Hard to Read, Hard to Maintain? Refactoring Complex SQL into Wvlet Pipelines

Kaina Anderson
The University of Osaka
Suita, Osaka, Japan
anderson.kaina@ist.osaka-u.ac.jp

Yohanes Yohanie Fridelin Panduman
The University of Osaka
Suita, Osaka, Japan
yohanes@ist.osaka-u.ac.jp

Makoto Onizuka
The University of Osaka
Suita, Osaka, Japan
onizuka@ist.osaka-u.ac.jp

Taro L. Saito
Treasure AI
Mountain View, CA, USA
leo@treasure.ai

ABSTRACT

Modern analytical SQL queries are often hard to read and maintain because of duplicated logic, deep nesting, and the gap between syntax order and logical evaluation flow. We present *Wvlet Cognitive Workbench* (WCW), a browser-based system designed to enhance both query readability and maintainability. WCW transforms SQL queries into Wvlet queries, improves query structure through duplicate extraction and safe join flattening, and supports side-by-side comparison with readability metrics and semantic equivalence checking. The demo also supports workload-level refactoring by extracting shared reusable models across multiple queries.

PVLDB Reference Format:
Kaina Anderson, Yohanes Yohanie Fridelin Panduman, Makoto Onizuka, and Taro L. Saito. Hard to Read, Hard to Maintain? Refactoring Complex SQL into Wvlet Pipelines. PVLDB, 19(12): 4858 - 4861, 2026.
doi:10.14778/3827998.3828140

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/OnizukaLab/wcw-demo.

1 INTRODUCTION

SQL has been the standard language for database querying for more than fifty years. Its original design goal was to provide a *walk-up-and-read* style syntax that non-experts could understand intuitively[2]. However, query workloads in modern data analytics have become much more complex, and the readability of SQL has become a serious concern for maintainability and long-term operation [5, 7]. Since software maintenance accounts for a large portion of total development cost, and understanding existing code is one of the most time-consuming activities in maintenance [1], improving the readability of query languages is an important systems problem rather than a purely syntactic one.

We focus on two major issues that reduce the readability of SQL queries. First, there is a gap between SQL's syntax order and its logical evaluation flow [4, 5, 7]. Although SQL is written in a fixed syntax order such as `SELECT`, `FROM`, and `WHERE`, it is usually understood by following the underlying dataflow starting from `FROM` clause. Figure 1 illustrates this mismatch between written syntax and natural reading order. In complex queries, the mismatch often leads to deeply nested subqueries and forces readers to trace the logic in an unnatural inside-out manner. Since human working memory is limited [3], such deeply nested structures increase cognitive load and make queries harder to understand and modify.

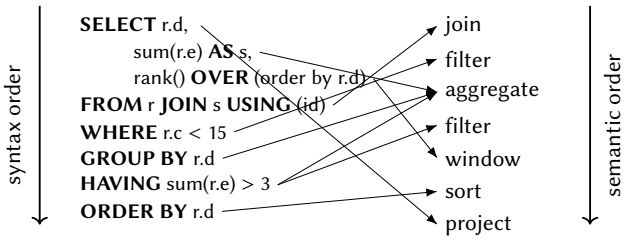

Figure 1: SQL Clause Order: Syntactic vs. Semantic Execution (From [5, 7])

Second, SQL has limited support for expressive abstraction for reuse. While SQL provides views and common table expressions (CTEs), they do not support parameterized reuse for defining a reusable query fragment with different parameters, such as input tables or filter conditions. As a result, similar logic is copied or slightly modified across a query or workload, which introduces duplication and reduces readability. An analysis of 138,993 queries on a commercial data warehouse confirms that 75.7% contain subquery nesting; structural duplication patterns recur over 80,000 times across the workload. Recent proposals, such as Google Pipe Syntax [7] and PRQL[5], address the first issue, the gap between syntax order and logical evaluation flow; however, they still do not directly support parameterized reusable abstractions.

