Graph Data Models, Query Languages and Programming Paradigms [*]

[Tutorial Summary]

Alin Deutsch
UC San Diego
deutsch@cs.ucsd.edu

Yannis Papakonstantinou
UC San Diego
yannis@cs.ucsd.edu

ABSTRACT

Numerous databases support semi-structured, schemaless and heterogeneous data, typically in the form of graphs (often restricted to trees and nested data). They also provide corresponding high-level query languages or graph-tailored programming paradigms.

The evolving query languages present multiple variations: Some are superficial syntactic ones, while other ones are genuine differences in modeling, language capabilities and semantics. Incompatibility with SQL presents a learning challenge for graph databases, while table orientation often leads to cumbersome syntactic/semantic structures that are contrary to graph data. Furthermore, the query languages often fall short of full-fledged semistructured and graph query language capabilities, when compared to the yardsticks set by prior academic efforts.

We survey features, the designers' options and differences in the approaches taken by current systems. We cover both declarative query languages, whose semantics is independent of the underlying model of computation, as well as languages with an operational semantics that is more tightly coupled with the model of computation. For the declarative languages over both general graphs and tree-shaped graphs (as motivated by XML and the recent generation of nested formats, such as JSON and Parquet) we compare to an SQL baseline and present SQL reductions and extensions that capture the essentials of such database systems. More precisely, rather than presenting a single SQL extension, we present multiple configuration options whereas multiple possible (and different) semantics are formally captured by the multiple options that the language's semantic configuration options can take. We show how appropriate setting of the configuration options morphs the semantics into the semantics of multiple surveyed languages, hence providing a compact and formal tool to understand the essential semantic differences between different systems.

Finally we compare with prior nested and graph query languages (notably OQL, XQuery, Lorel, StruQL, PigLatin) and we transfer into the modern graph database context lessons from the semistructured query processing research of the 90s and 00s, combining them with insights on current graph databases.

PVLDB Reference Format:
A. Deutsch, Y. Papakonstantinou. Graph Data Models, Query Languages and Programming Paradigms. *PVLDB*, 11 (7): xxxx-yyyy, 2018.
DOI: https://doi.org/TBD

1. TUTORIAL TOPIC AND AUDIENCE

Numerous graph and nested databases promoted as SQL-on-Hadoop, NewSQL and NoSQL support Big Data applications. These databases generally support semi-structured, schemaless and heterogeneous data, which is the focus of this tutorial. Broadly, the data models are either graphs, or trees (nested data) as exemplified by JSON and variations thereof. The databases provide corresponding query languages. We review characteristic ones: Apache Hive [27], MongoDB [22], Couchbase's N1QL [12, 9], SPARQL [26], Neo4j's Cypher [13], Facebook's GraphQL [17], Apache Spark SQL [25], CosmosDB, Apache Gremlin [18], TigerGraph's GSQL [19]. We use SQL-92 as the anchor of SQL compliance and also comment on additional databases and query languages ([10, 11, 7, 6, 16, 5]).

The audience of this tutorial is roughly divided in three categories: First, developers that want to use these databases are overwhelmed by their number and want to make sense of the options provided in this space. Second, database language designers who are building new features (for graph and nested data) and are interested in a deep analysis of the available options. Third, database builders and researchers who work on expanding the databases' query language abilities. Both parties face the challenges described below regarding surveying and comparing models and query languages, past and present. Both parties need a deep understanding of what is different from plain SQL. This tutorial provides a deep understanding of the current data models and query languages of graph and nested data databases, hence enabling comparisons.

The tutorial does not limit itself to the current status of graph querying: The database builders and researchers need to draw lessons from the rich body of past research on nested [20, 24, 1], object-oriented [4] and semistructured data models and querying [14, 8, 23], which have been a topic of intense database research: They were first researched in the

[*]Supported by NSF IIS129263, NSF SHB1237174, Informatica Inc. gift and Couchbase Inc. gift.

Proceedings of the VLDB Endowment, Vol. 11, No. 7
Copyright 2018 VLDB Endowment 2150-8097/18/3.
DOI: https://doi.org/TBD

form of labeled graphs in the mid-90s. Then semistructured data research boomed in the form of XML and its labeled tree abstraction. Many of the important language design issues of the first eras must be recalled in the new era of semistructured data.

More broadly, we compare SQL (which we use as a baseline) with the recent crop of graph and nested databases and connect the recent activity around graph and JSON querying to the (much richer) past activity on nested relational, OQL and XML/labeled tree models and respective query languages and implementations.

2. CHALLENGES IN COMPREHENDING THE SPACE OF GRAPH AND NESTED DATA DBS

A first challenge is that the evolving semistructured query languages have many variations. (By "semistructured" we refer to both graph and nested data.) Some variations are due to superficial syntactic differences that simply create "noise" when one tries to understand and compare systems. However, other variations are genuine differences in query language capabilities and semantics.

Indeed, the evolving query languages of both the genuine semistructured databases and the SQL/JSON databases fall short of full-fledged semi-structured query language capabilities - as set by early academic efforts in the 90s.[1] The designers of the new query languages can gain by understanding and picking the salient features of past full-fledged *declarative* query languages for non-relational data models: OQL [4], the nested relational model [20, 24, 1], XQuery, and other XML query languages [23, 14, 8].

