-formed TRC queries, we also show that our variant can express all relational patterns of TRC. We illustrate the translation with query (5) from Example 3 displayed in Fig. 5a.

(1) Preprocessing: First, we translate any well-formed TRC into TRC$^{\neg\exists\wedge}$ by preserving its atoms as described in Section 5.1. Then, we replace every join and selection predicate with the corresponding anchor relations as described in Section 5.2. Equijoin predicates ($R.A = S.B$) that occur in the same negation scope as one of their relations R or S do not need to be replaced (e.g. this is the case in Fig. 4c after replacing the "<" operator with "=", but not in Fig. 1c and Example 9). As example, query (5) is written as:

$$\begin{aligned}\{q(A) \mid \exists r \in R[q.A = r.A \wedge \neg(\exists s \in S, c \in \text{“}>0\text{”}[s.A = c.\$1 \wedge \\ \neg(\exists r_2 \in R[\neg(\neg(\exists j_1 \in \text{“}=\text{”}[r_2.B = j_1.\$1 \wedge j_1.\$2 = s.B]) \wedge \\ \neg(\exists j_2 \in \text{“}=\text{”}[r_2.C = j_2.\$1 \wedge j_2.\$2 = s.C])) \wedge r_2.A = r.A])])]\}\end{aligned} \tag{10}$$

Notice that the equijoins "$q.A = r.A$" and "$r_2.A = r.A$" are not replaced with anchor relations



(a) (b)

Fig. 5. (a) Section 5.3.1: Pattern-preserving diagrammatic representation with anchor relations for query (5) from Example 3 via query (10). (b) Section 7.1: Relational Diagrams can't preserve the pattern (Example 9) and need to translate from (6).

(2) Creating canvas partitions: Similar to Relational Diagrams, The scopes of the negations in a TRC are nested by definition. We translate the nested hierarchy of negation scopes (the *negation hierarchy*) into a nested partition of the canvas by dashed boxes with rounded corners.

(3) Placing input, anchor, and output relations: Relation names from each binding atom are placed into the canvas partition that corresponds to the respective negation scope. Notably, anchor relations are treated identically as other relations. In particular, multiple occurrences of the same anchor relation (e.g. <) lead to separate relations being placed in appropriate partitions. For our example, the binding $\exists j_1 \in$ "=" is represented by an anchor relation $\boxed{=}$ in nesting level 4 of the negation hierarchy. Clearly inspired by Relational Diagrams, the output relation Q for non-Boolean queries and all attributes from its header are represented with a relation $\boxed{Q}$ outside all nesting scopes, which we refer to as the *base partition* in reference to the AST. Recall that for well-formed TRC queries $\{q(\mathbf{H}) \mid \varphi\}$, φ can contain only one single free variable q.

(4) Placing equijoin predicates: In contrast to Relational Diagrams, both join and selection predicates are now treated identically as equijoins with appropriate anchor relations. For each equijoin $R.A = S.B$ in the query, we simply add two attributes, one for each relation R and S and connect them via an unlabeled edge. For table attributes that occur in multiple equijoins, we only draw one attribute and connect it to multiple edges. Notice that by this construction, the attributes of each occurrence of an anchor relation are connected to exactly one edge. For example, "$r_2.B = j_1.\$1$" is represented with an edge between attribute $\boxed{B}$ of the 2nd occurrence of relation $\boxed{R}$ with attribute $\boxed{\$1}$ of the first occurrence of anchor relation $\boxed{=}$.

Completeness. This 4-step translation guarantees the uniqueness of the following aspects: (1) nesting hierarchy (corresponding to the negation hierarchy), (2) where input and anchor relations are placed (canvas partitions corresponding to the negation scope), (3) which relation attributes participate in equijoins with what other relational attributes. Notice that due to our unified treatment of selection and join predicates as equijoins with appropriate anchor relations, our translation (after preprocessing) is slightly simpler than the one originally proposed by the authors of Relational Diagrams [33] and, more importantly, also more general: *Any well-formed TRC query* can be represented in a way that preserves all atoms and thus also relational patterns.

6 Representation B (without anchor relations)

Our preliminary solution from the previous section solves the disjunction problem and even reduces the necessary vocabulary.[14] However, it has arguably two important problems: (1) We lost the ability to use standard syntactic safety conditions to determine whether a query is domain-independent. (2) The anchor relations and multiple nestings of negations increase the visual complexity and make the diagrams hard to read.[15] The two solutions we introduce in this section are conceptually simple: In Section 6.1 we substitute the anchor relations with visual formalisms proposed in prior literature, yet keep our rigorous and principled semantics defined earlier (recall Fig. 4d). Section 6.2 reintroduces disjunctions as (visual) shortcuts for our earlier rigorous semantics. The result is a precisely defined pattern-preserving diagrammatic representation for *any TRC query* that allows visual verification of the safety conditions and that specializes into Relational Diagrams for the fragment of disjunction-free TRC.

6.1 Substituting anchor relations

6.1.1 Simpler anchors for unary anchor relations. Unary anchor relations consist of two boxes: the predicate (condition) name (e.g. $\boxed{<4}$) and an attribute box $\boxed{\$1}$. We eliminate this unnecessary indirection[16] and substitute both boxes with one box containing the condition (e.g. $\boxed{<4}$). We thereby also recover visual formalisms from prior proposals, such as VisualSQL [47] and VQL [56] (see Figs. 2d and 2e): a selection is an equijoin between an attribute and a condition (e.g. $\boxed{A}$—$\boxed{<4}$). That condition still provides an anchor and could be in a deeper nesting than the table. We call this the "canonical" representation.

If the condition is in the same negation scope as the relation, then we apply a shortcut that fuses the two attributes and thereby recovers the selection formalism used by Relational Diagrams (e.g. $\boxed{A<4}$).[17] Our formalism is thus backward compatible to Relational Diagrams, yet also allows us to give the condition a separate "anchor", which we need to express certain relational patterns.

6.1.2 Binary anchor relations. Binary anchor relations consist of three boxes: The predicate name (e.g. $\boxed{<}$) and two attributes connected to the respective relational attributes via equijoins. We substitute these anchor relations with the symbols that were originally used by Relational Diagrams *as labels* on directed edges (arrows), however we give them an explicit bounding box (e.g. ⧀). The important difference is that we treat the former *labels* now as *anchors* with the full semantic interpretation we developed in the last section (see e.g. Fig. 6b). This semantics allows us to explicitly place the anchor in a deeper nesting than either of the relations joined by that comparison predicate and thereby improve upon the limited pattern expressiveness of Relational Diagrams.