Our goal is to improve query readability while preserving the semantics of SQL and compatibility with existing query engines. To achieve this, we employ Wvlet[1], a query language that provides pipeline-style syntax aligned with logical evaluation flow, making query structure more consistent with logical plans, as well as parameterized `model` abstractions for reusable query definitions

[1]https://wvlet.org/wvlet/, pronounced as weave-let. Last accessed August 2026.

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828140

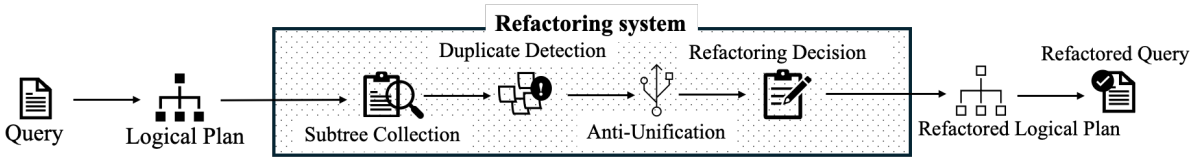

Figure 2: Workflow of duplicate logic consolidation

with different arguments. By leveraging these functionalities, we develop a refactoring system on top of Wvlet that improves readability by 1) extracting duplicated logic using parameterized `model` abstractions and 2) flattening nested query structures to eliminate unnecessary nesting. The system also analyzes query syntax and visualizes readability issues.

The purpose of this demonstration is to let participants interactively use the refactoring system to experience the following:

- How complex analytical SQL queries can be transformed into more readable Wvlet queries that are aligned with logical dataflow.
- How duplicated logic and unnecessary nesting are reduced through refactoring: reusable model extraction and join flattening.
- How the readability and correctness of the refactored queries are validated through structural metrics and in-browser semantic equivalence checking.

2 READABILITY-ORIENTED REFACTORING

Our goal is to reduce the structural complexity of SQL queries while preserving their semantics, in order to improve readability and maintainability.

2.1 Refactoring System

We build a readability-oriented refactoring system that operates on logical plans rather than raw query text, and uses Wvlet as the target language for refactored output. This design provides three key advantages. First, logical plans normalize away superficial differences such as formatting and alias naming, making it easier to detect semantically similar structures. Second, plan-level refactoring is less fragile than text-based rewriting because it operates on logical structure rather than surface syntax, enabling refactoring that preserves the original query semantics. Third, Wvlet provides a flow-style syntax aligned with logical evaluation flow, together with parameterized `model` abstractions for reusable query definitions. These properties make Wvlet not only a more readable query language, but also an effective target for plan-based refactoring. Our system consists of two main refactoring operations.

Duplicate logic extraction. This operation improves readability by consolidating duplicated computation patterns into reusable abstractions. As illustrated in Figure 2, the system first converts an input query into a logical plan and collects candidate subtrees. It then detects structurally similar subplans based on operator composition, allowing it to identify repeated computation patterns even when surface syntax differs. To generalize these patterns, the system applies *anti-unification* [6], which separates shared logic from varying elements. The shared logic is materialized as a reusable Wvlet `model` and the varying elements are lifted into its parameters. The refactored logical plan is then rendered as a Wvlet program with parameterized `model` definitions, which effectively removes duplicated logic.

Not all detected duplicates are worth consolidating as reusable models. Therefore, the system makes a selective *refactoring decision* step that balances the benefit of removing duplication against the complexity introduced by parameterization.

Join flattening. This operation improves readability by reducing unnecessary nesting. Many SQL queries contain redundant grouping introduced by subqueries or excessive parenthesization. Such structures increase indentation depth and visually separate related components, which makes queries harder to understand. Our refactoring system simplifies these structures by removing redundant grouping layers in the logical plan when the transformed plan remains equivalent to the original. In particular, it removes redundant parentheses and grouping nodes in inner-join structures, while avoiding rewrites around outer joins. After flattening, queries can be rendered in a more linear, top-to-bottom form that is easier to inspect and that also improves alignment between query syntax and logical evaluation flow.

