Adaptive Risk Scoring Queries for Longitudinal Student Mental Health Data

Hemit Rana[1], Sarvesh Solanke[1], Charmiben Patel[1], Chetan Thakur[1],
Jasmeen Kaur[1], Harpreet Singh[1], Anurag Sharma[1]
[1]University of Windsor, Windsor, ON, Canada
rana6c@uwindsor.ca, solanke1@uwindsor.ca, patel4mc@uwindsor.ca, thakurc@uwindsor.ca,
jasmeen1@uwindsor.ca, singh434@uwindsor.ca, sharm2u1@uwindsor.ca

***Abstract*—Universities collect longitudinal student mental health data and deploy risk-scoring queries to identify at-risk students. However, risk definitions evolve as institutions update policies and incorporate new insights—thresholds, weights, contributing factors, and even aggregation operators change frequently rather than remaining static. This project treats *semantic change as a first-class workload characteristic* and defines a change model spanning parametric updates (weights, thresholds), structural updates (adding/removing factors, operators, windows), and hybrid relational+vector updates. We will (a) express adaptive risk-scoring queries declaratively over a polyglot database (relational + vector embeddings), (b) detect semantic similarity across query versions to reuse stable subexpressions, and (c) evaluate four execution strategies (SQL recomputation, materialized views, incremental maintenance, and window-function-centric plans) on longitudinal student mental health data. The goal is a systems contribution: comparative evidence and strategy-selection rules that quantify latency, storage, and maintenance trade-offs under semantic evolution—insights that current benchmarks and optimizers (which assume static semantics) do not provide. Results will inform practical system design and future database optimizer development.**

I. Introduction and Motivation

Universities increasingly deploy longitudinal well-being surveys to track student stress, sleep quality, academic engagement, and emotional state over time. Risk models built on these data are critical: they identify students at high risk of negative outcomes, enabling timely institutional interventions. A typical risk model at a midsize university (5000 students) aggregates 20+ indicators across 2-4 survey cycles per year, producing risk scores used in real-time counseling decisions. The problem is not theoretical: when definitions change, recomputing all risk scores from scratch can take 10–100$\times$ longer than running the same query repeatedly under static semantics. However, unlike traditional database applications where query semantics stabilize once implemented, risk models evolve continuously. When evidence emerges that sleep quality is critical, institutions add it as a factor (structural change). When outcome data shows that the stress threshold was miscalibrated, thresholds are adjusted (parametric change). When new diagnostic instruments are adopted, aggregation logic changes. When vector embeddings of student free-text responses (e.g., sentiment analysis of stress descriptions) become available, risk definitions incorporate vector similarity thresholds (hybrid change). Institutional policy changes trigger wholesale restructuring of models across both structured and unstructured signals. This reality—semantic evolution of analytical queries—is fundamentally different from the assumption that dominates database systems research: that query definitions are static once deployed, and that the only source of variability is data changes. Yet database research offers no principled guidance on which execution strategy (materialized views, incremental computation, or window-function-centric plans) is optimal when query logic itself changes at a given frequency. Modern optimizers, benchmarks, and cost models all assume static semantics. This project addresses this gap by designing and evaluating concrete execution strategies for evolving queries, providing empirical evidence of which strategies dominate under different workload characteristics. The work is timely and practical: modern analytical systems (used in healthcare, finance, and policy) all exhibit this workload pattern—yet it remains unexplored by the database systems community and represents a critical blind spot in current optimization strategies.

