A Tutorial on Relational Language Design

Wolfgang Gatterbauer
Northeastern University
Boston, MA, USA

Abstract

Relational query languages have been studied and used for more than 50 years, with SQL dominant in practice. Yet that dominance is now being questioned from several directions at once: higher-level abstractions such as entity-relationship and functional data models, application languages that integrate querying with application logic, algebraic intermediate representations that blur the boundary between logical and physical specification, and large language models (LLMs) that both generate and explain queries. These developments highlight that *relational languages differ not only in expressive power*, but also in what relational structure they make explicit and in how effectively they support humans and machines in writing, understanding, revising, and reasoning about queries.

This tutorial uses this moment to give the data management community a framework for comparing and redesigning relational languages in an era of AI-assisted query generation, explanation, verification, and revision. Rather than beginning from formal definitions, we start from a fixed set of representative SQL queries and compare how classical alternatives and a range of recent languages express the same relational intent. From these examples, we derive a common vocabulary of recurring design dimensions and trade-offs in relational language design. This vocabulary distinguishes three parent aspects (*query intent*, *relational intent*, and *notation*), together with several subaspects (such as *relational pattern*, *semantic conventions*, and *modality*). Participants will leave with clearer mental models for comparing existing and proposed languages, evaluating their usability for humans and AI systems, and articulating open problems in the design and evaluation of relational query languages.

The tutorial webpage is at: https://northeastern-datalab.github.io/relational-language-tutorial.

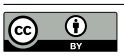
CCS Concepts

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

Keywords

Relational Query Language, Query Language Design, Query Comprehension, Language Usability, Relational Patterns

ACM Reference Format:
Wolfgang Gatterbauer. 2026. A Tutorial on Relational Language Design. In *Companion of the International Conference on Management of Data (SIGMOD Companion '26), May 31-June 05, 2026, Bengaluru, India.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3788853.3801875



SIGMOD Companion '26, Bengaluru, India

ACM ISBN 979-8-4007-2450-3/2026/05
https://doi.org/10.1145/3788853.3801875

1 Introduction

Relational query languages are usually studied in terms of expressive power, which answers what a language can say [14]. But *expressiveness* is only part of language design, *effectiveness* is another. A language also affects how easily we can say what we want to say, what structure becomes visible, which explanations are easy to follow, and how readily a query can be revised. In his 1979 Turing Award lecture "*Notation as a Tool of Thought*" [37] Iverson cites Whitehead via Cajori as "*By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems, and in effect increases (our) mental power...*" This perspective is less prominent in the database community than expressive power, but it is equally important for language design: a query language is not only a means of specifying computation for execution, but also a tool through which humans and machines understand, explain, and revise queries. More broadly, it connects to the idea that *language can shape the way we think and reason* [65].

This perspective is increasingly timely in practice. A number of recent efforts revisit the question of whether SQL should remain the default language for relational querying and propose either extensions or alternatives (including [1, 6, 50, 56]). These proposals differ not only in individual features, but also in how they present relational structure: for example, through nested or pipelined composition, named or positional access to relation components, and tuple or domain variables. Such choices influence what becomes explicit or implicit, which structural patterns are easy to recognize, and which rewritings are easy to apply.

At the same time, the rise of generative AI and large language models (LLMs) is reshaping how users interact with data. Relational query languages are no longer only for telling machines what to do; increasingly, they are for helping humans understand what machines have done or intend to do. As noted in [4, 29], this shift moves attention from human query composition toward machine query generation and human *query interpretation*. When LLM-generated queries can be wrong or misleading, the key design question is no longer only whether a language is easy to write, but whether it is easy to read and supports local revision. In that setting, a query is no longer merely a one-shot specification for execution, but a shared working representation (a draft) that humans and machines iteratively inspect, verify, and revise [29, Fig. 2].

What is still missing is a principled framework for saying *how different relational languages realize the same relational intent while making different structure, assumptions, and notation visible* to humans and machines. More specifically, we need a way to compare languages according to how they derive the same result from the same base relations while differing in notation, structure, and semantic conventions. Without a more precise vocabulary and framework, debates about language design oscillate between surface syntax and isolated examples.