In practice, join flattening is applied first to normalize the plan structure, after which duplicate detection operates on the flattened representation; this ordering ensures that structurally equivalent subplans are not obscured by redundant grouping layers. Figure 3 illustrates the combined effect on TPC-H Q11: the three-table join subquery duplicated in the `HAVING` clause is extracted into a reusable `model` after flattening, reducing the query from 74 to 64 tokens (−14%).

In the next section, we present the architecture of the demonstration system and show how users can interactively explore these refactoring operations.

2.2 Readability Metrics

To evaluate the effects of duplicate abstraction, join flattening, and Wvlet rendering on readability, we use three core readability metrics together with two auxiliary ones.

- **DRY (Don't Repeat Yourself)**: Measures the degree of logic duplication. Computed over logical plan structures rather than surface text, it identifies semantically equivalent fragments even if aliases or formatting differ.
- **SN (Syntactic Nesting)**: Measures the depth and complexity of nested subqueries. Queries with flatter structures achieve better SN scores.
- **SSOA (Syntactic–Semantic Order Agreement)**: Quantifies the mismatch between the written fixed-clause order (e.g., SELECT-FROM-WHERE) and the logical data flow.

Table 1: Readability metrics for TPC-H Q11 (Important Stock Identification).

Metric	DRY	SN	SSOA	JI	PR
SQL	0.077	0.750	0.667	0.550	0.295
Wvlet	0.077	0.750	1.000	1.000	0.426
Wvlet (refactored)	0.778	0.750	1.000	1.000	0.426

```
SELECT ps_partkey,
  SUM(ps_supplycost * ps_availqty)
    AS value
FROM partsupp
JOIN supplier
  ON ps_suppkey = s_suppkey
JOIN nation
  ON s_nationkey = n_nationkey
WHERE n_name = 'GERMANY'
GROUP BY ps_partkey
HAVING
  SUM(ps_supplycost * ps_availqty) >
  (SELECT
     SUM(ps_supplycost * ps_availqty)
     * 0.0001
   FROM partsupp
   JOIN supplier
     ON ps_suppkey = s_suppkey
   JOIN nation
     ON s_nationkey = n_nationkey
   WHERE n_name = 'GERMANY')
ORDER BY value DESC;
```

```
model auto_filter = {
  from partsupp
  join supplier
    on ps_suppkey = s_suppkey
  join nation
    on s_nationkey = n_nationkey
  where n_name = 'GERMANY'
}
from auto_filter
group by ps_partkey
where
  SUM(ps_supplycost * ps_availqty) > {
    from auto_filter
    select
      SUM(ps_supplycost * ps_availqty)
      * 0.0001
  }
select ps_partkey,
  SUM(ps_supplycost * ps_availqty)
  as `value`
order by `value` desc
```

Figure 3: TPC-H Q11: SQL (left) vs. Wvlet refactored (right).

Higher alignment reduces the cognitive load required to mentally reorder operations.

Additionally, we use auxiliary metrics: **JI (Join Intent)** for evaluating how clearly join relationships are expressed, and **PR (Predicate Readability)** for the complexity of filtering conditions.

Table 1 demonstrates these metrics on TPC-H Q11, a query that contains a correlated subquery in its `HAVING` clause and repeated aggregation logic —representative of the structural complexity issues motivating this work. We separate the evaluation into two stages: SQL-to-Wvlet translation and subsequent plan-based refactoring. The translation step alone achieves perfect SSOA alignment (0.667⟶1.000), as Wvlet's pipeline syntax directly maps clause order onto logical evaluation flow. The refactoring step then substantially improves DRY (0.077⟶0.778) by consolidating the repeated aggregation subquery into a reusable `model` definition. SN remains constant across all stages for this query because Q11's join structure contains no redundant grouping layers amenable to flattening; Section 4.2 presents workload-level evidence where SN improvements are observable.