II. Problem Statement

We define the evolving query semantics problem as follows. A risk-scoring query is parameterized by a *risk definition* D specifying: (i) a set of factors F (e.g., stress, sleep, academic performance), (ii) weights w for combining factors (e.g., $w_{\text{stress}} = 0.4$), (iii) aggregation granularity (e.g., rolling 12-week window), (iv) thresholds determining risk classification (e.g., score $>$ 75th percentile $\Rightarrow$ high-risk), and (v) vector similarity thresholds for text-derived signals (e.g., keywords in free-text responses). Allowed semantic changes include three categories: (1) *parametric changes*: adjusting weights, thresholds, or similarity cutoffs while keeping the set of factors fixed; (2) *structural changes*: adding or removing factors, changing aggregation functions or temporal windows; (3) *hybrid changes*: alterations that couple structured and vector-based signals (e.g., upweighting an embedded text feature in risk calculation). Given a longitudinal dataset of student responses (structured and unstructured) and a sequence of risk definitions $D_1, D_2, \ldots, D_k$ representing changes over time, the problem is: *what execution strategy minimizes the total cost (latency + storage + maintenance) of computing risk scores under all definitions?* Concretely, should an institution materialize aggregate views, recompute from scratch, incrementally update

cached results, or use window functions—and does the answer depend on the frequency (weekly vs. monthly vs. quarterly) and type (parametric vs. structural vs. hybrid) of semantic changes it expects? The cost depends on (a) the *frequency* of semantic changes, (b) the *type* of change, and (c) the *size* of the dataset and number of students affected. This is a database systems problem because the answer depends on the choice of execution strategy: recomputation, materialized aggregates, incremental computation exploiting stable subexpressions, or window-function-centric plans. Modern systems do not model this workload or provide guidance on strategy selection.

III. Related Work

Materialized View Maintenance and Incremental Computation. The foundational work by Gupta and Mumick [1] established that precomputing and maintaining materialized views can dramatically accelerate repeated queries. Their cost model analyzes how to minimize the cost of view refresh operators when data changes, providing a framework for deciding which views to materialize. However, their framework—and subsequent sophisticated work by Nikolic et al. [2] on delta-based incremental maintenance using recursive queries and change propagation—assumes the view definition (the logical structure of the query) is *fixed*. Cost is parameterized by data volume and data change rates, not by definition evolution. When a SQL view is defined as `CREATE MATERIALIZED VIEW risk AS SELECT ... WHERE stress_weight=0.4`, these works tell us how to refresh that view efficiently when student data changes. But they cannot answer: what should we do when the stress_weight parameter itself changes from 0.4 to 0.5? Should we still maintain the materialized view, or is recomputation now cheaper? We differ fundamentally: we assume the query definition evolves and ask whether the optimal execution strategy (materialization, recomputation, or incremental) changes as a function of definition evolution frequency and type.

Adaptive Query Processing. Arasu et al.'s STREAM system [3] pioneered adaptive query processing for continuous streaming data, enabling queries to adjust execution based on arriving data characteristics. Similarly, Babu and Widom [4] studied continuous queries over data streams, investigating how buffer utilization and arrival rates affect plan selection. Psaroudakis et al. [5] extended adaptivity to handle cardinality misestimates during execution via mid-query re-optimization. All these systems define adaptivity as runtime response to variable *data* characteristics or *resources*—data distribution, selectivity, arrival rates, memory availability. The critical distinction: these systems optimize *how* to execute a *fixed* query given varying inputs. Our work asks a different question: how to execute *different* queries (definitions). When a risk definition changes from "stress weight: 0.4" to "stress weight: 0.5," existing adaptive systems either re-optimize the same logical query plan or ignore the change. We explicitly model and measure the execution consequences of the definition change itself—and we show that the optimal strategy depends on how frequently definitions change, what types of changes occur (parametric vs. structural), and the size of the dataset affected.

Learned and Cost-Model-Driven Optimization. Recent work on learned query optimizers represents significant progress in handling complex execution decisions. Bao [6] applies machine learning to predict which physical plan will have minimum latency, improving on traditional cardinality-based cost models by learning from historical execution traces. Contemporaneously, Kersten et al. [7] provided detailed microbenchmarking of compiled vs. vectorized execution engines, showing how low-level execution strategies interact with data and hardware. Both advances assume the *logical query structure* is known and fixed a priori. Bao cannot learn cost predictions when the query definition itself varies; its training set assumes static query logic. Cost models in both systems encode the semantics of the query once, then optimize execution. None of these works ask the empirical question our project addresses: does materialized view strategy A dominate incremental strategy B under weekly semantic changes but not monthly changes? Does the dominance profile flip between parametric and structural changes? We provide the systematic evidence that answers this.

