SHORT-PAPER

PatternVis: A Tool for Relational Pattern Visualization

DIANDRE MIGUEL SABALE, Northeastern University, Boston, MA, United States

WOLFGANG GATTERBAUER, Northeastern University, Boston, MA, United States

Published: 22 June 2025

SIGMOD/PODS '25: International Conference on Management of Data
June 22 - 27, 2025
Berlin, Germany

Conference Sponsors:
SIGMOD

https://doi.org/10.1145/3722212.3725128
ISBN: 9798400715648

PatternVis: A Tool for Relational Pattern Visualization

Diandre Miguel Sabale
Northeastern University
Boston, MA, USA
sabale.d@northeastern.edu

Wolfgang Gatterbauer
Northeastern University
Boston, MA, USA
w.gatterbauer@northeastern.edu

Abstract

We demonstrate PatternVis, an interactive tool that—given a SQL query and its schema—can represent the relational pattern of the query in an extended representation of Relational Diagrams. Notably, PatternVis can represent groupings and aggregation in addition to the non-disjunctive fragment of Relational Diagrams.

Our interactive demo has two main contributions: (1) To the best of our knowledge, this is the first interactive SQL visualization tool that can give an *unambiguous* representation of nested SQL queries with GROUP BY clauses and aggregation. (2) We present a set of tasks involving multiple-choice questions in which the goal is to identify the correct output of a given aggregation query and database instance. Users can choose to complete each task in one of three modalities (SQL only, PatternVis only, or both) and experience the varying difficulties in each modality.

CCS Concepts

• **Information systems** → **Relational database query languages**.

Keywords

Visual Query Representation, Logical Diagrams, Relational Patterns

ACM Reference Format:
Diandre Miguel Sabale and Wolfgang Gatterbauer. 2025. PatternVis: A Tool for Relational Pattern Visualization. In *Companion of the 2025 International Conference on Management of Data (SIGMOD-Companion '25), June 22–27, 2025, Berlin, Germany*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3722212.3725128

1 Introduction

Query understanding (or query reading) is the task of interpreting *the intent behind a database query* written in a specific query language. It is critical for database users and various applications, including query reuse [11] and query semantics debugging [5]. It is also becoming increasingly important for verifying the queries written by LLMs via text-to-query interfaces [10]. However, query reading can be a difficult cognitive task [16] as it requires users to be familiar with the syntax of the query language and notice subtle details (like join predicates) which impact the query semantics.

Query diagrams. Visualizations can help users understand complicated messages. A widely cited paper on learning styles found that "most people of college age and older (were) visual" and recommends using "schematics, graphs, and simple sketches" instead of verbal material alone [4]. Although visualizations are not always beneficial for program understanding [14], recent user studies have provided evidence that representing logical constructs such as relational queries as diagrams instead of text can help users understand them faster and more accurately [8, 12]. Thus, the goal of *query visualization* is to design diagrammatic representation systems for relational queries that improve query comprehension by depicting the logical structure of a given query [10]. Another user study has shown the benefits of such diagrams for query formulation [13].

Relational Patterns. Recent computational experiments show that applying a *relational bottleneck principle*–focusing on object relationships–enables rapid learning and pattern generalization in neural networks, potentially reflecting human cognition [19]. Diagrammatic representations may be a direct application of this principle, aligned with Gestalt principles that describe how the arrangement of visual elements relate to *perceived relationships* [20]. Thus, one of the key advantages of logical diagrams in general is that their overall structure emphasizes the *relationship* between objects involved in the query, rather than the syntax required for the query language to support the operation, allowing for efficient abstraction and generalization. The relationships between the table references in a query are captured by its *relational pattern*, which is the logical function the query represents after treating each table reference as independent base tables [9]. To be effective, query diagrams must be able to unambiguously represent all patterns of a given query language (i.e., they should be pattern-complete).

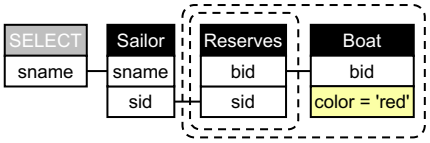
PatternVis. PatternVis uses a visual query representation that extends Relational Diagrams [2] and can represent SQL queries with groupings, aggregations, and nested queries.