3 SYSTEM ARCHITECTURE

We implement *Wvlet Cognitive Workbench* (WCW), a browser-based demonstration system for readability-oriented query refactoring. As illustrated in Figure 4, WCW operates entirely on the client side, leveraging Web Workers to ensure a responsive UI. The system architecture is organized into four tightly integrated layers: Interaction, Transformation, Assessment, and Verification.

Interaction Layer. The front-end provides a dual-editor interface. Users can inspect an individual query (*Single Query* mode) or extract reusable models across a workload (*Batch Refactor* mode). Rather than just outputting translated code, it visually explains

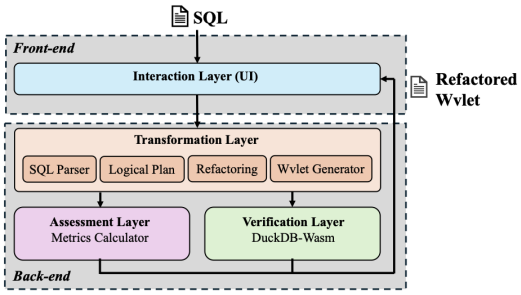

Figure 4: WCW System Architecture

improvements through side-by-side code comparisons, logical-plan tree visualizations, and a cognitive dashboard summarizing readability metrics.

Transformation Layer. The core engine translates an input SQL statement into a logical plan, applies refactoring operations (duplicate logic extraction and join flattening), and generates Wvlet code. To avoid dependence on external services during live demonstrations, transformation is executed locally in the browser via a Scala.js-based Web Worker. If local execution is unavailable, the system falls back to a REST API, and if that is also unavailable, it uses precomputed outputs for representative queries. The layer also translates Wvlet back to SQL for semantic validation.

Readability Assessment Layer. This module computes complementary readability metrics for both SQL and Wvlet codes. We evaluate the readability metrics, DRY (duplicated substructures), SN (nesting depth), SSOA (alignment between syntactic order and semantic evaluation flow), JI (explicit vs. implicit join ratio), and PR (predicate complexity). These metrics are combined into a weighted summary score, R_{core} (default weights available at the project repository), to provide an overall view of readability improvement, while the individual metrics highlight which specific aspects have improved.

Semantic Verification Layer. Improved readability is meaningful only if semantics are preserved. A crucial design choice of WCW is its in-browser execution engine using DuckDB-Wasm. The system executes both the original and regenerated SQL over a local sample dataset (e.g., TPC-H). It then bidirectionally compares outputs using `EXCEPT`-based difference queries (evaluating tuples in $A \setminus B$ and $B \setminus A$). This confirms exact result-set equality without requiring any external DBMS connection, lowering the barrier for reviewers and participants.

4 DEMONSTRATION

The goal of this demonstration is to provide an interactive experience of readability-oriented query refactoring on realistic analytical workloads. Participants interact with the Wvlet Cognitive Workbench (WCW) through a browser-based interface, where they can iteratively explore how structural issues in SQL queries are identified, transformed, and validated.

Figure 6: Interactive workload-level refactoring, showing extracted shared models and aggregated readability improvements.

WCW supports two complementary modes: *Single Query* mode for interactive exploration of individual queries, and *Batch Refactor* mode for workload-level analysis. In both modes, participants can either paste their own SQL queries or select from predefined workloads such as TPC-H, JOB, and curated examples involving nested queries, joins, and aggregations.

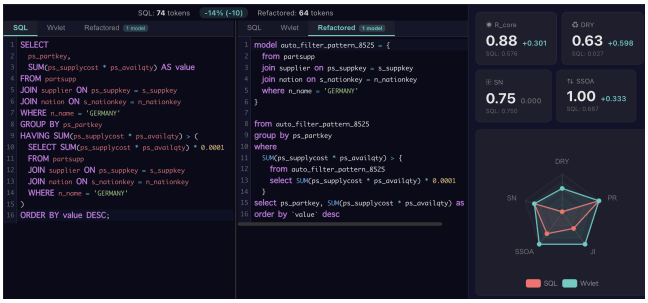

Figure 5: Interactive interface for single-query refactoring, showing side-by-side comparison, metric improvements, and semantic validation results.

