March 2019 W3C workshop in Berlin on graph data management standards

Graph types and Language interoperation

Peter Furniss, Alastair Green, Hannes Voigt
Neo4j Query Languages Standards and Research Team
11 January 2019

We are actively involved in the openCypher community, SQL/PGQ standards process and the ISO GQL (Graph Query Language) initiative.

In these venues we in Neo4j are working towards the goals of a single industry-standard *graph query language* (GQL) for the property graph data model. We feel it is important that GQL should inter-operate well with other languages (for property graphs, and for other data models).

Hannes is a co-author of the SIGMOD 2017 G-CORE paper and a co-author of the recent book Querying Graphs (Synthesis Lectures on Data Management).

Alastair heads the Neo4j query languages team, is the author of the The GQL Manifesto and of Working towards a New Work Item for GQL, to complement SQL PGQ.

Peter has worked on the design and early implementations of property graph typing in the Cypher for Apache Spark project. He is a former editor of OSI/TP and OASIS Business Transaction Protocol standards.

Along with Hannes and Alastair, Peter has centrally contributed to proposals for SQL/PGQ Property Graph Schema.

We would like to *contribute to or help lead discussion* on two linked topics.

Property Graph Types

The information content of a classic Chen Entity-Relationship model, in combination with “mixin” multiple inheritance of structured data types, allows a very concise and flexible expression of the type of a property graph.

A named (catalogued) graph type is composed of node and edge types. A node type composes a single element type (which expresses many labels and their associated mandatory or optional properties); an edge type composes tail, head and edge element types). Element types can extend zero to many supertypes. This allows edge types to be expressed in terms of tail and head subtypes, allowing for maximally compressed type descriptions for the whole graph.

Uniqueness keys can be defined with respect to individual element types (and transitively for node types), and with respect to edge types (where properties of the tail/head nodes and edges can be used in the key). Interestingly, for language interoperation purposes, this allows a graph to be viewed as a set of tables with unique keys (which is quite independent of the actual graph database implementation strategy). This in turn opens the road to SQL queries over graph data (or tabular access control over graph data).

Property graph query languages which use labels as predicate operands (such as Cypher, PGQL, G-CORE and SQL/PGQ) can operate without awareness of element type inheritance, giving backward compatibility.

Current and future work building on these foundations focuses on the related topics of graph type inheritance (extending or restricting a whole graph type); graph type evolution, and partial typing (allowing untyped extensions to an instance of a graph type). There are interesting possibilities for standard tooling and language “refractions” of the metamodel of a property graph type.

Graph types can be used to provide strong typing in interfaces that accept graphs as inputs or produce output graphs. This leads to the second, related topic.

Language/model interoperation and graph program composition

One obvious case where graph typing can help composition is in query composition. openCypher proposals, as implemented in the Cypher for Apache Spark project, and the G-CORE language posit graph projection (including graph views). Graph typing (whether used prescriptively or descriptively) aids implementations of composable queries, and their optimization.

The GQL Scope and Features proposal, put forward within the ISO standards process, which was primarily authored by Stefan Plantikow and Tobias Lindaaker at Neo4j, introduces a more general concept. A graph processing *program* may have different aspects. It may be necessary to query for data declaratively, to explore data through incremental traversals, or to carry out computations on the graph.

The concept of *graph procedures* (of which graph functions are a subset) is advanced in that proposal. A graph procedure may be written in any language, but each language must share an understanding of the property graph model (and therefore of the metamodel or base type of all property graph types). For the satisfactory composition of procedures written in two domain-specific languages, it is necessary to be able to share graph type information. It may also be useful to cross data model boundaries (for example, between property and pure graphs, or between graphs and tables). Once again, property graph types are a valuable, and probably essential tool for facilitating complex data manipulation programs which entail viewing the same data through alternate model prisms.

This technical approach mirrors the social need to productively *bridge* between relational, semantic web/RDF and property graph data models and their associated languages.

Our contributions are posited on a philosophy of “cultural awareness and respect”, with necessary translations to accommodate these adjacent communities. We believe that a “single master language” like Latin, French or English (or a concoction like Esperanto) is unlikely to emerge or succeed in the context of modern data management. Property graph types (and their analogues for other models) can contribute significantly to structured polyglot interoperations.

Relationship to issues of KRR and computational statistics

The ability to transform from a graph of one type to a graph on another is relevant in current research thinking with respect to graph networks in the context of machine learning, arguably an aspect of computational statistics.

See in particular p.23 of this recent paper Relational inductive biases, deep learning, and graph networks, and the references to:

Li, Y., Vinyals, O., Dyer, C., Pascanu, R., and Battaglia, P. (2018). Learning deep generative models of graphs. In *Workshops at the International Conference on Learning Representations* (ICLR)

Kipf, T., Fetaya, E., Wang, K.-C., Welling, M., and Zemel, R. (2018). Neural relational inference for interacting systems. In *Proceedings of the International Conference on Machine Learning* (ICML).

Unusual points of view that we may bring to the workshop

None that we’re aware of.