> Example 1 (Universal query patterns). *Consider the universal query asking for "sailors who reserved all red boats" on the sailors database Sailor(sid, sname, rating, age), Boat(bid, bname, color), Reserves(sid, bid, day) from the "cow database textbook" [15].*
>
> *Figure 1 shows two ways to write this query with slightly different interpretations: Figure 1a shows a query pattern that uses only existential quantifiers and nested negation; it is thus expressible in relational calculus. Figure 1b, on the other hand, shows a query that uses grouping and aggregation yet no negation; it compares the count of different red boats reserved by each sailor with the total number of red boats.*
>
> *In particular, these two queries use significantly different underlying logic to express this intention. They then give slightly different results: The first query result includes sailors who have not reserved any boat, while the second one does not. With PatternVis, patterns involving grouping, aggregation, and arbitrarily nested queries (even in the HAVING clause) can be represented and compared.*

SIGMOD-Companion '25, Berlin, Germany

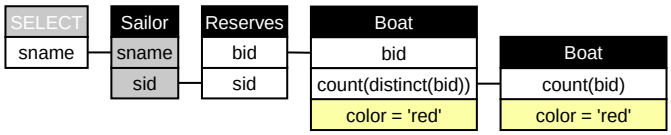
ACM ISBN 979-8-4007-1564-8/2025/06
https://doi.org/10.1145/3722212.3725128

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE NOT EXISTS
  (SELECT *
  FROM Boat B
  WHERE B.color = 'red'
  AND NOT EXISTS
    (SELECT *
    FROM Reserves R
    WHERE R.bid = B.bid
    AND R.sid = S.sid))
```



(a) Query using a pattern from first-order logic