Example 6 (Substituting anchor relations). Consider the Boolean TRC query $\exists r \in R[\neg(r.A < 4)]$ shown in the lower row of Fig. 6a as Representation B. The top row shows Relational Diagrams with anchor relations which correspond to $\exists r \in R[\neg(\exists c \in \text{"}{<}4\text{"}[r.A = c.\$1])]$. Next, consider the Boolean query $\exists r \in R[\neg(\exists s \in S[\neg(r.A < S.B)])]$ shown as Representation B on the bottom of

[14] Our preliminary solution has fewer primitive diagrammatic elements than Relational Diagrams. A selection $R.A > 0$ is simply represented as equijoin $\boxed{A}$—$\boxed{\$1}$ where $\boxed{\$1}$ is the only attribute of a table $\boxed{>0}$ and $\boxed{A}$ is an attribute of a table $\boxed{R}$. Thus, there is no "boxed selection" as in $\boxed{A>0}$, edges have no labels and no arrows, and there are no "union cells".

[15] There is a reason why we have the disjunction operator in logic and natural language, although it is not strictly necessary. Citing from [8] on disjunctions: "*...the fewer connectives we have, the harder it is to understand our sentences.*"

[16] This is similar in spirit to Tufte's recommendation to avoid legends if possible: "*labels are placed (directly) on the graphic itself; no legend is required.*" [74]

[17] We use a slight blue background for selection conditions as enabling feature (Section 2.1.1), similar to the yellow background used by QueryVis [19].

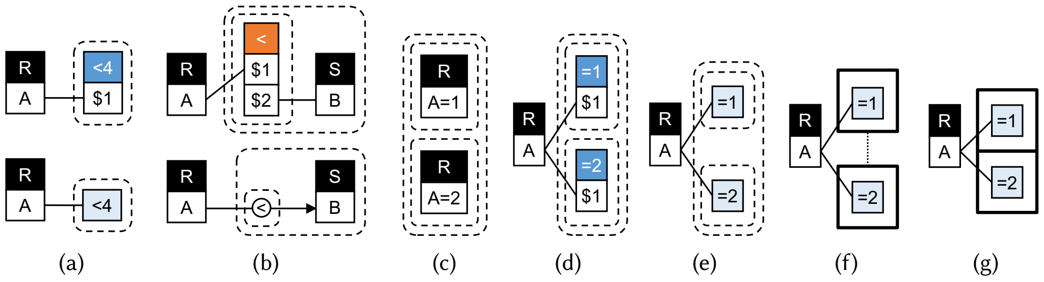


Fig. 6. Example 6(a-b): Representation B at the bottom, and Relational Diagrams with anchor relations at the top. Example 7 (c-g): $\exists r \in R[r.A=1 \vee r.A=2]$: (c): Relational Diagrams. (d): Relational Diagrams with anchor relations. (e-g): Representation B. (f-g) show our *visual shortcut* for disjunctions, formally justified with what we refer to as "De Morgan-fuse boxes".

Fig. 6b. The top shows Relational Diagrams with anchor relations which correspond to $\exists r \in R[\neg(\exists s \in S[\neg(\exists j \in \text{"<"}[r.A = j.\$1 \wedge j.\$2 = s.B])])]$

6.2 Visual shortcut for disjunctions

As already mentioned earlier, frugality in primitive elements has downsides: The fewer connectives we have, the harder it is to understand our sentences. Despite negation and conjunction being truth-functionally complete, we regularly use disjunctions in logic and natural language. We next introduce a visual shortcut for disjunction that allows us to recover a testable safety condition and that generalizes the various (incomplete) approaches for disjunctions we have seen in Section 3.2.

Our key idea is to *keep the formal semantics we have developed so far* (and that solves the disjunction problem), but to allow a visual shortcut that we refer to as "De Morgan-fuse boxes." These boxes allow us to express $\neg(\neg(\varphi_1) \wedge \ldots \wedge \neg(\varphi_k))$, $k \geq 2$ with $\varphi_1 \vee \ldots \vee \varphi_k$ by substituting nested double negations with bold rectangles, optionally connected via dotted lines.

Definition 8 (De Morgan-fuse boxes). *Bold rectangular boxes that are adjacent or connected via dotted lines represent disjunctions over their contents. Within each box, anchored elements are implicitly conjoined (i.e., interpreted as a conjunction via juxtaposition).*

The overall interpretation of De Morgan-fuse boxes follows De Morgan's transformation: each box is treated as being enclosed in its own negation scope, and all such boxes are wrapped by an outer negation. That is, a disjunction of conjunctions is represented as the negation of a conjunction of negated boxes.

Example 7 (Simple disjunction). Consider the following disjunction from Section 3.2:

$$\exists r \in R[r.A=1 \vee r.A=2] \tag{11}$$

Relational Diagrams need to show two R tables, either with union cells (Fig. 2h) or with a double negation (Fig. 6c):

$$\exists r_1 \in R[r_1.A=1] \vee \exists r_2 \in R[r_2.A=2]$$
$$\neg(\neg(\exists r_1 \in R[r_1.A=1] \vee \exists r_2 \in R[r_2.A=2]))$$
$$\neg(\neg(\exists r_1 \in R[r_1.A=1]) \wedge \neg(\exists r_2 \in R[r_2.A=2]))$$

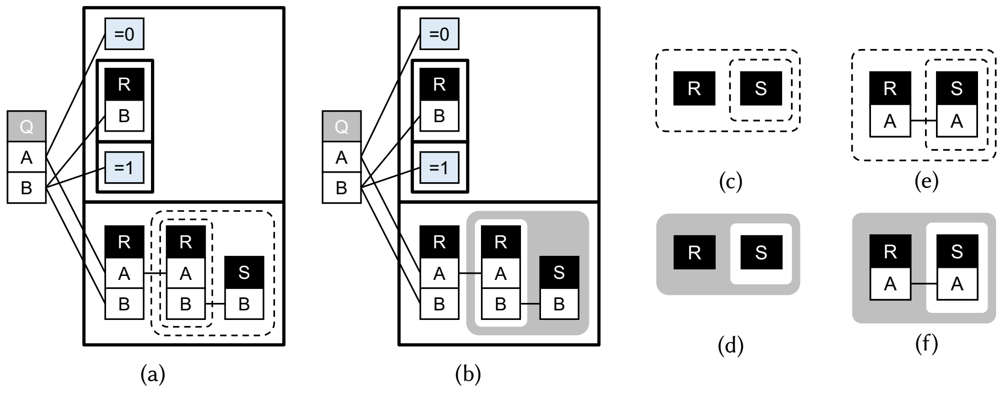


Fig. 7. (a) Example 8: Representation B for Example 2 and Fig. 3. (b) Section 7.1: after applying Peirce shading to (a). (c)-(f) Section 7.1: Peirce shading allows alternative reading of $\neg(antecedent \wedge \neg(consequent))$ as $antecedent \rightarrow consequent$.

