Technical Perspective: Implementing Views for Property Graphs

Stefania Dumbrava
SAMOVAR - Télécom SudParis
ENSIIE, France
stefania.dumbrava@telecom-sudparis.eu

Property graph databases provide a flexible framework for integrating heterogeneous data, while supporting queries that can extract complex graph patterns. Still, efficiently implementing *graph views*—which abstract, transform, and unify data—remains a challenge. The authors address this by introducing techniques for defining, maintaining, and querying views over property graphs through custom query rewriting, materialization, and indexing strategies.

One of the key contributions of the work is designing a relational encoding of graph views that integrates core elements of very recent formalisms proposed by the database community. As GQL had not yet been officially published, the authors built on the foundations of the G-CORE language that directly informed the standard. Moreover, they incorporated features described in the PG-Schema and PG-Keys languages for schema and key constraints.

The framework leverages concepts from these languages and provides mechanisms for elegantly expressing more sophisticated views than previously possible. Indeed, while the Graph Pattern Matching Language (GPML), common to GQL and SQL/PGQ, could be instrumented to express views, it does not allow one to directly encode local graph transformations. Such transformations—expressing, for example, zooming in and out of subgraph patterns in provenance tracking—are prevalent in biomedical and social science applications and would benefit from adequate support.

The authors introduce a formalism for mapping graph views to tuple-generating dependencies and recursive Datalog rules, as a principled way to express graph transformations in a relational setting. This translation allows queries over views to be rewritten into relational expressions, ensuring efficient execution while maintaining schema constraints.

A major challenge in graph view transformations is supporting subgraph substitutions, where small portions of a graph are modified while preserving the broader structure. Current approaches require explicitly copying unchanged elements, making view definitions verbose and redundant. The paper introduces transformation views that simplify these operations by introducing default rules. These automatically copy all elements of the input graph except those explicitly modified or removed, significantly reducing redundancy, improving readability, and ensuring transformations maintain schema integrity. The paper also defines a notion of well-behaved views, guaranteeing that transformations are deterministic, non-conflicting—in that they disallow, for example, simultaneously deleting and mapping the same node—and schema-compliant.

Beyond transformations, the paper also explores efficient materialization and update strategies for graph views. While virtual views rewrite queries dynamically, materialized views store precomputed results, improving query performance, but requiring incremental updates when the underlying graph changes. The authors propose dependency tracking techniques that minimize recomputation by identifying and updating only affected parts of the view. This is particularly beneficial for dynamic applications, in which views need to evolve continuously while still remaining schema compliant.

To further enhance query performance, the paper introduces Subgraph Substitution Relation (SSR) indexing, a technique that accelerates queries over graph views by precomputing mappings between transformed and original subgraphs. SSR indexing optimizes pattern-matching operations by enabling queries to run directly on the transformed view, eliminating the need for costly recomputations.

The experimental evaluation comprehensively analyzes the effectiveness of the proposed techniques for processing transformation views. The authors implement their graph view benchmarking system and compare the performance of PostgreSQL, the Neo4j graph database, and the Datalog-based LogicBlox engine for view materialization, SSR indexing, and query execution over views, on diverse, real-world graphs. SSRs are shown to accelerate view materialization (up to 6.9× faster) and query execution (up to 6.4× faster), as well as to enable LogicBlox and PostgreSQL to achieve query performance comparable to Neo4j, highlighting their potential for efficient graph view management.

The paper proposes a formal framework for expressing views over property graphs in relational terms, integrating PG-Schema constraints, and introducing efficient transformation views, materialization strategies, and indexing techniques. By bridging the gap between relational and graph databases, it provides a foundation for scalable and flexible graph view implementations. These techniques are particularly relevant for dynamic graph applications.

This research opens the exciting perspective of integrating mechanisms for expressing graph views in the future versions of the official GQL standard, and of implementing these in real-world systems. As graph-based applications grow in complexity, efficient view management will remain essential for ensuring scalability, consistency, and interoperability.