Polyglot Databases and Hybrid Search. Recent research on hybrid relational-vector systems [8] has advanced the state of combining relational tables with vector embeddings for semantic search and ML-over-relational analytics. Dong et al. studied how to optimize indices and join orders when mixing exact relational predicates with approximate vector similarity. Concurrently, frameworks integrating databases with machine learning [9] investigate feature management, model training pipelines, and ensuring consistency between relational and learned representations. These works are advances in polyglot data management—but all assume *static* semantic definitions. A risk definition might specify "return students where stress ¿ threshold AND embed_similarity(free_text, 'burnout') ¿ 0.8," but once deployed, the definition remains fixed. Real workloads violate this assumption: text-derived features (vector embeddings) and relational definitions co-evolve. When a new understanding of burnout emerges, both the embedding model and the similarity threshold may change. When a policy decision adds a new vector-based risk signal (sentiment analysis of support-seeking language), the definition gains a new factor. We extend polyglot systems by measuring how vector and relational components co-evolve in practice, and showing that semantic change impacts cross-model execution (join selectivity, vector similarity thresholds, materialization decisions) in ways not captured by single-model analysis.

Summary and Positioning. Existing research implicitly assumes analytical queries are semantically static once deployed. Materialization research asks: which views to precompute for static queries? Adaptive processing asks: how to handle variable data for static queries? Learned optimizers ask: which plan for static queries? Polyglot systems ask: how to optimize mixed relational-vector execution for static definitions? We challenge the foundational assumption and ask a new question: what execution strategy is optimal when *the query definition*

itself changes? Our contribution is not a new optimization technique, algorithmic improvement, or index design. Rather, we identify and systematically characterize an underexplored workload dimension. We provide (1) a formal model of semantic change (parametric, structural, hybrid), (2) concrete execution strategies specialized for evolving definitions, and (3) empirical evidence of strategy selection rules—demonstrating that the optimal approach depends on the frequency, type, and magnitude of semantic changes. Modern database systems, benchmarks (TPCH, TPDS, industry evals), and cost models are collectively blind to this dimension because they assume static semantics. Our systematic evaluation on real student health data fills that gap and provides actionable guidance for practitioners and optimizers designing future systems.

IV. Novelty and Contributions

What assumption are we breaking? The core assumption in all prior database optimization and materialization work is that *analytical queries are semantically static*. Optimizers minimize latency given a fixed logical plan. Materialization strategies precompute results for static queries. Cost models calculate amortized cost over many executions of the *same* query definition. We challenge this assumption by studying workloads where the *query definition itself changes*, and show that this orthogonal dimension of variability creates distinct optimal strategies and reveals trade-offs not visible in static-query analysis.

Our contributions are: (1) *Semantic Change Model:* We formally categorize allowed changes—parametric (weight/threshold/similarity adjustments), structural (adding/removing factors, changing operators or windows), and hybrid (alterations coupling relational and vector components)—and provide change annotations that guide detection of reuse opportunities and strategy selection. (2) *Similarity-Aware Reuse:* We design mechanisms that detect unchanged subexpressions across query versions (stable windows, shared factors, invariant aggregates in both relational and vector domains) and avoid full recomputation, going beyond traditional delta-based incremental maintenance by considering definition changes. (3) *Polyglot Evaluation:* We measure how text-derived signals stored as vectors co-evolve with structured risk definitions in real workloads. We show that semantic change impacts cross-model execution (join selectivity, vector similarity thresholds) in ways not captured by single-model analysis. (4) *Empirical Guidance and Impact:* Through systematic measurement on real student health data under controlled semantic-change workloads, we provide concrete evidence of which strategy dominates under different change frequencies and types—actionable insights for practitioners selecting analytics solutions and for database designers building future optimizers and cost models. Results will be disseminated as a systems workshop paper or short conference paper at SIGMOD, VLDB, or CIDR, contributing to the emerging intersection of adaptive databases and evolving analytical semantics. The novelty is not in proposing a new optimizer algorithm, but in identifying and empirically characterizing this underexplored workload dimension that modern systems and benchmarks ignore.