Relational Diagrams with anchor relations replace the selection predicates with equijoins to two anchor relations (Fig. 6d):

$$\exists r \in R[\neg(\neg(\exists e_1 \in \text{“=1”}[r.A = e_1.\$1]) \wedge \neg(\exists e_2 \in \text{“=2”}[r.A = e_2.\$1]))]$$

Representation B can represent the statement either via De Morgan (Fig. 6e):

$$\exists r \in R[\neg(\neg(r.A = 1) \wedge \neg(r.A = 2))]$$

or via De Morgan-fuse boxes (Fig. 6g), optionally connected via dotted edges (Fig. 6f).

6.3 Representation B is pattern-complete for TRC and preserves safety

We are ready to state our second of two main results of this paper.

Theorem 9 (Pattern-isomorphism and safety of Representation B). *Every Relational Diagram with anchor relations for built-in predicates has a pattern-isomorphic representation as Representation B and vice versa. At the same time, Representation B preserves the safety conditions of TRC, i.e. the syntactic safety conditions can be directly verified from the diagrammatic representation.*

The proof uses constructive translations from TRC to Representation B and back that are [Error: Link “pattern-preserving” does not exist] and safety-preserving. Representation B thus solves both the disjunction and the safety problem. Recall that $\text{TRC}^{\neg\exists\wedge\vee}$ preserves the relational pattern and the safety conditions. Because our translation from $\text{TRC}^{\neg\exists\wedge\vee}$ preserves the negation scopes, the disjunctions, and all atoms from the AST, the 4 safety conditions from Section 4.2 can be immediately read and verified from a Representation B diagram. Figure 7 discusses our running example.

Example 8 (Example 2 continued). The TRC query from Example 2 and its AST from Fig. 3 is equivalent to the following safe $\text{TRC}^{\neg\exists\wedge\vee}$ fragment:

$$\begin{aligned} \{q(A, B) \mid (&q.A = 0 \wedge (\exists r \in R[q.B = r.B] \vee q.B = 1)) \\ &\vee (\exists r_2 \in R[q.A = r_2.A \wedge q.B = r_2.B \wedge \\ &\quad \neg(\exists s \in S[\neg(\exists r_3 \in R[r_2.A = r_3.A \wedge r_3.B = s.B])])])\} \end{aligned} \tag{12}$$

Figure 7a shows Representation B for this query. Notice how the 4 *explicit safety* conditions from Section 4.2 can be applied directly on this diagram to verify that this query is explicitly safe.

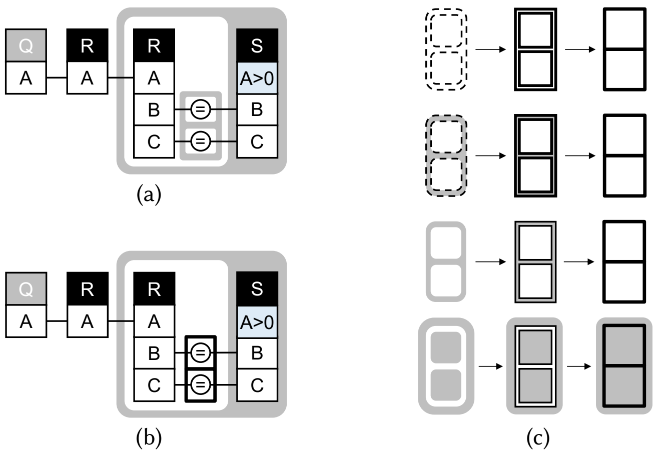


Fig. 8. (a), (b) Example 9: Peirce shading applied to Representation B by translating from query (5) ($\mathrm{TRC}^{\neg\exists\wedge}$) (a), or (b) directly from query (4) ($\mathrm{TRC}^{\neg\exists\wedge\vee}$) with our notation for disjunctions: De Morgan-fuse boxes (b). (c) Section 7.1: De Morgan-fuse boxes do not change the parity of a zone and thus effortlessly interact with Peirce shading.

6.4 Size of representation

Representation B has the same asymptotic size as TRC. The proof follows from the fact that the leaves of the AST (and the negation and disjunction scopes) get directly mapped to objects in the diagram. At the same time, Representation B is an exponentially smaller representation of TRC than Relational Diagrams. This is because Relational Diagrams require CNF formulas to be first transformed into DNF (i.e. to have disjunctions or unions as the root, which requires an exponential blow-up in size), while *our approach leaves disjunctions as inner operators in the AST.*

Proposition 10 (Size preservation of TRC). *Representation B has the same asymptotic size as TRC and can be exponentially smaller than Relational Diagrams.*

7 Enabling Features, Perceptual Choices, and Generality of Representation B

We add an enabling feature to Representation B (Section 7.1) and justify its perceptual choices (Section 7.2). The full version [32] additionally shows how Representation B unifies and generalizes prior representations for disjunctions.

7.1 Peirce shading as enabling feature

We next add alternative shading as enabling feature (Section 2.1.1) to Representation B. This idea was originally proposed by Peirce [58, Paragraph 4.621] and became known in the diagrammatic reasoning community due to Sowa [66, 67]. Define the parity of a zone in the diagram as *positive* if it is nested within an even number of negation scopes (including zero), and *negative* if it is nested within an odd number of negation scopes. Then, to improve contrast and facilitate reading, shade negative areas in gray and keep positive areas in white.

A fascinating aspect of Peirce shading is that it is a surprisingly effective enabling feature that "enables" multiple readings of a given diagram: It helps humans *read universal quantifiers without a need for an additional dedicated symbol.* Peirce called a nesting of a positive zone within a negative zone (as in Fig. 7c) a *scroll* and observed that it can be alternatively read as "*either R is false or S is true*" or "*if R is true, then S is true.*" Peirce shading (Fig. 7d) makes the second reading more explicit.