1 **query intent:** *a question the user wants answered, prior to fixing a schema or query language.* (an information need)

2 **relational intent (semantics / denotation):** *a relation-valued mapping from admissible input instances to output relations.*

- 2a **relational pattern:** *a notation-independent abstract relational structure specifying how an output relation is derived from table references [27, 28], up to the semantic background.* (Without self-joins: "relational intent = relational pattern + semantic background")
- 2b **semantic background:** *the implicit assumptions and conventions not directly encoded in the query expression that determine which input instances are admissible and how outputs are computed.*
  - 2bi **semantic conventions:** *language-level semantic choices that affect how a query expression maps admissible inputs to outputs.* (e.g., set vs. bag semantics, null value behavior, the value of an empty sum $\sum_{\emptyset}$)
  - 2bii **schema assumptions:** *assumptions about admissible schemas.* (such as PK/FK constraints, nullability, and domain constraints)

3 **notation (syntax):** *the externally represented scheme of symbols, arrangement conventions, and formation rules by which query expressions are constructed and recognized.*

- 3a **modality**[1]**:** *the external representational form in which notation is realized.* (such as text or diagrams)
- 3b **structural notation:** *the modality-independent structural notational design choices that determine how a relational pattern is encoded in a representation.* (e.g., nested vs. pipelined composition, named vs. positional access to relation components, tuple vs. domain variables)
- 3c **surface notation:** *the modality-specific notational choices that affect only local spelling, syntax, naming, or layout (e.g., "R.A" vs. "R[A]" vs. "A(R)")*

query expression: *a concrete, well-formed representational object in a particular notation.* (A query expression encodes a relational pattern in a particular notation and, under a fixed semantic background, denotes a relational intent.)

query language: *a notation together with associated language-level semantic conventions, thereby yielding a family of well-formed query expressions.*

Figure 1: Preliminary Definitions in our relational language design framework.

Relational Language Design Framework. To systematize this discussion, we distinguish and disentangle several aspects that discussions of query languages often conflate. The three parent aspects are: 1 *query intent* (a question the user wants answered, prior to fixing a schema or query language), 2 *relational intent* (a relation-valued mapping from admissible input instances to output relations, i.e. the denotation or semantics of a query expression), and 3 *notation or syntax* (the externally represented scheme of symbols, arrangement conventions, and formation rules by which query expressions are constructed and recognized).

Relational intent decomposes into two orthogonal aspects: 2a *relational pattern* (a notation-independent abstract relational structure specifying how an output relation is derived from table references [27, 28], up to the semantic background), and 2b *semantic background* (the implicit assumptions and conventions not directly encoded in the query expression that determine which input instances are admissible and how outputs are computed). The semantic background decomposes into 2bi *semantic conventions* that are language-specific and 2bii *schema assumptions* that are not language-specific (see Fig. 1). A relational pattern provides the structural component of relational intent, but it does not determine the result on its own without semantic background. The semantic background determines whether two different relational patterns denote the same relational intent.

[1]Earlier in [30], modality was phrased as *an alternative isomorphic representational form of the same relational intent, possibly tailored to a particular audience.*

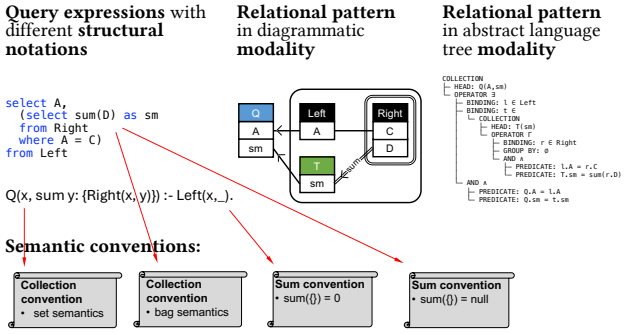


Figure 2: Two query expressions in SQL and Soufflé that encode the same relational pattern with different structural notation choices. Semantic conventions (such as bag semantics and empty-sum behavior) vary independently.

Notation (syntax) itself is determined by three aspects that are not perfectly orthogonal: 3a *modality* (the external representational form in which notation is realized, such as text or diagrams), and two types of notational design choices, 3b *structural notation* and 3c *surface notation*, which are either modality-independent or modality-dependent (see Fig. 1).

A *query expression* is a concrete, well-formed representational object in a particular notation. Given our various prior distinctions, a query expression can also be seen as a derived notion: A query expression encodes a relational pattern in a notation. A *query language* is a notation together with associated language-level semantic conventions, thereby yielding a family of well-formed query expressions. Figure 1 summarizes this terminology, and Fig. 2 illustrates it with a concrete example.

Keeping these aspects distinct is essential: without the notion of relational pattern, debates on language design tend to focus on logical expressiveness [27, 28]. And without clearly separating semantic conventions from syntax, comparisons may attribute to notational choices effects that are actually due to background context not directly encoded in a query expression.