```
SELECT DISTINCT S.sname
FROM Sailor S, Boat B, Reserves R
WHERE R.bid = B.bid
AND R.sid = S.sid
AND B.color = 'red'
GROUP BY S.sid, S.sname
HAVING COUNT(DISTINCT B.bid) =
  (SELECT COUNT(B.bid)
  FROM Boat B
  WHERE B.color = 'red')
```



(b) Query using grouping, aggregation, and nesting in the HAVING clause

Figure 1: Example 1 demonstrating two different ways to represent the same query finding "sailors who reserved all red boats"

1.1 Recent Designs for Query Visualization

QueryVis [2] is the closest work to PatternVis. QueryVis implemented an online demonstration that generated diagrams representing SQL queries. However, while the tool supported some forms of grouping, it was not relationally complete and did not support nested aggregate queries in the HAVING clause.

The original Relational Diagrams [8] improve certain aspects of QueryVis and allow an unambiguous visual representation of first-order logic (and thus any relationally complete language having the same expressiveness as relational algebra and relational calculus). PatternVis implements Relational Diagrams and extends them with support for grouping and aggregation.

SQLVis [13] provides a graph-based visual representation of queries. Because the system is designed to help users compose SQL queries, aspects of its visual design are tied to SQL syntax. Some logical relationships are expressed using text rather than diagrammatic elements. For instance, two tables joined by one or more predicates are depicted as connected by a single line.

DataPlay [1] is a tool that allows users to edit a query tree to receive feedback about updates to the output. Because the tool is catered towards interactive query writing rather than query representation, the visual representation does not emphasize the logic describing the relations between objects in the query. By design, the type of queries is limited to joins across foreign-primary key constraints, and there is no support for grouping.

Visual representations were used in a study by Sundin and Cutts to help users understand relational operations [18]. The authors illustrated the transformation between the input and output of an operation (like projection) with icons and example databases with different numbers of columns. Their focus is on transformations without negation. PatternVis also uses patterns to help users understand queries, but the patterns involve the whole query instead of individual data transformations. We focus on logical relationships between tables and their attributes instead of example databases.

A recent survey of visual query languages [17] noted that their main weaknesses were low expressiveness, poor support for software engineering workflows, and lack of evaluations. Another recent survey of query visualizations [6] focuses on the relationally complete fragment only, yet for that fragment gives a consistent comparison by translating the same set of queries into different visualizations. PatternVis addresses common issues with visual query languages while focusing on the complementary goal of query visualization, extending existing designs for a greater fragment of SQL and providing a simple tool where any user can easily input a query and quickly receive a principled diagram.

1.2 Contribution

PatternVis is the first interactive and unambiguous query visualization tool that supports the full disjunction-free fragment of relational calculus extended with nested grouping and aggregation. PatternVis also implements additional subqueries such as derived tables (nested queries in the FROM clause) and common table expressions (CTEs), which previous work could not support [2, 8]. The list of supported grammar is described in Fig. 4.

We also illustrate the usefulness of relational patterns in practical applications. To do so, we contribute a set of interactive activities that let users experience the difficulty of understanding queries in different modalities and experience how visualizing logical patterns can help them interpret a given SQL query.

2 Diagram Design

Identifying similar relational query patterns across languages is not trivial, as different languages may adopt distinct paradigms (like procedural or declarative) to express the same logical pattern. In addition, a language like SQL allows several syntactic variants to express the *same* query pattern. Our goal is to help a user abstract from the syntax and be able to focus on the pattern alone.

Our diagram design is an extension of Relational Diagrams [8], which covered tables, attributes, selection and join predicates, and

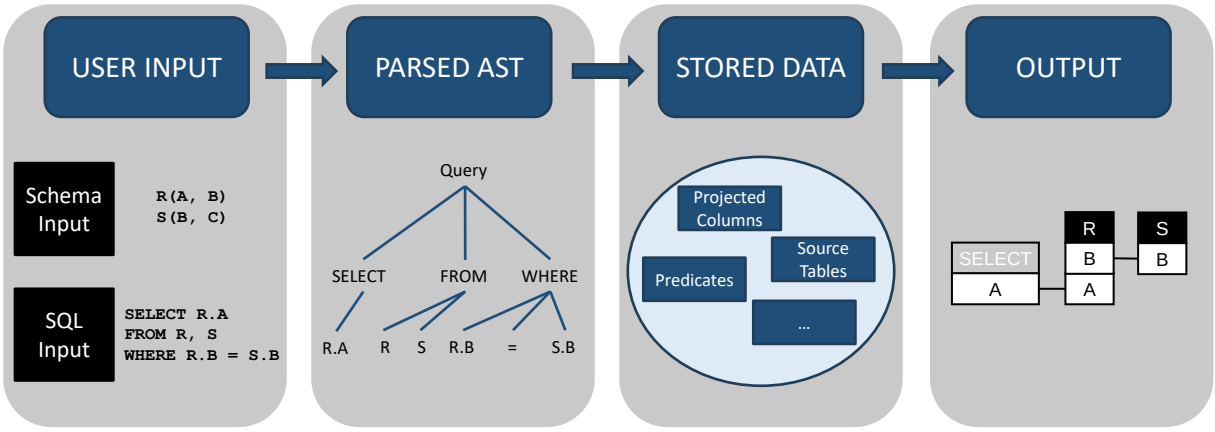


Figure 2: The workflow of PatternVis is similar to that of QueryVis but has an unambiguous and more expressive diagram design.

negation scopes. The visual formalism can express relational operations regardless of the syntactic variant of the query. Our visualization builds upon that design by adding visual constructs for grouping, aggregation, and arbitrary subqueries, even in the HAVING clause. Previously, the QueryVis demonstration supported only a limited set of aggregation, such as the example in Fig. 3 [2, 12]. The visualization features a query that groups table *R* by attribute *B* then returns an aggregate from each group. Meanwhile, Figure 1b shows a query using a nested query in the HAVING clause, which is currently not supported by any tool that we know of.