V. System Design and Algorithm Overview

We build a straightforward system using standard tools: PostgreSQL (with pgvector), Python, and SQL. We do **not** build a new optimizer or database engine; instead, we express each strategy as concrete implementations and measure them directly. The workload is deterministic: define a baseline risk model, apply semantic changes from a trace file, and for each change, execute all four strategies, recording latency, storage, and maintenance costs.

Example implementations: The SQL recomputation strategy executes: `SELECT student_id, 0.4 * stress + 0.3 * sleep + 0.3 * academic FROM aggregates GROUP BY student_id`, measuring time via `EXPLAIN ANALYZE`. Materialized views pre-execute aggregate views, then on definition change, issue refresh commands and re-join. The incremental strategy parses changes, identifies affected factors via a dependency graph, and recomputes only impacted portions. Window-function strategy uses SQL OVER clauses to measure sensitivity to window vs. factor changes.

The data model consists of six tables: *students*, *survey_cycles*, *questions*, *structured_responses*, *vector_responses*, and *risk_definitions* (serialized as JSON). We implement four strategies: (1) *SQL Recomputation* — fresh query per change; (2) *Materialized Views* — refresh affected views; (3) *Incremental Computation* — recompute only affected components; (4) *Window-Function-Centric* — minimize redundant time-based computation. A Python DSL compiler generates strategy-specific SQL from JSON definitions. Reuse detection is implemented via string/AST matching. The evaluation harness executes all four strategies per change, logging results to CSV.

VI. Implementation Plan

Languages and Tools: PostgreSQL 14+ with pgvector extension, Python 3.9+, pandas and sqlalchemy libraries. Embeddings generated using sentence-transformers library. No custom database engine or optimizer written. **Dataset:** American College Student Health Survey (ACSHS) public dataset or equivalent, containing structured responses (stress, sleep, academic engagement scores) and free-text responses (open-ended stress descriptions). Ingest $\sim$5,000–10,000 student records across 6–8 survey cycles. **Components:** (1) PostgreSQL schema setup (SQL DDL); (2) data ingest pipeline (pandas); (3) vector embedding generation and storage (sentence-transformers + SQL); (4) four strategy implementations (SQL procedures + Python wrappers); (5) DSL compiler (Python); (6) evaluation harness (Python timing loops, result logging). **Team Allocation and Feasibility:** With 7 team members working in parallel on independent phases (schema/data: 2 people; baseline+views: 2 people; incremental+window-functions: 2 people; DSL+evaluation: 1 person), total effort is 9–10 weeks. Each phase has concrete deliverables and can

be validated independently, enabling concurrent progress. No custom algorithmic or compiler work required; all components use standard libraries and well-documented PostgreSQL extensions. **Scope:** Measure latency (∼seconds per strategy), storage overhead (MB for materialized views), and maintenance cost (refresh time). Total implementation effort: concrete, achievable, and well-scoped for an academic project.