This tutorial provides a systematic overview of relational language design, i.e. of query languages for the relational model. Rather than beginning from formal definitions, we start from a fixed set of representative SQL queries and compare how classical and recent relational languages realize the same relational intent with different relational patterns and structural notations. To make relational pattern visible, we use the diagrammatic modality of Abstract Relational Calculus (ARC) [30] as a common reference representation, which is itself an extension of Relational Diagrams [26, 28]. Our focus is on languages for flat input and output relations (first normal form). Our goal is not to advocate a replacement for SQL, but to

clarify the design space and develop a shared vocabulary for relational language design, encoding styles, and the evolving demands on languages that must support both human understanding and machine generation.

2 Goals and Scope

2.1 Why now is the right time to study relational language design

Although debates about SQL's limitations have been ongoing for nearly 50 years [13], and despite the cautionary lesson that 'what goes around comes around' [57], we believe that now is the right time to rethink relational language design, for five reasons:

(1) SQL has accumulated *decades of patches*, resulting in a language that often feels more irregular and harder than it needs to be. It has many clause variants (for example, IN vs. EXISTS), numerous syntactic quirks, and enough complexity to make queries unnecessarily hard to maintain and debug, from simple student queries to real workloads. SQL is also difficult to extend in a modular fashion with reusable and extensible libraries.

(2) Researchers are pushing toward higher-level abstractions such as entity-relationship (ER) data models [18] and pure functional treatments of data [20, 31]. They are also looking at more unified languages: for example, Rel re-envisions relational languages as covering entire application programs [6]. At the same time, major systems groups are again willing to revise the surface language: Google's pipe syntax for SQL [56] and similar ideas such as PRQL [1] and SaneQL [50] aim to make queries more modular and compositional.

(3) LLMs are shifting the main burden for humans from writing SQL to understanding and verifying SQL. Models can generate complex queries, but humans (and tools) must still interpret them, and SQL's many quirks make that harder than necessary. Prompts such as 'Please explain what this query really does' have become increasingly common. At the same time, SQL's syntax and semantics are poorly aligned [34], which can also make it harder for LLMs to reason about SQL. A language that is slightly more verbose but more regular might actually make generation, explanation, and editing easier.

(4) At the lower end of the stack, new intermediate representations (IRs) are also being proposed to enable algebraic optimizations across different domains (e.g., across both relational algebra and linear algebra). Examples include the semiring dictionary language SDQL [54] and Substrait [3]. Even the traditional boundary between logical and physical layers is becoming blurry.

(5) We now have enough theory, engineering experience, and AI-assisted tools to build prototype language transducers much faster than before. This makes it possible not only to rethink language design, but also to 'just do it', informed by lessons learned from 50 years of real production systems, and by new interaction patterns (such as AI-assisted querying). Together, these developments make this an especially good moment to analyze existing relational query languages and explore new alternatives.

2.2 Learning outcomes, audience, and scope

Attendees will leave with a sharper vocabulary for discussing relational query languages and more informed mental models of the design trade-offs between them. They will learn to recognize recurring relational patterns across different notations. Importantly, these patterns exist independently of any particular notation. Rather than reproducing examples chosen by recent language proposals, the tutorial starts from a common set of representative queries and examines how different languages express this shared workload.[2] Along the way, it introduces well-established concepts in relational language design (such as domain vs. tuple perspective, and named vs. positional access to relation components) and more recent terminology from the author's work (such as 'inside-out' vs. 'outside-in'). These concepts are illustrated first through running examples and only then formalized. Thus instead of starting from abstract concepts, we develop them by looking at many examples. By the end of the tutorial, attendees should be able to look at an unfamiliar relational language, separate relational pattern from semantic context and notation, and explain the main design trade-offs that follow.

Audience and prerequisites. This 90-minute tutorial targets researchers and practitioners seeking an intuitive yet comprehensive survey of relational language design. It focuses on the commonalities and differences across various languages. The tutorial is easiest to follow for attendees familiar with relational algebra (RA), relational calculus (RC), and SQL, but it remains accessible to attendees with only a background in SQL.

Distribution. Slides will be made available afterward on the tutorial webpage,[3] as with other recent tutorials by the presenter and collaborators.[4]

Scope of this tutorial. The tutorial focuses on relational languages whose inputs and outputs are flat relations (first normal form, 1NF). These include SQL, relational calculus (RC), relational algebra (RA), and Datalog-style languages with recursion. Depending on time, we may also touch on dataframe-style languages (such as pandas and dplyr), as well as point-free languages (such as APL and J). We may also briefly touch upon more expressive formalisms that combine recursion with negation or disjunction, such as disjunctive logic programming.