```
SELECT SUM(R.A) sum
FROM R
GROUP BY R.B
```

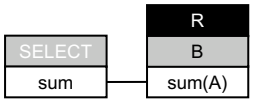


Figure 3: A simple query with grouping and aggregation.

We focus on the disjunction-free fragment of SQL with grouping, which covers most fundamental logical operations. Our design of grouping and aggregation is orthogonal to recent ideas on disjunction, so future updates may incorporate new paradigms such as disjunction boxes [7] into the visualization.

Another valuable example of the common query patterns that are represented distinctively by the visualization design of PatternVis are the four categorical propositions from logic (some/none/all/not all) [21]. These logical patterns can apply to many different contexts. For example, with the sailor schema from Example 1, we can find new expressions like "all red boats are owned by some sailors", "no red boats are owned by any sailors", just by substituting the schema and predicates.[1] By focusing the diagrams on the logical relations between table references of a query, the diagrams allow a generalizable interpretation of the pattern across different relational schemas and syntactic options.

[1]See https://relationaldiagrams.com/ for visual examples.

Overall, the diagram design used in PatternVis aims to use a minimal number of visual constructs to unambiguously represent a query's logic across any schema or syntax.

3 Implementation

PatternVis is implemented in Python. First, the schema is parsed from the input string using simple regex matching assuming that the schemas come in the form "TABLE_NAME (ATTR1, ATTR2,...)". The parsed schema is used to verify the validity of the given SQL query. Then, a SQL parser is used to parse an abstract syntax tree (AST) from the input SQL query string, which describes the structure of the elements present in the query. Next, this AST is processed by traversing specific nodes corresponding to specific SQL query clauses. Information about the fundamental query components for its logical representation, such as join predicates and source tables, is stored in intermediate data structures. After all the information is gathered, assignment of table positions in the final diagram is established by performing a breadth-first search on the (join) predicates that are stored. Tables involving predicates which are more closely connected to the attributes in the output will be assigned lower "levels" and placed closer to the left side of the diagram, with the output table always having the leftmost position. The general workflow for processing user-input schemas and queries is depicted in Fig. 2.

Once the necessary tables, attributes, joins, and predicates have been established, a layout algorithm is used to determine the positions of the elements. Future updates could further optimize the layout with algorithms like Stratisfimal Layout [3] to optimize the position of elements in the diagram and minimize collisions between edges.

PatternVis currently supports the relationally complete disjunction-free fragment of SQL, with extensions for grouping and aggregates. Note that grouping applies additional constraints on the contents of a query: if a column is used to group a query, then all other columns may only be further used as aggregates or the WHERE clause. Additionally, aggregates are not permitted in the WHERE clause. The full list of supported constructs can be found in Fig. 4.

Supported SQL Grammar	Definitions
Q::= [WITH T AS (Q)] SELECT E|V|* [[AS] A] {,E|V [[AS] A]} [FROM S|Q [[AS] T] {,S|Q [[AS] T]}] [WHERE P] [GROUP BY C|V {,C|V}] [HAVING P]	Query: Common Table Expression SELECT Clause FROM Clause WHERE Clause GROUP Clause HAVING Clause
E ::= C | G	Expression
C ::= [T.]A	Column
G ::= MIN|MAX|SUM|AVG|COUNT ([DISTINCT] C)	Aggregate
S ::= T [AS T]	Source (Table)
P ::= P [AND P] | E O E | E O V | [NOT] EXISTS (Q) | E [NOT] IN (Q) | E O (ALL|ANY (Q)) | E O (Q) | E IS [NOT] NULL	Predicate: Conjunction Join Selection Existential Membership Quantified Subquery Null
O ::= < | <= | = | >= |> | <> | !=	Operator
T ::=	Table (Identifier)
A ::=	Attribute (Identifier)
V ::=	Value (String or Numerical)

Figure 4: Grammar supported by PatternVis. Additionally, the usual sanity conditions for SQL apply.

4 Demonstration

The aim of the demonstration is to show that the diagrams from PatternVis serve as a broad and unambiguous way of identifying logical patterns (regardless of a wide variety of syntax options), and that they allow for efficient and intuitive interpretation of database queries. For the live demonstration, we offer multiple tasks:

(1) Explore Visualization Capabilities: Participants can experiment with visualizations of various SQL queries using a virtual tool which takes a schema and a query as input, then outputs a diagram representing the query logic. Becoming familiar with the diagram design (in particular the new support for grouping and aggregates) is helpful for participation in the other tasks. Before participants fully explore the tool, the demonstrators will load in and showcase a few examples of visualizations from existing schema-query pairs.