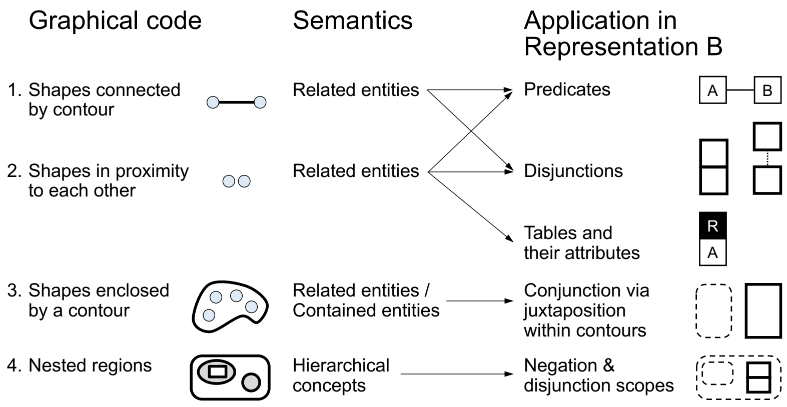

Graphical code
Semantics
Application in Representation B
1. Shapes connected by contour
Related entities
Predicates
A
B
2. Shapes in proximity to each other
Related entities
Disjunctions
Tables and their attributes
R
A
3. Shapes enclosed by a contour
Related entities / Contained entities
Conjunction via juxtaposition within contours
4. Nested regions
Hierarchical concepts
Negation & disjunction scopes

Fig. 9. Section 7.2: Our design choices in the larger context of the visual grammar of node-link diagrams and relationship representations. The notation in this figure is heavily inspired by Ware's "semantic pattern mappings" [80], but modified to our context of logical diagrams.

Similarly, consider the Boolean query (Fig. 7e)

$$\forall r \in R[\exists s \in S[r.A = s.A]] \tag{13}$$

Without universal quantifiers, it needs to be rewritten as

$$\neg(\exists r \in R[\neg(\exists s \in S[r.A = s.A])]) \tag{14}$$

Applying then Peirce shading to it, as in Fig. 7f, recovers the former reading more easily.

See also Fig. 7b which applies Peirce shading to Fig. 7a from Example 8.

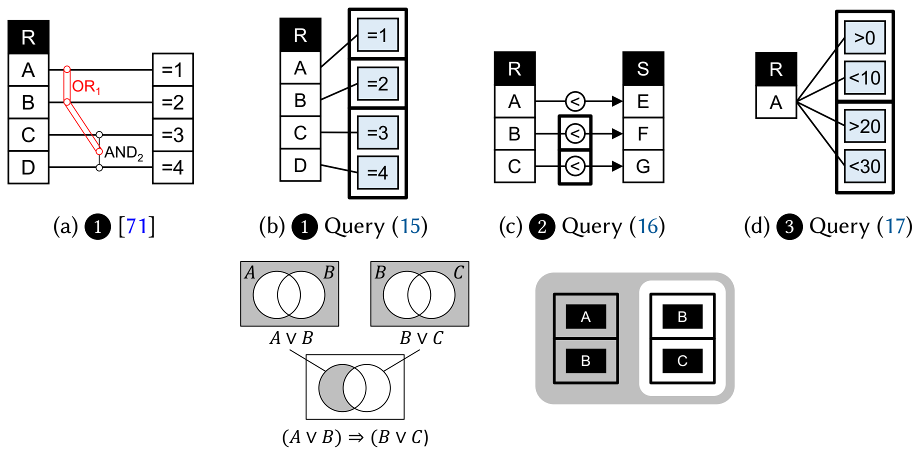
Example 9 (Example 3 continued). Figure 5a showed Example 3 as Relational Diagram with anchor relations. Figure 8a now shows Representation B by translating from query (5) in TRC$^{\neg\exists\wedge}$. In contrast, Fig. 8b shows Representation B by translating directly from the TRC$^{\neg\exists\wedge\vee}$ query (4) and using disjunctions. Both are pattern-isomorphic representations of the original formulas.

Since the semantics of our De Morgan-fuse boxes hinges on double negation, they *do not change the parity of a zone* in which they appear. Peirce shading therefore aligns naturally with disjunction, and disjunction boxes can be seen as visual shortcuts, providing a post hoc justification for the name De Morgan-fuse boxes (Fig. 8c).

7.2 Perceptual justification of our solution

Our main contribution in this paper is a representation system that solves the disjunction and the safety problems with its constitutive features. We discuss here some of our perceptual choices and the enabling features of Representation B that facilitate interpretation but are not constitutive.

Chamberlin in his recent CACM article on SQL [16] states "*Our specific goals were to design a query language with the following properties: ... user with no specialized training could, in simple cases, understand the meaning of a query simply by reading it. We called this the 'walk-up-and-read' property.*" Diehl [23] writes: "*If done right, diagrams group relevant information together to make searching more efficient, and use visual cues to make information more explicit.*" Similarly, our goal was to develop an (*i*) intuitive and (*ii*) principled diagrammatic representation for (*iii*) arbitrary nestings of disjunctions, (*iv*) with minimal additional notations. Our solution Representation B is heavily inspired by the design choices of Relational Diagrams and meets the challenge without introducing any fundamentally novel visual symbol to the visual grammar of Relational Diagrams.



(a) ❶ [71] (b) ❶ Query (15) (c) ❷ Query (16) (d) ❸ Query (17)

(e) ❹ [28, Fig. 44] (f) ❹ Query (18)

Fig. 10. Section 8.1: Representation B representations for challenging examples raised in prior work.

We merely give a stricter syntactic interpretation of edge label anchor boxes for built-in predicates, allow constants outside table attributes, and allow disjunction boxes used inside diagrams.

Figure 9 shows how our conservative extension with disjunctions fits into the overall visual grammar and semantic patterns (sometimes called "codes") of node-link diagrams and relationship representations [79]. As Ware writes, "*a good diagram takes advantage of basic perceptual mechanisms that have evolved to perceive structure in the environment*" [79], and we tried to follow that practice. Notice how our choices of disjunctions inherit the "nested regions" code from negation scopes and De Morgan, while also using the "shapes in proximity" code (widely used and known from UML) and alternatively the "shapes connected by contour" code.

8 Solutions to prior challenging queries

We show how our solution solves prior representation challenges.

8.1 Examples from Section 1

Figure 10 gives solutions to 4 challenges listed in Section 1. We use the word "query" also for a statement (i.e. a Boolean query). Images from the original literature are given in the full version [32].