Out-of-scope. Graph query languages, languages for nested relations, and languages for time-series or streaming data are outside the tutorial's main scope. We mention them only briefly in the concluding outlook, mainly to position the flat relational setting within the broader landscape. Graph query languages have already been the subject of several recent tutorials [7, 19, 42, 46] and surveys [5].

Related tutorials. To the best of our knowledge, this is the first database-conference tutorial focused on relational language design since Tannen's PODS 1994 tutorial on query languages over collection types [58] and Kanellakis' PODS 1995 tutorial on constraint programming and database languages [40]. This stands in sharp contrast to the large number of recent tutorials on graph query languages.

[2] For a 515-slide example-driven tutorial in a similar spirit, see [24]: https://northeastern-datalab.github.io/diagrammatic-representation-tutorial/ICDE_2024-Diagrammatic-Representations-Tutorial.pdf
[3] https://northeastern-datalab.github.io/relational-language-tutorial
[4] https://northeastern-datalab.github.io/diagrammatic-representation-tutorial/, https://northeastern-datalab.github.io/visual-query-representation-tutorial/, https://northeastern-datalab.github.io/topk-join-tutorial/, https://northeastern-datalab.github.io/responsive-dbms-tutorial/

3 Tutorial Content and Outline

The tutorial is organized around a small set of representative SQL queries that are translated into multiple relational languages and shown side by side.Rather than defining terminology up front, we introduce it through comparisons of concrete query expressions (e.g., contrasting domain and tuple relational calculus) and only then distill the recurring structural notation choices. This example-first organization matches the overall goal of the tutorial: to separate query intent, relational pattern, structural notation, and semantic conventions, and to give attendees a vocabulary for recognizing the same relational structure across notationally different languages.

Core concepts. Using ARC as a common reference representation and Relational Diagrams as an auxiliary visual aid, we introduce the main concepts that recur across relational languages. These include tuple vs. domain variables, named vs. positional access to relation components, set vs. bag semantics (and list semantics, where relevant), nested vs. pipelined composition, pointwise (variable-based) vs. point-free (tacit) notation [6, 8, 9], declarative vs. procedural style [63], null handling and binary vs. ternary logic [44], explicit joins via equality predicates, implicit joins via shared variables, joins via path expressions [22, 67], and grouping from the inside out (FIO) vs. from the outside in (FOI) [30]. We relate these choices to different user tasks: composing queries and reading queries [52], and revising existing queries. We also borrow concepts from programming-language design, especially orthogonality [9, 39].

Languages. We begin with the classical core languages SQL [34], tuple relational calculus (TRC), domain relational calculus (DRC), relational algebra (RA), and Datalog [12]/Soufflé [2, 53], to establish the main concepts. We then examine comprehension- and functional-style languages [31, 32, 51] (such as DAPLEX [55], Kleisli [66], Links [15, 16], and LINQ [48, 49]), with emphasis on how they expose or hide relational structure. We also discuss Rel [6] as a recent relational programming language that revisits the boundary between query language and host language, and extends relational querying toward programming in the large. We then take a deeper look at formalisms for aggregation, starting from Klug's formalism for aggregate queries under set semantics [41], which influenced subsequent comprehension-based models for database programming languages (DBPLs) [10, 21, 33, 59], including extensions of logic with aggregate operators [36], and, if time permits, brief hands-on code in Haskell [38, 62]. Finally, we discuss recent compositional proposals such as Google's pipe syntax [56], SaneQL [50], and PRQL [1], asking how pipelining affects modularity and ease of reading queries Depending on time, we briefly connect these ideas to dataframe-style systems such as pandas [47] and dplyr [64], and to point-free languages such as APL and J.

Outlook. We conclude by sketching how the framework extends beyond flat relational tables. In particular, we briefly discuss the challenges in moving beyond flat relations to the nested relational model (such as SQL++ [11] and JSON Relational Duality [35]), and to graph query languages. We may also outline how extensions such as unstratified negation and disjunctive rule heads substantially change expressive power [17] and can even allow automating NP hardness reductions [45]. We close with open problems in relational language design.

4 Open research issues

The tutorial discusses notational choices in terms of what they encode and how they encode it. An important issue not addressed here, but central in practice, is the motivating question from the introduction: *how language shapes the way we think and reason.* In our setting, this means how notation shapes what users notice, what they can mentally simulate, and where misunderstandings are likely to arise.

