Technical Perspective: PRISM and the Case for Strategic Opacity in UDF Optimization

Alkis Simitsis
Athena Research Center
Athens, Greece
alkis@athenarc.gr

The history of UDF optimization in relational databases reads as a story of increasing transparency [2]. Compilation makes UDFs faster but leaves them opaque to the optimizer. Batching and fusion amortize invocation costs but still treat UDFs as black boxes. Inlining, most notably demonstrated by Microsoft's FROID system [3], aspires to the ultimate transparency by converting UDFs entirely into SQL, hence letting the optimizer reason about their contents. The trajectory seems inevitable: the path to better performance runs through making everything visible.

The core idea. PRISM, proposed by Arch et al. [1], upends this narrative. It shows that selectively restoring opacity, i.e., deliberately hiding parts of a UDF from the optimizer, yields better performance than full transparency. This is a counterintuitive result. Optimizers are designed to improve plans when given more information and yet PRISM shows that feeding them the full inlined translation of a complex UDF could harm performance. The LATERAL joins and deeply nested subqueries often produced by complete inlining overwhelm unnesting algorithms, prevent predicate pushdown, and generate bloated plans with unnecessary joins. PRISM reframes UDF optimization as a partitioning problem: rather than inlining an entire UDF into SQL, the framework decomposes each function, selectively inlines only the relational fragments helpful to the optimizer, and compiles the remaining procedural code into opaque native functions via *UDF outlining*, a technique borrowed in spirit from compiler engineering, where code is deliberately extracted into separate functions rather than inlined. This is a principled reversal of the prevailing wisdom: instead of asking "how do we inline everything?", PRISM asks "what is the minimum we need to inline?". The result is structurally simpler queries that are easier to unnest, enable data skipping, and avoid unnecessary joins.

The technique. The technical foundation of PRISM rests on a compiler-inspired decomposition of UDFs. The framework parses each function into a static single assignment (SSA) based intermediate representation (IR) organized as a tree of single-entry single-exit (SESE) regions: leaf, conditional, loop, and sequential. This structure is the linchpin of the approach. As every region has exactly one entry and one exit, any region can be extracted into its own function without restructuring the surrounding code. PRISM exploits this property through region-based outlining: it identifies the largest procedural regions that contain no embedded SELECT statements and extracts them into compiled, opaque functions. What remains for inlining is often just the relational fragments the optimizer can actually exploit. PRISM applies five complementary optimizations in sequence.

Query motion hoists embedded SELECT statements out of procedural control flow, separating relational code from procedural code so each can be optimized independently.

Region-based UDF outlining then identifies the largest procedural code SESE regions and extracts them into separate functions that are compiled to machine code, hiding them from the optimizer.

Instruction elimination leverages SSA's single-assignment property to replace each variable use with its definition, progressively collapsing the UDF (frequently, to a single RETURN instruction) and eliminating all LATERAL joins.

Subquery elision bypasses inlining altogether by substituting the UDF's return expression directly into the calling query, sidestepping the subquery wrapper that inlining would normally introduce.

Predicate hoisting injects UDF-derived filtering conditions into the query's WHERE clause, hence unlocking data-skipping optimizations such as index scans and zone maps.

Together, these optimizations produce queries structurally simpler than what any single step could achieve alone. The power lies in their sequencing: query motion enables outlining, outlining enables instruction elimination, elimination enables elision, and hoisting addresses the orthogonal problem of predicate visibility.

The future. As SQL is increasingly extended to support emerging workloads such as LLM inference and semantic operators, effective UDF optimization becomes ever more critical. Yet many challenges remain. PRISM's inline-versus-outline decisions are currently driven by structural heuristics (e.g., whether a region contains embedded queries) rather than by a cost model (e.g., [4]) that estimates downstream plan quality for alternative partitions. Developing such a model could capture cases where the structural rule is suboptimal, though at the expense of increased analysis complexity. Also, in engines lacking a mechanism to mark compiled UDFs as safe for parallel execution, outlined functions force single-threaded execution, partially eroding PRISM's multi-threaded gains. PRISM also inherits a key limitation: loops with embedded queries still require full inlining, precluding finer-grained decomposition. More broadly, extending PRISM's SSA-based analysis to dynamically typed UDF languages (e.g., Python, JavaScript), where type information is resolved at runtime and control flow is less structured, remains an open challenge. Similarly, understanding how the inline/outline partitioning interacts with unified compilation frameworks (e.g., MLIR, GraalVM) could yield even finer-grained decomposition strategies.

References

[1] Samuel Arch, Yuchen Liu, Todd C. Mowry, Jignesh M. Patel, and Andrew Pavlo. 2024. The Key to Effective UDF Optimization: Before Inlining, First Perform Outlining. *PVLDB* 18, 1 (2024), 1–13.
[2] Yannis Foufoulas and Alkis Simitsis. 2023. Efficient Execution of User-Defined Functions in SQL Queries. *PVLDB* 16, 12 (2023), 3874–3877.
[3] Karthik Ramachandra, Kwanghyun Park, K. Venkatesh Emani, Alan Halverson, César A. Galindo-Legaria, and Conor Cunningham. 2017. Froid: Optimization of Imperative Programs in a Relational Database. *PVLDB* 11, 4 (2017), 432–444.
[4] Johannes Wehrstein, Tiemo Bang, Roman Heinrich, and Carsten Binnig. 2025. GRACEFUL: A Learned Cost Estimator for UDFs. In *ICDE*. IEEE, 2450–2463.