❶ Figure 10b shows Representation B for Fig. 10a, a visual representation given in two presentations [70, 71] by Thalheim. The representation reads as:

$$\exists r \in R[R.A=1 \vee R.B=2 \vee (R.C=3 \wedge R.D=4)] \tag{15}$$

❷ Figures 10c and 10d show Representation B for the two challenges listed in [33]:

$$\exists r \in R, s \in S[r.A < s.E \wedge (r.B < s.F \vee r.C < s.G)] \tag{16}$$

$$\exists r \in R[(r.A > 0 \wedge r.A < 10) \vee (r.A > 20 \wedge r.A < 30)] \tag{17}$$

❸ Gardner in his 1958 book 'Logic Machines and Diagrams' [28] discusses a challenging disjunction and concludes that "*there seems to be no simple way in which the statement, as it stands, can be diagrammed*" [28, Section 4.3]:

$$(A \vee B) \rightarrow (B \vee C) \tag{18}$$

He proposes what he calls a *diagrammatic compound statement* that needs to repeat the individual predicates (Fig. 10e). Figure 10f shows Representation B for query (18), which follows from rewriting

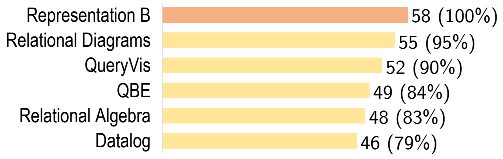


Fig. 11. Section 8.2: Fraction of pattern-isomorphic representations among 58 queries across 5 textbooks. Representation B is the first diagrammatic representation to achieve 100% coverage.

the statement as $\neg((A \vee B) \wedge \neg(B \vee C))$. Notice atomic propositions like A can be interpreted as nullary (0-ary) relations $A()$ that can be set to true or false. In other words, a symbol "A" is true if and only if "$\exists a \in A$", i.e. thus table A is not empty. Also notice that this example illustrates the value of treating disjunction as a primitive operator and diagrammatic element. Rewriting the disjunction using De Morgan's laws results in a significantly more complex expression: $\neg(\neg(\neg A \wedge \neg B) \wedge (\neg B \wedge \neg C))$.

❹ Most interesting is Fig. 1a, which was raised as a challenge in the earlier mentioned tutorial [31]. Here we point out an issue that – to the best of our knowledge – has never been raised in the literature on visual representation and diagrammatic reasoning: There are two different ways to interpret the figure shown in Fig. 1a:

$$\exists r \in R[(\exists s \in S[r.A = s.A] \wedge r.B = 0) \vee (r.B = 1 \wedge r.C = 2)] \tag{19}$$

$$\exists r \in R, s \in S[(r.A = s.A \wedge r.B = 0) \vee (r.B = 1 \wedge r.C = 2)] \tag{20}$$

The difference is that query (19) is true on the database $R = \{(9, 1, 2)\}$, $S = \emptyset$, whereas query (20) is false. The reason is the different scoping of S. Our principled translation into Representation B with De Morgan-fuse boxes creates the two *distinct* diagrams shown in Fig. 1 and can thus handle the distinction, as expected. We do not know any prior diagrammatic representation that could represent and thus distinguish between those two interpretations.

8.2 100% coverage of textbook benchmark

The authors of Relational Diagrams [33] gathered 58 queries from the relationally complete fragment from 5 popular database textbooks [18, 20, 24, 61, 65] and made them available on OSF.[18] We refer to that set simply as "the textbook benchmark." They evaluated the pattern expressiveness of various text-based and diagram-based languages (we replicate their numbers) and showed that Relational Diagrams covered 95% (55/58) of the queries in that benchmark. Our approach is pattern-complete for TRC and thus achieves 100% pattern coverage (Fig. 11).

Figure 12 shows one of 3 queries that Relational Diagrams cannot pattern-represent, its AST, and its pattern-isomorphic representation in Representation B: "Make a list of project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as manager of the controlling department for the project" [24, Query 4, Ch. 8.6]. The other two queries are given in the full version [32].

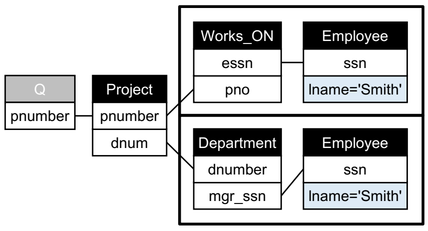
9 Conclusion

We proposed Representation B, a principled diagrammatic representation system that can express any well-formed TRC query without changing its table signature, thereby solving the disjunction problem, a long-standing gap around disjunctions in diagrammatic query representations. Our solution rests on 3 ideas: (*i*) reifying join and selection predicates as anchor relations, (*ii*) giving existing visual notations a clean relation-based semantics, and (*iii*) introducing De Morgan-fuse boxes as a visual formalism for disjunction that can be applied to subtrees of the AST representation

[18] Textbook benchmark: https://osf.io/u7c4z/. Reproducibility report: [78].

Q4: { *p*.Pnumber | PROJECT(*p*) **AND** (((∃*e*)(∃*w*)(EMPLOYEE(*e*)
AND WORKS_ON(*w*) **AND** *w*.Pno=*p*.Pnumber
AND *e*.Lname='Smith' **AND** *e*.Ssn=*w*.Essn))
OR
((∃*m*)(∃*d*)(EMPLOYEE(*m*) **AND** DEPARTMENT(*d*)
AND *p*.Dnum=*d*.Dnumber **AND** *d*.Mgr_ssn=*m*.Ssn
AND *m*.Lname='Smith')))}

(a) [24, Query 4, Ch. 8.6]



(b) Representation B

```
QUERY
├─ OUTPUT: Q(pnumber)
└─ QUANTIFIER ∃
   ├─ BINDING: p ∈ Project
   └─ AND ∧
      ├─ PREDICATE: Q.pnumber = p.pnumber
      └─ OR ∨
         ├─ QUANTIFIER ∃
         │  ├─ BINDING: w ∈ Works_on
         │  ├─ BINDING: e ∈ Employee
         │  └─ AND ∧
         │     ├─ PREDICATE: w.pno = p.pnumber
         │     ├─ PREDICATE: w.essn = e.ssn
         │     └─ PREDICATE: e.lname = 'Smith'
         └─ QUANTIFIER ∃
            ├─ BINDING: d ∈ Department
            ├─ BINDING: e ∈ Employee
            └─ AND ∧
               ├─ PREDICATE: d.dnumber = p.dnum
               ├─ PREDICATE: d.mgr_ssn = e.ssn
               └─ PREDICATE: e.lname = 'Smith'
```

(c) AST

Fig. 12. Section 8.2: Query from the textbook benchmark that cannot be represented in a pattern-isomorphic way by prior diagrammatic approaches and the solution in Representation B.

of TRC queries and that composes naturally with negation scopes. Together, these ideas unify and extend the 3 main prior diagrammatic approaches to disjunction.

Finding an accurate representation for the relationally complete fragment of relational query languages was important because all relational query languages such as SQL, and even relational programming languages such as Rel [5] are grounded in first-order logic. As large language models (LLMs) increasingly take over query generation, users need help in understanding the produced queries, especially as they grow more complex [35]. Automatically generated diagrammatic representations that preserve the relational patterns of queries and that support progressive disclosure (via collapse/expand interactions) can serve as an "explanation layer" and assist users in interpreting these complicated queries more effectively.