The tutorial does not attempt to answer these empirical questions. Rather, it provides the vocabulary and conceptual framework needed to formulate them precisely, so that future evaluations can compare notational choices in a principled way. At the core of these questions is which aspects of relational patterns a notation makes visible to the reader and which it leaves implicit.

The usability of a language, whether for humans or machines, and for particular applications, must ultimately be established empirically, ideally through preregistered, adequately powered, and reproducible user studies. The database literature contains some such targeted studies (e.g., [27, 43]), but we still lack studies of foundational structural notation design questions, such as whether users understand queries better when they are expressed with domain variables, tuple variables, or in point-free notation.

We also still lack a principled understanding of how suitable existing languages are for the increasingly large machine audience, motivating what one might call "machine user studies" (e.g., whether future agents will develop their own structured agent languages). At the same time, we may want to rethink the boundary between user-facing languages and internal representations for query optimization, and perhaps even the traditional separation between logical and physical optimization (there is room for a new categorization). More broadly, we need extensive comparative studies of the wider family of query languages, including graph query languages and languages for nested relational models.

This tutorial aims to motivate and structure such work by surfacing the trade-offs made by existing languages and proposing a common vocabulary to support future empirical and theoretical studies.

5 Author information

Wolfgang Gatterbauer is an Associate Professor at the Khoury College of Computer Sciences at Northeastern University. His research interests lie at the intersection of theory and practice of data management. He received an NSF CAREER award and, with his students and collaborators, a SIGMOD research highlight award, the best paper award at EDBT 2021, best-of-conference mentions for PODS 2021, SIGMOD 2017, WALCOM 2017, and VLDB 2015, and two reproducibility awards for papers from SIGMOD 2020. He has given tutorials on visual query representations at VLDB'23 [23] and ICDE'24 [24] and on query optimization at SIGMOD'20 [60] and ICDE'22 [61]. All slides from those materials are available online on the tutorial pages, including recorded videos for one of them.

The content of this tutorial is heavily informed by his Spring 2026 class on relational language design [25], by his work on visual representations of relational queries [26–29], and by the recent proposal of an abstract relational query language that can embed other relational languages and represent their patterns [30].

6 Acknowledgments

Thanks to Diandre Sabale for helpful comments on an earlier draft, and to Molham Aref, Leonidas Fegaras, and Val Tannen for taking the time to discuss various aspects of relational language design. I acknowledge the use of AI-based language tools to assist with drafting and revising the exposition. All technical content and claims remain my responsibility.