4.1 Interactive Refactoring of Single Query

The demonstration begins with a single-query scenario. Participants may paste a SQL query or select one from the built-in workload browser. After pressing *Analyze* button, WCW parses the query, generates its Wvlet representation, and, when applicable, produces a *Refactored* version that consolidates duplicated logic into reusable `model`s and reduces unnecessary nesting. The original SQL statement, the generated Wvlet code, and the refactored result are presented side by side in the interface (Figure 5).

Participants can evaluate the effect of refactoring through metrics such as DRY, SN, SSOA, and the aggregate score R_{core}, and confirm that the refactored query remains semantically equivalent to the original. As a representative example, TPC-H Q11 yields substantial improvements across multiple metrics (see Section2.2 and Table 1). In typical cases, this interaction completes within sub-second response time.

4.2 Interactive Refactoring Across Query Workload

The second part of the demonstration extends the interaction to workload-level analysis. Participants switch to *Batch Refactor* mode and provide multiple queries either by selecting a predefined workload or by uploading a SQL file. By pressing the *Refactor All* button, WCW analyzes the workload as a whole and performs cross-query refactoring.

As shown in Figure 6, WCW identifies recurring structures across queries and consolidates them into shared `model` definitions. The interface provides both query-level and workload-level views, enabling participants to inspect how cross-query redundancy is detected, abstracted, and reduced. Unlike the single-query mode, this mode surfaces global structural patterns that are invisible when queries are analyzed in isolation—demonstrating how readability-oriented refactoring scales from per-query simplification to workload-wide abstraction. On the JOB benchmark (113 queries), R_{core} improves from 0.450 to 0.803.

The system is designed to remain robust and responsive even at large scale. Batch processing incurs higher latency than single-query interaction due to cross-query analysis; the system mitigates this by providing progressive feedback through partial results. If global abstraction fails for any query, WCW falls back to per-query refactoring, ensuring that participants can always observe meaningful transformations.

5 CONCLUSION

Overall, the demonstration provides an end-to-end workflow for exploring whether complex SQL queries can be rewritten into more readable forms while preserving semantics. By combining interactive refactoring, metric-based assessment, and in-browser semantic verification, WCW enables participants to experience both the readability benefit and the safety of plan-based query refactoring.

ACKNOWLEDGMENTS

This work was supported by JSPS KAKENHI Grant Number 23K17456.

REFERENCES

[1] Raymond PL Buse and Westley R Weimer. 2009. Learning a metric for code readability. *IEEE Trans. Softw. Eng.* 36, 4 (2009), 546–558.
[2] Don Chamberlin. 2023. 49 Years of Queries. In *Proc. SIGMOD Companion*. 1.
[3] Nelson Cowan. 2001. The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behav. Brain Sci.* 24, 1 (2001), 87–114.
[4] Torsten Grust. 2017. Take everything from me, but leave me the comprehension: invited talk. In *Proc. DBPL*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3122831.3130856
[5] Thomas Neumann and Viktor Leis. 2024. A Critique of Modern SQL and a Proposal Towards a Simple and Expressive Query Language. In *CIDR*.
[6] Frank Pfenning. 1991. Unification and anti-unification in the Calculus of Constructions. In *LICS*, Vol. 91. 74–85.
[7] Jeff Shute, Shannon Bales, Matthew Brown, Jean-Daniel Browne, Brandon Dolphin, Romit Kudtarkar, Andrey Litvinov, Jingchi Ma, John Morcos, Michael Shen, et al. 2024. SQL has problems. We can fix them: Pipe syntax in SQL. *Proc. VLDB Endow.* 17, 12 (2024), 4051–4063.