Many open problems remain, in particular: 1) How to support advanced language features from practical languages, such as aggregates, recursion, and bag semantics (see [36] for recent progress). 2) Our focus in this work was feasibility, not usability. Are there other topologically different representations and/or enabling features that help users understand queries faster and more accurately? Part of those questions may have to be evaluated with comparative user studies [33, 49]. We view Representation B as a foundation for a broader agenda in the context of human-AI collaboration where query understanding via visual query representations becomes part of the regular human-query interaction [29].

Acknowledgments

This work was supported in part by the National Science Foundation (NSF) under award IIS-1762268.

References

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases.* Addison-Wesley. http://webdam.inria.fr/Alice/

[2] Azza Abouzied, Joseph M. Hellerstein, and Avi Silberschatz. 2012. DataPlay: interactive tweaking and example-driven correction of graphical database queries. In *UIST.* ACM, 207–218. doi:10.1145/2380116.2380144

[3] Active Query Builder. 2019. https://www.activequerybuilder.com/.

[4] Michele Angelaccio, Tiziana Catarci, and Giuseppe Santucci. 1990. Query by diagram: A fully visual query system. *J. Vis. Lang. Comput.* 1, 3 (1990), 255–273. doi:10.1016/S1045-926X(05)80009-6

[5] Molham Aref, Paolo Guagliardo, George Kastrinis, Leonid Libkin, Victor Marsault, Wim Martens, Mary McGrath, Filip Murlak, Nathaniel Nystrom, Liat Peterfreund, Allison Rogers, Cristina Sirangelo, Domagoj Vrgoc, David Zhao, and Abdul Zreika. 2025. Rel: A Programming Language for Relational Data. In *Companion of SIGMOD/PODS 2025.* 283–296. doi:10.1145/3722212.3724450

[6] Marcelo Arenas, Pablo Barcelo, Leonid Libkin, Wim Martens, and Andreas Pieris. 2022. *Database Theory: Querying Data.* Open source at https://github.com/pdm-book/community.

[7] Eirik Bakke and David R. Karger. 2016. Expressive Query Construction through Direct Manipulation of Nested Relational Results. In *SIGMOD.* ACM, 1377–1392. doi:10.1145/2882903.2915210

[8] David Barker-Plummer, Jon Barwise, and John Etchemendy. 2011. *Language, Proof, and Logic: Second Edition* (2nd ed.). Center for the Study of Language and Information/SRI. https://www.gradegrinder.net/Products/lpl-index.html

[9] Jacques Bertin. 1981. *Graphics and graphic information-processing.* de Gruyter. doi:10.1515/9783110854688

[10] Filippo Bonchi, Alessandro Di Giorgio, Nathan Haydon, and Pawel Sobocinski. 2024. Diagrammatic Algebra of First Order Logic. In *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'24).* Article 16, 15 pages. doi:10.1145/3661814.366207

[11] Robert Brijder, Floris Geerts, Jan Van den Bussche, and Timmy Weerwag. 2019. MATLANG: Matrix operations and their expressive power. *SIGMOD Rec.* 48, 1 (2019), 60–67. doi:10.1145/3371316.3371331

[12] Tiziana Catarci, Maria Francesca Costabile, Stefano Levialdi, and Carlo Batini. 1997. Visual Query Systems for Databases: A Survey. *J. Vis. Lang. Comput.* 8, 2 (1997), 215–260. doi:10.1006/jvlc.1997.0037

[13] Tiziana Catarci and Giuseppe Santucci. 1994. Query by Diagram: A Graphical Environment for Querying Databases. In *SIGMOD.* 515. doi:10.1145/191839.191976

[14] Tiziana Catarci, Giuseppe Santucci, and Michele Angelaccio. 1993. Fundamental Graphical Primitives for Visual Query Languages. *Inf. Syst.* 18, 2 (1993), 75–98. doi:10.1016/0306-4379(93)90006-M

[15] Stefano Ceri, Georg Gottlob, and Letizia Tanca. 1989. What you Always Wanted to Know About Datalog (And Never Dared to Ask). *IEEE TKDE* 1, 1 (1989), 146–166. doi:10.1109/69.43410

[16] Donald D. Chamberlin. 2024. 50 Years of Queries. *Commun. ACM* 67, 8 (2024), 110–121. doi:10.1145/3649887

[17] Peter P. Chen. 1976. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Trans. Database Syst.* 1, 1 (1976), 9–36. doi:10.1145/320434.320440

[18] Thomas M. Connolly and Carolyn E. Begg. 2015. *Database Systems: A Practical Approach to Design, Implementation and Management, Global Edition* (5 ed.). Pearson Addison Wesley. https://www.pearson.com/en-gb/subject-catalog/p/database-systems-a-practical-approach-to-design-implementation-and-management-global-edition/P200000003964/

[19] Jonathan Danaparamita and Wolfgang Gatterbauer. 2011. QueryViz: Helping Users Understand SQL queries and their patterns. In *EDBT demos.* ACM, 558–561. doi:10.1145/1951365.1951440 Project page: https://queryvis.com/.

[20] Christopher J. Date. 2003. *An introduction to database systems* (8 ed.). Pearson/Addison Wesley Longman. https://dl.acm.org/doi/10.5555/861613

[21] C. J. Date and Hugh Darwen. 2000. *Foundation for future database systems: the third manifesto* (2nd ed ed.). Addison-Wesley Professional. https://dl.acm.org/doi/10.5555/556540

[22] Frithjof Dau. 2009. The Advent of Formal Diagrammatic Reasoning Systems. In *ICFCA (International Conference on Formal Concept Analysis) (LNCS, Vol. 5548).* Springer, 38–56. doi:10.1007/978-3-642-01815-2_3

[23] Stephan Diehl. 2007. *Software visualization: visualizing the structure, behaviour, and evolution of software.* Springer. doi:10.1007/978-3-540-46505-8

[24] Ramez Elmasri and Sham Navathe. 2015. *Fundamentals of database systems* (7 ed.). Addison Wesley. https://dl.acm.org/doi/book/10.5555/2842853

[25] Herbert Enderton and Herbert B Enderton. 2001. *A mathematical introduction to logic.* Elsevier. doi:10.1016/C2009-0-22107-6

[26] George Englebretsen. 1996. Review of Sun-Joo Shin, The Logical Status of Diagrams. *Modern Logic* 6, 3 (1996), 322–330. https://projecteuclid.org/journals/modern-logic/volume-6/issue-3/Review-of-Sun-Joo-Shin-The-Logical-Status-of-Diagrams/rml/1204835743.full

[27] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. 2018. Cypher: An Evolving Query Language for Property Graphs. In

SIGMOD. 1433–1445. doi:10.1145/3183713.3190657

[28] Martin Gardner. 1958. *Logic Machines and Diagrams*. McGraw-Hill. https://archive.org/details/logicmachinesdia00mart

[29] Wolfgang Gatterbauer. 2011. Databases will Visualize Queries too. *PVLDB* 4, 12 (2011), 1498–1501. https://www.vldb.org/pvldb/vol4/p1498-gatterbauer.pdf Recorded Talk: https://www.youtube.com/watch?v=kVFnQRGAQls. Slides: https://gatterbauer.name/download/vldb2011_Database_Query_Visualization_presentation.pdf. Project page: https://queryvis.com/.