References

[1] 2025. PRQL (Pipelined Relational Query Language). https://prql-lang.org/
[2] 2025. Soufflé. Aggregates and Generative Functors. https://souffle-lang.github.io/aggregates.
[3] 2025. Substrait: Cross-Language Serialization for Relational Algebra. https://substrait.io/
[4] Anastasia Ailamaki, Samuel Madden, Daniel Abadi, Gustavo Alonso, Sihem Amer-Yahia, Magdalena Balazinska, Philip A. Bernstein, Peter A. Boncz, Michael J. Cafarella, Surajit Chaudhuri, Susan B. Davidson, David J. DeWitt, Yanlei Diao, Xin Luna Dong, Michael J. Franklin, Juliana Freire, Johannes Gehrke, Alon Y. Halevy, Joseph M. Hellerstein, Mark D. Hill, Stratos Idreos, Yannis E. Ioannidis, Christoph Koch, Donald Kossmann, Tim Kraska, Arun Kumar, Guoliang Li, Volker Markl, Renée J. Miller, C. Mohan, Thomas Neumann, Beng Chin Ooi, Fatma Ozcan, Aditya G. Parameswaran, Ippokratis Pandis, Jignesh M. Patel, Andrew Pavlo, Danica Porobic, Viktor Sanca, Michael Stonebraker, Julia Stoyanovich, Dan Suciu, Wang-Chiew Tan, Shivaram Venkataraman, Matei Zaharia, and Stanley B. Zdonik. 2025. The Cambridge Report on Database Research. *CoRR* abs/2504.11259 (2025). doi:10.48550/ARXIV.2504.11259
[5] Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan L. Reutter, and Domagoj Vrgoc. 2017. Foundations of Modern Query Languages for Graph Databases. *ACM Comput. Surv.* 50, 5 (2017), 68:1–68:40. doi:10.1145/3104031
[6] 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
[7] Marcelo Arenas, Claudio Gutierrez, and Juan F. Sequeda. 2021. Querying in the Age of Graph Databases and Knowledge Graphs. In *SIGMOD*. 2821–2828. doi:10.1145/3448016.3457545
[8] John W. Backus. 1978. Can Programming Be Liberated From the von Neumann Style? A Functional Style and its Algebra of Programs. *CACM* 21, 8 (1978), 613–641. doi:10.1145/359576.359579
[9] Raymond T. Boute. 2005. Functional declarative language design and predicate calculus: a practical approach. *ACM Trans. Program. Lang. Syst.* 27, 5 (2005), 988–1047. doi:10.1145/1086642.1086647
[10] Peter Buneman, Leonid Libkin, Dan Suciu, Val Tannen, and Limsoon Wong. 1994. Comprehension Syntax. *SIGMOD Rec.* 23, 1 (1994), 87–96. doi:10.1145/181550.181564
[11] Michael J. Carey, Don Chamberlin, Almann Goo, Kian Win Ong, Yannis Papakonstantinou, Chris Suver, Sitaram Vemulapalli, and Till Westmann. 2024. SQL++: We Can Finally Relax!. In *ICDE*. 5501–5510. doi:10.1109/ICDE60146.2024.00438
[12] Stefano Ceri, Georg Gottlob, and Letizia Tanca. 1989. What you Always Wanted to Know About Datalog (And Never Dared to Ask). *IEEE Trans. Knowl. Data Eng.* 1, 1 (1989), 146–166. doi:10.1109/69.43410
[13] Donald Chamberlin. 2024. 50 Years of Queries. *CACM* 67, 8 (Aug. 2024), 110–121. doi:10.1145/3649887
[14] E. F. Codd. 1972. Relational Completeness of Data Base Sublanguages. *Research Report / RJ / IBM / San Jose, California* RJ987 (1972). https://citeseerx.ist.psu.edu/document?doi=6a048dc38250ffce49c5e6a5040b4c91ca05e83d
[15] Ezra Cooper. 2009. The Script-Writer's Dream: How to Write Great SQL in Your Own Language, and Be Sure It Will Succeed. In *Proceedings of the 12th International Symposium on Database Programming Languages (DBPL '09)*. 36–51. doi:10.1007/978-3-642-03793-1_3
[16] Ezra Cooper, Sam Lindley, Philip Wadler, and Jeremy Yallop. 2006. Links: Web Programming Without Tiers. In *5th International Symposium on Formal Methods for Components and Objects (FMCO 2006)*. 266–296. doi:10.1007/978-3-540-74792-5_12
[17] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. 2001. Complexity and expressive power of logic programming. *ACM Comput. Surv.* 33, 3 (2001), 374–425. doi:10.1145/502807.502810
[18] Amol Deshpande. 2025. Beyond Relations: A Case for Elevating to the Entity-Relationship Abstraction. In *CIDR*. https://vldb.org/cidrdb/papers/2025/p15-deshpande.pdf
[19] Alin Deutsch and Yannis Papakonstantinou. 2018. Graph data models, query languages and programming paradigms. *PVLDB* 11, 12 (Aug. 2018), 2106–2109. doi:10.14778/3229863.3229879
[20] Jens Dittrich. 2026. A Functional Data Model and Query Language is All You Need. In *EDBT*. 619–626. doi:10.48786/EDBT.2026.50