(2) Tutorial: If users prefer a more organized format for learning how to interpret the query diagrams, after the demonstrated example, a self-paced written tutorial will be available which provides descriptions of SQL queries and their corresponding query diagrams. This tutorial would also be beneficial for users curious about how newly supported operations are visualized in complex settings.

(3) Pattern Identification Test: Participants can explore the importance of relational patterns in query interpretation by participating in tasks similar to the experimental setup of the Relational Diagram user study [8]. In addition to serving as a way to demonstrate the effectiveness of the diagrams, these tasks also familiarize participants with how the diagram designs correspond to SQL logic. First, we define multiple many-to-many database schemas (such as the sailor-boat schema used in Example 1) to use in our test questions. Then, we generate multiple-choice questions where users must interpret a query to identify the correct option. Participants can choose to receive SQL only, PatternVis only, or both, to compare the difficulty between modalities.

Acknowledgments

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

References

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

[2] Jonathan Danaparamita and Wolfgang Gatterbauer. 2011. QueryViz: helping users understand SQL queries and their patterns. In *EDBT*. 558–561. doi:10.1145/1951365.1951440

[3] Sara Di Bartolomeo, Mirek Riedewald, Wolfgang Gatterbauer, and Cody Dunne. 2021. Stratisfimal Layout: A modular optimization model for laying out layered node-link network visualizations. *IEEE TVCG (VIS'21)* 28, 1 (2021), 324–334. doi:10.1109/TVCG.2021.3114756

[4] Richard M. Felder and Linda K. Silverman. 1988. Learning and teaching styles in engineering education. *Engr. Education* 78, 7 (1988), 674–681. https://www.academia.edu/download/31039406/LS-1988.pdf

[5] Sneha Gathani, Peter Lim, and Leilani Battle. 2020. Debugging Database Queries: A Survey of Tools, Techniques, and Users. In *CHI '20*. 1–16. doi:10.1145/3313831.3376485

[6] Wolfgang Gatterbauer. 2024. A Comprehensive Tutorial on over 100 Years of Diagrammatic Representations of Logical Statements and Relational Queries. In *ICDE*. IEEE, 5387–5392. doi:10.1109/ICDE60146.2024.00407 Tutorial page: https://northeastern-datalab.github.io/diagrammatic-representation-tutorial/.

[7] Wolfgang Gatterbauer. 2024. A Principled Solution to the Disjunction Problem of Diagrammatic Query Representations. doi:10.48550/arXiv.2412.08583 arXiv:2412.08583 [cs.DB]

[8] Wolfgang Gatterbauer and Cody Dunne. 2024. On The Reasonable Effectiveness of Relational Diagrams: Explaining Relational Query Patterns and the Pattern Expressiveness of Relational Languages. *PACMMOD* 2, 1, Article 61 (March 2024), 27 pages. doi:10.1145/3639316

[9] Wolfgang Gatterbauer and Cody Dunne. 2025. Relational Diagrams and the Pattern Expressiveness of Relational Languages. *SIGMOD Rec.* 54, 1 (2025), 80–88. https://sigmodrecord.org/?smd_process_download=1&download_id=14010

[10] 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

[11] Nodira Khoussainova, Magdalena Balazinska, Wolfgang Gatterbauer, YongChul Kwon, and Dan Suciu. 2009. A Case for A Collaborative Query Management System. In *CIDR*. https://doi.org/10.48550/arXiv.0909.1778

[12] 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

[13] Daphne Miedema and George Fletcher. 2021. SQLVis: Visual Query Representations for Supporting SQL Learners. In *VL/HCC*. IEEE, 1–9. doi:10.1109/VL/HCC51201.2021.9576431

[14] Marian Petre. 1995. Why looking isn't always seeing: readership skills and graphical programming. *CACM* 38, 6 (1995), 33–44. doi:10.1145/203241.203251

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

[16] P. Reisner. 1977. Use of Psychological Experimentation as an Aid to Development of a Query Language. *IEEE Transactions on Software Engineering* SE-3, 3 (1977), 218–229. doi:10.1109/TSE.1977.231131

[17] Edson Silva, Robson Fidalgo, Márcio Ferro, and Natália Franco. 2023. Visual query languages to design complex queries: a systematic literature review. *Software and Systems Modeling* 22, 4 (2023), 1217–1249. doi:10.1007/s10270-022-01071-4

[18] Lovisa Sundin and Quintin Cutts. 2019. Is it feasible to teach query programming in three different languages in a single session? A study on a pattern-oriented tutorial and cheat sheets. In *UKICER*. ACM, Article 7, 7 pages. doi:10.1145/3351287.3351293

[19] Taylor W Webb, Steven M Frankland, Awni Altabaa, Simon Segert, Kamesh Krishnamurthy, Declan Campbell, Jacob Russin, Tyler Giallanza, Randall O'Reilly, John Lafferty, et al. 2024. The relational bottleneck as an inductive bias for efficient abstraction. *Trends in Cognitive Sciences* 28, 9 (2024), 829–843. doi:10.1016/j.tics.2024.04.001

[20] Max Wertheimer. 1938. Gestalt theory. In *A source book of Gestalt psychology*, W. D. Ellis (Ed.). Kegan Paul, Trench, Trubner & Company, 1–11. doi:10.1037/11496-001

[21] Wikipedia contributors. 2025. Categorical Proposition — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Categorical_proposition [Online; accessed March-2025].