[30] Wolfgang Gatterbauer. 2023. A Tutorial on Visual Representations of Relational Queries. *PVLDB* 16, 12 (2023), 3890–3893. doi:10.14778/3611540.3611578 Tutorial page: https://northeastern-datalab.github.io/visual-query-representation-tutorial/, Slides: https://northeastern-datalab.github.io/visual-query-representation-tutorial/slides/VLDB_2023-Visual_Representations_of_Relational_Queries.pdf.

[31] Wolfgang Gatterbauer. 2024. A Comprehensive Tutorial on over 100 Years of Diagrammatic Representations of Logical Statements and Relational Queries. In *ICDE*. IEEE. doi:10.1109/ICDE60146.2024.00407 Tutorial page: https://northeastern-datalab.github.io/diagrammatic-representation-tutorial/, Slides: https://northeastern-datalab.github.io/diagrammatic-representation-tutorial/ICDE_2024-Diagrammatic-Representations-Tutorial.pdf.

[32] Wolfgang Gatterbauer. 2024. A Principled Solution to the Disjunction Problem of Diagrammatic Query Representations. doi:10.48550/ARXIV.2412.08583

[33] Wolfgang Gatterbauer and Cody Dunne. 2024. On The Reasonable Effectiveness of Relational Diagrams: Explaining Relational Query Patterns and the Pattern Expressiveness of Relational Languages. *Proc. ACM Manag. Data* 2, 1, Article 61 (2024), 61:1–61:27 pages. doi:10.1145/3639316 Recorded video: https://www.youtube.com/watch?v=IVRPD-074Ro. Slides: https://gatterbauer.name/download/sigmod2024-Relational-Diagrams-slides.pdf. Project page: https://relationaldiagrams.com/. Main supplemental material folder on OSF: https://osf.io/q9g6u/. Online appendix with all proofs, further illustrations, and study materials: https://arxiv.org/pdf/2401.04758. Textbook analysis: https://osf.io/u7c4z. User study tutorial: https://osf.io/mruzw. Stimuli-generating code: https://osf.io/kgx4y. The stimuli: https://osf.io/d5qaj. Stimuli/schema index CSV: https://osf.io/u8bf9. Stimuli/schema index JSON: https://osf.io/sn83j. Server code for hosting the study: https://osf.io/suj4a. Collected data: https://osf.io/8vm42. Executed user study analysis code: https://osf.io/f2xe3. Preregistered user study: https://osf.io/4zpsk/.

[34] Wolfgang Gatterbauer and Cody Dunne. 2025. Relational Diagrams and the Pattern Expressiveness of Relational Languages. *SIGMOD Record* 54, 1 (2025), 80–88. doi:10.1145/3733620.3733637 Project page: https://relationaldiagrams.com/.

[35] Wolfgang Gatterbauer, Cody Dunne, H. V. Jagadish, and Mirek Riedewald. 2022. Principles of Query Visualization. *IEEE Data Eng. Bull.* 45, 3 (2022), 47–67. http://sites.computer.org/debull/A22sept/p47.pdf

[36] Wolfgang Gatterbauer and Diandre Miguel Sabale. 2026. Database Research needs an Abstract Relational Query Language. In *Conference on Innovative Data Systems Research (CIDR)*. www.cidrdb.org. https://vldb.org/cidrdb/papers/2026/p27-gatterbauer.pdf

[37] Allen Van Gelder and Rodney W. Topor. 1991. Safety and Translation of Relational Calculus Queries. *ACM Trans. Database Syst.* 16, 2 (1991), 235–278. doi:10.1145/114325.103712

[38] Michael Genesereth and Eric J. Kao. 2017. *Introduction to Logic* (3rd ed.). Morgan & Claypool Publishers. doi:10.2200/S00734ED2V01Y201609CSL008

[39] Janice Glasgow, N. Hari Narayanan, and B. Chandrasekaran. 1995. *Diagrammatic Reasoning: Cognitive and Computational Perspectives*. MIT Press, Cambridge, MA, USA. https://dl.acm.org/doi/10.5555/546459

[40] Thomas R. G. Green and Marian Petre. 1996. Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework. *Journal of Visual Languages & Computing* 7, 2 (1996), 131–174. doi:10.1006/jvlc.1996.0009

[41] Paolo Guagliardo, Leonid Libkin, Victor Marsault, Wim Martens, Filip Murlak, Liat Peterfreund, and Cristina Sirangelo. 2025. Queries with External Predicates. In *ICDT (LIPIcs, Vol. 328)*. 22:1–22:20. doi:10.4230/LIPICS.ICDT.2025.22

[42] Nathan Haydon and Pawel Sobocinski. 2020. Compositional Diagrammatic First-Order Logic. In *11th International Conference on the Theory and Application of Diagrams (Diagrams 2020) (LNCS, Vol. 12169)*. Springer, 402–418. doi:10.1007/978-3-030-54249-8_32

[43] Marti A. Hearst. 2023. Show It or Tell It? Text, Visualization, and Their Combination. *Commun. ACM* 66, 10 (2023), 68–75. doi:10.1145/3593580

[44] John Howse. 2008. Diagrammatic Reasoning Systems. In *ICCS (International Conference on Conceptual Structures) (LNCS, Vol. 5113)*. Springer, 1–20. doi:10.1007/978-3-540-70596-3_1

[45] John Howse, Gem Stapleton, and John Taylor. 2005. Spider Diagrams. *LMS Journal of Computation and Mathematics* 8 (2005), 145–194. doi:10.1112/S1461157000000942 London Mathematical Society.