[21] Leonidas Fegaras and David Maier. 2000. Optimizing object queries using an effective calculus. *ACM TODS* 25, 4 (2000), 457–516. doi:10.1145/377674.377676
[22] Jürgen Frohn, Georg Lausen, and Heinz Uphoff. 1994. Access to Objects by Path Expressions and Rules. In *VLDB*. 273–284. http://www.vldb.org/conf/1994/P273.PDF
[23] 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.
[24] 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.
[25] Wolfgang Gatterbauer. 2026. CS 7575: A Seminar On Relational Language Design (Spring 2026). https://northeastern-datalab.github.io/cs7575/sp26/
[26] Wolfgang Gatterbauer. 2026. A Principled Solution to the Disjunction Problem of Diagrammatic Query Representations. *PACMMOD* 4, 1 (2026), 3:1–3:28. doi:10.1145/3786617 Full version: https://arxiv.org/pdf/2412.08583.
[27] 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 (2024). doi:10.1145/3639316 Full version: https://arxiv.org/pdf/2401.04758.
[28] Wolfgang Gatterbauer and Cody Dunne. 2025. Relational Diagrams and the Pattern Expressiveness of Relational Languages. *SIGMOD Record* 54, 1 (2025), 80–88. https://sigmodrecord.org/?smd_process_download=1&download_id=14010
[29] 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
[30] Wolfgang Gatterbauer and Diandre Miguel Sabale. 2026. Database Research Needs an Abstract Relational Query Language. In *CIDR*. https://vldb.org/cidrdb/papers/2026/p27-gatterbauer.pdf Project webpage: https://relationaldiagrams.com/.
[31] Peter M. D. Gray. 2009. Functional Query Language. In *Encyclopedia of Database Systems*. Springer US, 1201–1204. doi:10.1007/978-0-387-39940-9_1092
[32] Peter M. D. Gray, Peter J. H. King, and Alexandra Poulovassilis. 2004. *Introduction to the Use of Functions in the Management of Data*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1–54. doi:10.1007/978-3-662-05372-0_1
[33] Torsten Grust and Marc H. Scholl. 1999. How to Comprehend Queries Functionally. *J. Intell. Inf. Syst.* 12, 2-3 (1999), 191–218. doi:10.1023/A:1008705026446
[34] Paolo Guagliardo and Leonid Libkin. 2017. A Formal Semantics of SQL Queries, Its Validation, and Applications. *PVLDB* 11, 1 (2017), 27–39. doi:10.14778/3151113.3151116
[35] Shashank Gugnani, Zhen Hua Liu, Hui Chang, Beda Hammerschmidt, Srinivas Kareenhalli, Kishy Kumar, Tirthankar Lahiri, Ying Lu, Douglas McMahon, Ajit Mylavarapu, Sukhada Pendse, and Ananth Raghavan. 2025. JSON Relational Duality: A Revolutionary Combination of Document, Object, and Relational Models. In *Companion of SIGMOD/PODS 2025*. 431–443. doi:10.1145/3722212.3724441
[36] Lauri Hella, Leonid Libkin, Juha Nurmonen, and Limsoon Wong. 2001. Logics with aggregate operators. *JACM* 48, 4 (2001), 880–907. doi:10.1145/502090.502100
[37] Kenneth E. Iverson. 1980. Notation as a tool of thought. *CACM* 23, 8 (Aug. 1980), 444–465. doi:10.1145/358896.358899
[38] Simon L. Peyton Jones and Philip Wadler. 2007. Comprehensive comprehensions. In *Proceedings of the ACM SIGPLAN Workshop on Haskell, Haskell 2007, Freiburg, Germany, September 30, 2007*, Gabriele Keller (Ed.). ACM, 61–72. doi:10.1145/1291201.1291209
[39] Guy L. Steele Jr. 1999. Growing a Language. *High. Order Symb. Comput.* 12, 3 (1999), 221–236. doi:10.1023/A:1010085415024
[40] Paris Kanellakis. 1995. Constraint programming and database languages: a tutorial. In *PODS*. 46–53. doi:10.1145/212433.212447
[41] Anthony C. Klug. 1982. Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions. *JACM* 29, 3 (1982), 699–717. doi:10.1145/322326.322332
[42] Haridimos Kondylakis, Stefania Dumbrava, Matteo Lissandrini, Nikolay Yakovets, Angela Bonifati, Vasilis Efthymiou, George Fletcher, Dimitris Plexousakis, Riccardo Tommasini, Georgia Troullinou, and Elisjana Ymeralli. 2025. Property Graph Standards: State of the Art & Open Challenges. *PVLDB* 18, 12 (2025), 5477–5481. doi:10.14778/3750601.3750698
[43] 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
[44] Leonid Libkin and Liat Peterfreund. 2023. SQL Nulls and Two-Valued Logic. In *PODS*. 11–20. doi:10.1145/3584372.3588661