VII. Risks and Mitigation

Risk 1 — Data Availability: ACSHS dataset may be limited or require IRB approval. *Mitigation:* We identify and pre-download public dataset sources (University of Minnesota SRCD, APA Psych database). If real data unavailable, we generate synthetic longitudinal survey data with realistic correlation structures using standard benchmarking techniques. **Risk 2 — Vector Embedding Computational Cost:** Generating 384-dimensional embeddings for thousands of free-text responses may be slow. *Mitigation:* Embeddings are generated once offline; we parallelize using Python multiprocessing. If needed, we reduce to 100–150 dimensions or use smaller models (DistilBERT vs. BERT). **Risk 3 — Strategy Implementation Complexity:** The incremental strategy requires careful dependency tracking and delta management. *Mitigation:* We start with a simple graph structure (factor $\rightarrow$ component $\rightarrow$ score) and implement incremental updates iteratively, testing against baseline recomputation at each step. If too complex, we fall back to comparing only recomputation, materialized views, and window functions (3 of 4 strategies still provide strong evidence). **Risk 4 — Limited Time for Extensive Evaluation:** Measuring all combinations of change frequency/type/magnitude is time-consuming. *Mitigation:* We prioritize the most common scenarios (monthly changes, parametric + structural shifts). We limit the number of semantic change events in the trace to 20–30 per experiment, not hundreds.

VIII. Timeline and Milestones

Weeks 1–2 (Schema & Data): Design PostgreSQL schema, download/ingest real or synthetic dataset, generate embeddings, validate data integrity. *Deliverable:* PostgreSQL database with 6 tables populated, 10,000 student records, 500 embeddings. **Weeks 3–4 (Baseline & Views):** Implement SQL recomputation strategy (15–20 SQL queries), implement materialized views strategy with refresh logic, measure latency for 5 baseline definitions. *Deliverable:* Two strategies fully working, initial timing data. **Weeks 5–6 (Incremental & Windows):** Implement incremental strategy with dependency graph, implement window-function strategy, test against baseline. *Deliverable:* All four strategies implemented and validated. **Weeks 7–8 (DSL & Evaluation):** Build Python DSL compiler, create change-trace file with 20–30 semantic changes (mix of parametric, structural, hybrid), run full evaluation loop, output results. *Deliverable:* CSV of latency/storage/cost for all strategies under all change scenarios. **Weeks 9–10 (Analysis & Documentation):** Analyze results, generate plots (latency vs. change frequency, strategy comparison), summarize findings, write final report. *Deliverable:* Final report with tables, plots, and conclusions. All phases are independent enough to parallelize among team members.

IX. References

References

[1] A. Gupta and I. S. Mumick, "Maintenance of materialized views: Problems, techniques, and applications," *IEEE TCDE Bull.*, vol. 18, no. 2, pp. 3–18, 1995.

[2] M. Nikolic, M. Olteanu, and J. Wen, "Incremental view maintenance for analytical queries," in *Proc. VLDB*, 2014, vol. 8, no. 1, pp. 25–36.

[3] A. Arasu, S. Babu, and J. Widom, "STREAM: The Stanford data stream management system," in *Proc. SIGMOD*, 2006, pp. 665–665.

[4] S. Babu and J. Widom, "Continuous queries over data streams," *ACM SIGMOD Rec.*, vol. 30, no. 3, pp. 109–120, 2001.

[5] I. Psaroudakis, T. Scheuer, and N. May, "Adaptive query processing on RAW data," in *Proc. SIGMOD*, 2016, pp. 993–1008.

[6] R. Marcus, P. Negi, H. Mao, N. Tatbul, M. Alizadeh, and T. Kraska, "Bao: Making learned query optimization practical," in *Proc. SIGMOD*, 2021, pp. 1275–1288.

[7] T. Kersten, V. Leis, T. Neumann, A. Pavlo, P. Boncz, and J. L. Kipf, "Everything you always wanted to know about compiled and vectorized queries but were afraid to ask," in *Proc. VLDB*, 2018, vol. 11, no. 13, pp. 2209–2222.

[8] X. Dong, S. Agarwal, S. Chen, M. Xiao, Y. Wang, and C. S. Prasad, "Hybrid search: When traditional databases meet vector search," in *Proc. SIGMOD*, 2024, pp. 1–18.

[9] I. F. Ilyas, R. K. Deepak, and A. S. Gunda, "Towards a unified framework for databases and machine learning," in *CIDR*, 2022, pp. 1–15.