[46] Yannis E. Ioannidis. 1996. Visual User Interfaces for Database Systems. *ACM Comput. Surv.* 28, 4es (1996), 137. doi:10.1145/242224.242399

[47] Hannu Jaakkola and Bernhard Thalheim. 2003. Visual SQL – High-Quality ER-Based Query Treatment. In *ER (Workshops) (LNCS)*. Springer, 129–139. doi:10.1007/978-3-540-39597-3_13

[48] Jens Lemanski, Mikkel Willum Johansen, Emmanuel Manalo, Petrucio Viana, Reetu Bhattacharjee, and Richard Burns (Eds.). 2024. *14th International Conference on Diagrammatic Representation and Inference (Diagrams 2024)*. LNCS, Vol. 14981. Springer. doi:10.1007/978-3-031-71291-3

[49] Aristotelis Leventidis, Jiahui Zhang, Cody Dunne, Wolfgang Gatterbauer, H. V. Jagadish, and Mirek Riedewald. 2020. QueryVis: Logic-based Diagrams help Users Understand Complicated SQL Queries Faster. In *SIGMOD*. 2303–2318. doi:10.1145/3318464.3389767

[50] Leonid Libkin. 2003. Expressive power of SQL. *Theor. Comput. Sci.* 296, 3 (2003), 379–404. doi:10.1016/S0304-3975(02)00736-3

[51] Kenneth Manders. 2008. Diagram-Based Geometric Practice. In *The Philosophy of Mathematical Practice*. Oxford University Press. doi:10.1093/acprof:oso/9780199296453.003.0004

[52] Scott McCloud. 1993. *Understanding Comics: The Invisible Art.* Tundra. https://archive.org/details/understanding-comics

[53] Merriam-Webster.com. 2024. Diagram. https://www.merriam-webster.com/dictionary/diagram

[54] Microsoft Access. 2019. https://products.office.com/en-us/access.

[55] Daphne Miedema and George Fletcher. 2021. SQLVis: Visual Query Representations for Supporting SQL Learners. In *Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 1–9. doi:10.1109/VL/HCC51201.2021.9576431

[56] Lil Mohan and Rangasami L. Kashyap. 1993. A Visual Query Language for Graphical Interaction with Schema-Intensive Databases. *TKDE* 5, 5 (1993), 843–858. doi:10.1109/69.243513

[57] Oxford Dictionary of English. 2010. Diagram. doi:10.1093/acref/9780199571123.001.0001

[58] Charles Sanders Peirce. 1933. *Collected Papers*. Vol. 4. Harvard University Press. https://archive.org/details/collectedpaperso0000unse_r5j9/

[59] pgAdmin. 2019. https://www.pgadmin.org/.

[60] QueryScope. 2019. https://sqldep.com/.

[61] Raghu Ramakrishnan and Johannes Gehrke. 2002. *Database Management Systems* (3 ed.). McGraw-Hill, Inc., USA. https://dl.acm.org/doi/book/10.5555/560733

[62] Nicole Schweikardt, Thomas Schwentick, and Luc Segoufin. 2010. *Database theory: query languages* (2 ed.). Chapman & Hall/CRC, 19. doi:10.5555/1882723.1882742

[63] Sun-Joo Shin. 1995. *The Logical Status of Diagrams.* Cambridge University Press. doi:10.1017/CBO9780511574696

[64] Sun-Joo Shin. 2002. *The Iconic Logic of Peirce's Graphs.* The MIT Press. doi:10.7551/mitpress/3633.001.0001

[65] Avi Silberschatz, Henry F. Korth, and S. Sudarshan. 2020. *Database System Concepts* (7 ed.). McGraw-Hill Book Company. https://www.db-book.com/db7/index.html

[66] John F. Sowa. 2008. Chapter 5 Conceptual Graphs. In *Handbook of Knowledge Representation*, Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter (Eds.). Foundations of Artificial Intelligence, Vol. 3. Elsevier, 213–237. doi:10.1016/S1574-6526(07)03005-2

[67] John F. Sowa. 2011. Peirce's tutorial on existential graphs. *Semiotica* 2011, 186 (2011), 347–394. doi:10.1515/semi.2011.060

[68] SQL Server Management Studio. 2019. https://www.microsoft.com/en-us/sql-server/sql-server-downloads.

[69] Gem Stapleton. 2013. Delivering the Potential of Diagrammatic Logics. In *Proceedings of the First International Workshop on Diagrams, Logic and Cognition (DLAC 2013)*. CEUR. https://ceur-ws.org/Vol-1132/paper1.pdf

[70] Bernhard Thalheim. 2007. Tutorial Visual SQL: An ER-Based Introduction to Database Programming. Slide deck Christian-Albrechts-Universität zu Kiel.

[71] Bernhard Thalheim. 2013. Visual SQL as the Alternative to Linear SQL. Slide deck Christian-Albrechts-Universität zu Kiel.

[72] Silvia De Toffoli. 2022. What are mathematical diagrams? *Synthese* 200, 86 (2022). doi:10.1007/s11229-022-03553-w

[73] Rodney Topor. 2018. Safety and Domain Independence. In *Encyclopedia of Database Systems, 2nd ed*, Ling Liu and M. Tamer Özsu (Eds.). Springer. doi:10.1007/978-1-4614-8265-9_1255

[74] Edward R. Tufte. 1986. *The Visual Display of Quantitative Information.* Graphics Press. https://dl.acm.org/doi/book/10.5555/33404

[75] Barbara Tversky. 2017. *Information Design: Research and Practice.* Routledge, Chapter 21: Diagrams, 349–360. doi:10.4324/9781315585680

[76] Jeffrey D. Ullman. 1988. *Principles of Database and Knowledge-base Systems, Vol. I.* Computer Science Press, Inc. https://dl.acm.org/doi/book/10.5555/42790

[77] John Venn. 1880. I. On the diagrammatic and mechanical representation of propositions and reasonings. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 10, 59 (1880), 1–18. doi:10.1080/14786448008626877

[78] Giorgio Vinciguerra, Guang Yang, Wolfgang Gatterbauer, and Cody Dunne. 2025. Reproducibility Report for ACM SIGMOD 2024 Paper: 'On The Reasonable Effectiveness of Relational Diagrams'. In *Reproducibility Reports for SIGMOD 2025 (SIGMOD/PODS '24)*. ACM, 60–63. doi:10.1145/3687998.3717044

[79] Colin Ware. 2020. *Information Visualization: Perception for Design* (4 ed.). Morgan Kaufmann Publishers Inc. doi:10.1016/C2016-0-02395-1

[80] Colin Ware. 2021. *Visual Thinking for Information Design* (2 ed.). Morgan Kaufmann. doi:10.1016/C2016-0-01395-5

[81] xkcd. 2011. How standards proliferate. https://xkcd.com/927/ Discussion: https://www.explainxkcd.com/wiki/index.php/927:_Standards.

[82] Moshé M. Zloof. 1977. Query-by-Example: A Data Base Language. *IBM Systems Journal* 16, 4 (1977), 324–343. doi:10.1147/sj.164.0324

Received July 2025; revised October 2025; accepted November 2025