[45] Neha Makhija and Wolfgang Gatterbauer. 2023. A Unified Approach for Resilience and Causal Responsibility with Integer Linear Programming (ILP) and LP Relaxations. *PACMMOD* 1, 4 (2023), 228:1–228:27. doi:10.1145/3626715

[46] Ioana Manolescu and Madhulika Mohanty. 2023. Full-Power Graph Querying: State of the Art and Challenges. *PVLDB* 16, 12 (Aug. 2023), 3886–3889. doi:10.14778/3611540.3611577

[47] Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman (Eds.). 56–61. doi:10.25080/Majora-92bf1922-00a

[48] Erik Meijer. 2011. The world according to LINQ. *CACM* 54, 10 (2011), 45–51. doi:10.1145/2001269.2001285

[49] Erik Meijer, Brian Beckman, and Gavin Bierman. 2006. LINQ: reconciling object, relations and XML in the .NET framework. In *SIGMOD*. 706. doi:10.1145/1142473.1142552

[50] Thomas Neumann and Viktor Leis. 2024. A Critique of Modern SQL and a Proposal Towards a Simple and Expressive Query Language. In *CIDR*. https://www.cidrdb.org/cidr2024/papers/p48-neumann.pdf

[51] Norman W. Paton and Peter M. D. Gray. 1990. Optimising and Executing DAPLEX Queries Using Prolog. *Comput. J.* 33, 6 (1990), 547–555. doi:10.1093/COMJNL/33.6.547

[52] Phyllis Reisner. 1981. Human Factors Studies of Database Query Languages: A Survey and Assessment. *ACM Comput. Surv.* 13, 1 (1981), 13–31. doi:10.1145/356835.356837

[53] Bernhard Scholz, Herbert Jordan, Pavle Subotic, and Till Westmann. 2016. On fast large-scale program analysis in Datalog. In *Proceedings of the 25th International Conference on Compiler Construction (CC)*. ACM, 196–206. doi:10.1145/2892208.2892226

[54] Amir Shaikhha, Mathieu Huot, Jaclyn Smith, and Dan Olteanu. 2022. Functional collection programming with semi-ring dictionaries. *PACMPL (OOPSLA1)* 6 (2022), 1–33. doi:10.1145/3527333

[55] David W. Shipman. 1981. The functional data model and the data languages DAPLEX. *ACM TODS* 6, 1 (March 1981), 140–173. doi:10.1145/319540.319561

[56] Jeff Shute, Shannon Bales, Matthew Brown, Jean-Daniel Browne, Brandon Dolphin, Romit Kudtarkar, Andrey Litvinov, Jingchi Ma, John D. Morcos, Michael Shen, David Wilhite, Xi Wu, and Lulan Yu. 2024. SQL has problems. We can fix them: Pipe syntax in SQL. *PVLDB* 17, 12 (2024), 4051–4063. doi:10.14778/3685800.3685826

[57] Michael Stonebraker and Andrew Pavlo. 2024. What Goes Around Comes Around... And Around... *SIGMOD Rec.* 53, 2 (July 2024), 21–37. doi:10.1145/3685980.3685984

[58] Val Tannen. 1994. Tutorial: languages for collection types. In *PODS*. 150–154. doi:10.1145/182591.182608

[59] Philip W. Trinder. 1991. Comprehensions, a Query Notation for DBPLs. In *3rd international workshop on Database Programming Languages (DBPL)*. 55–68. https://dl.acm.org/doi/10.5555/135260.135271

[60] Nikolaos Tziavelis, Wolfgang Gatterbauer, and Mirek Riedewald. 2020. Optimal Join Algorithms Meet Top-k. In *SIGMOD tutorials*. ACM, 2659–2665. doi:10.1145/3318464.3383132 Tutorial page: https://northeastern-datalab.github.io/topk-join-tutorial/.

[61] Nikolaos Tziavelis, Wolfgang Gatterbauer, and Mirek Riedewald. 2022. Toward Responsive DBMS: Optimal Join Algorithms, Enumeration, Factorization, Ranking, and Dynamic Programming. In *ICDE tutorials*. IEEE, 3205–3208. doi:10.1109/ICDE53745.2022.00299 Tutorial page: https://northeastern-datalab.github.io/responsive-dbms-tutorial/.

[62] Philip Wadler. 1990. Comprehending Monads. In *Proceedings of the 1990 ACM Conference on LISP and Functional Programming, LFP 1990, Nice, France, 27-29 June 1990*, Gilles Kahn (Ed.). ACM, 61–78. doi:10.1145/91556.91592

[63] Charles Welty and David W. Stemple. 1981. Human Factors Comparison of a Procedural and a Nonprocedural Query Language. *ACM TODS* 6, 4 (1981), 626–649. doi:10.1145/319628.319656

[64] Hadley Wickham, Romain François, Lionel Henry, Kirill Müller, and Davis Vaughan. 2026. *dplyr: A Grammar of Data Manipulation.* https://dplyr.tidyverse.org R package version 1.2.0.

[65] Wikipedia contributors. 2026. Linguistic relativity — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Linguistic_relativity [Online; accessed March-2026].

[66] Limsoon Wong. 2000. Kleisli, a functional query system. *Journal of Functional Programming* 10, 1 (2000), 19–56. doi:10.1017/S0956796899003585

[67] Carlo Zaniolo. 1983. The database language GEM. *SIGMOD Rec.* 13, 4 (May 1983), 207–218. doi:10.1145/971695.582226