Technical Perspective: Rel

Viktor Leis, Thomas Neumann
TUM
{leis|neumann}@in.tum.de

For more than half a century, relational data management has lived in a "two-language world": a powerful declarative sublanguage for data (SQL) glued to a host programming language for everything else. This split into two worlds creates friction: First, there is an "impedance mismatch" between the set oriented query language SQL and the usually more imperative host programming language. And second, while programming languages aim for producing reusable components, SQL queries are often ad-hoc, and there is little if any reuse of logic and functionality across queries.

The Rel language makes a clean and ambitious break with that paradigm. Rather than treating relational querying as a bolt-on, Rel is designed as a full programming language whose central abstraction is the relation itself. The paper goes beyond syntactic polish or yet another query notation, targeting both weaknesses of the traditional split: First, complex logic can be expressed naturally on relational data, eliminating the impedance mismatch between host language and query language. And second, the language allows for higher order constructs, which is the basis for implementing libraries of reusable components.

This shift directly targets long-recognized pain points of SQL. As we discuss in our CIDR'24 critique of SQL [2], it lacks extensibility and abstraction, forcing new capabilities to arrive as ever more surface syntax instead of reusable language-level constructs. Rel responds by elevating relational programming beyond "querying": it begins from Datalog and first-order logic, embraces recursion, and introduces abstraction and application as key constructs, enabling users to succinctly express computations that routinely escape the traditional boundaries of SQL. The result is not SQL plus recursion, but a language in which relations play the role that functions play in functional languages or procedures in imperative ones—a principled reframing that opens the door to richer, compositional program structure.

A particularly elegant idea behind Rel is its "growing a language" philosophy: the paper explicitly prioritizes mechanisms that let users build the language outward through libraries, rather than centralizing power in an ever-expanding standard. Rel's authors emphasize programming-in-the-large features precisely because large-scale development depends on modularity, parameterization, and reuse—not on an endless accumulation of special cases.

In the paper's organization, the progression from Rel's core to "Enabling Programming in the Large" and then to "Growing the Language" underscores a design that treats libraries as first-class citizens. Concretely, Rel supports passing richer parameters than individual scalar values (tuple and relation variables) and provides constructs that let developers package reusable relational building blocks that remain uniform with the rest of the language. This is exactly the kind of extensibility SQL struggles to offer.

Equally important for a language proposal is that Rel comes with formal semantics. The paper includes an addendum giving a semantics for a core of Rel, spelling out the data model, environments, and the semantics of expressions, formulas, and programs. A semantics-first foundation enables rigorous reasoning about program meaning, optimization opportunities, and equivalence. In that sense, Rel aligns with recent calls for semantics-first relational representations (e.g., ARQL/ARC [1]) that separate intent from surface syntax and make relational structure explicit.

In sum, Rel is based on a bold idea: that relational thinking can be the primary programming model, not a specialized fragment embedded in something else. By (1) treating Rel as a programming language capable of expressing rich computations beyond traditional SQL querying, (2) enabling a library ecosystem that lets users grow the language rather than wait for standards to ossify, and (3) grounding the whole design in formal semantics, this paper charts a compelling path toward simpler architectures, more reusable data logic, and more principled relational systems.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 2024 ACM 0001-0782/24/0X00 ...$5.00.

1. REFERENCES

[1] W. Gatterbauer and D. M. Sabale. Database research needs an abstract relational query language. In *CIDR*, 2026.

[2] T. Neumann and V. Leis. A critique of modern SQL and a proposal towards a simple and expressive query language. In *CIDR*, 2024.