Part of the confusion around semistructured query languages is derived from the lack of compatibility with the well known SQL. In the interest of broadening the audience, this tutorial assumes that the audience is well-aware of SQL and the standard material of graduate textbooks on SQL system implementation. The tutorial does not assume knowledge of other query languages. Consequently we explain the JSON model and query languages as minimal extensions to SQL. In particular SQL-92, as it represents the well-supported common denominator of all SQL systems and corresponds to normalized databases. The tutorial does not require knowledge of non-1NF, often proprietary, features that have been added to SQL-92. Rather it only requires textbook SQL-92 language and teaches the non-1NF concepts, as well as the graph concepts.

A final challenge in understanding the new space of semistructured data is the lack of a succinct, mathematically clear, formal syntax and semantics by the vendors.

In summary, the mentioned challenges and confusions hurt researchers and developers:

1. They inhibit a deep understanding of the capabilities and important idiosyncracies of the various query languages. Potential users can be lost in superficial details and miss fundamental points.

2. They impede progress towards declarative languages and systems for querying semi-structured data. Language designers and query processor implementors need to appreciate the available options, in order to proceed to well-designed fully-fledged languages and efficient implementations thereof.

3. A SYSTEMATIC SURVEY OF MODEL AND LANGUAGE OPTIONS AND VARIATIONS

Step 1: Extending the relational model and SQL for graphs As discussed above, part of the confusion is derived from the lack of compatibility with the well-known, baseline SQL, which both researchers and practitioners generally understand.

Towards a uniform explanation of the large space of current and past systems, this tutorial reduces the declarative languages to (minimally extended) SQL. We start by recalling the insight that graphs can be modeled as relational databases by using appropriate vertex tables and edge tables. Vice versa, most relational databases can be viewed as graphs whose vertices are tuples and whose edges are key-foreign key pairs. We develop this analogy and show that multi-way SQL join queries correspond to fixed-length multi-hop path navigation. We then introduce a series of minimal extensions that enhance the expressiveness of SQL towards reaching that of graph query languages.

One extension introduces controlled amounts of recursion via *path expressions* that specify reachability in the graph. We track the concept's evolution from early systems like OQL [4], Lorel [2], WebSQL [21], StruQL [15] via the standard XPath/XQuery [28] and the de facto standard Regular Path Queries (RPQs) [3], all the way to contemporary languages such as Cypher [13] and Gremlin [18]. Path expressions may start with a variable, and multiple path expressions may appear in the `FROM` clause, potentially correlated with previously defined variables of the same `FROM` clause. This correlation feature is ruled out by SQL-92 but it has been prominently present since OQL.

Additional extensions collaborate towards ensuring language compositionality, by enabling queries to output graphs. A key enabler is the ability to invent fresh values to model the identities of newly constructed nodes and edges. This ability has its roots in object-oriented languages and its various incarnations can be invoked by the programmer either explicitly or implicitly. We additionally demonstrate simultaneous construction of multiple linked tuples via each application of the `SELECT` clause. Moreover, we show full compositionality, in the sense that subqueries can appear anywhere, potentially creating nested results when they appear in the `SELECT` clause.

Query languages for unrestricted graphs *annotated with scalar data* are compared via reduction to this extended SQL.

We next shift attention to a highly important class of graphs: they are tree-shaped and annotated with non-1NF data. This class is motivated by the plethora of semistructured databases in circulation today for formats such as JSON and Parquet (as well as their XML precursors). Explicit id invention is not necessary any more, as the construction of tree structures is accomplished by nested (sub)queries. We draw the parallels between the two classes of languages (semistructured and graph) highlighting the simplifications emerging in the semistructured case. We incorporate into the discussion salient features of past full-

[1] Most semistructured databases also fall significantly short of full-fledged SQL capabilities.

fledged declarative query languages for non-relational data models: SQL non-1NF features (starting with SQL 2003), OQL, the nested relational model and query languages, and XQuery (and other XML-based query languages).

An important issue, mostly discussed in the context of semistructured query languages (but fundamentally applying broadly to graphs) is the treatment of schema. The discussion includes language features that allow pivoting/unpivoting from schema to data and vice versa, thus enabling "metadata" inspection.

The corresponding plethora of query languages is classified also via reduction to an SQL extension, *Configurable Graph SQL*. Neglecting temporarily the "configurable" aspect (discussed in Step 2), one may think of the presented language as an extension/modification of SQL-92 for graphs and semistructured data.

In this tutorial, a new student/researcher of graph and semistructured data, who missed the OQL and XQuery eras, will be able to absorb the essential teachings of OQL and XQuery while they are succinctly cast as a *minimally modified SQL*. We describe these modifications next, which will enable an audience member with SQL background to comprehend the fundamentals of the extension to genuine JSON databases with minimal effort.

After having taught Step 1, we will be able to show that multiple model and language differences are superficial syntactic differences.

Step 2: Substantial Semantic Differences However, not all differences are superficial. Furthermore, this tutorial does not suggest that the SQL extension (or some close descendant thereof) will become a standard and remove the many variations that are now found in this space. There is too much variation and legacy for such to happen. Yet, the language designers and researchers need to know now the design options that are available to them and the options that have been used by others, especially as pertaining to the handling of schema-less aspects (semantics for paths leading to nowhere, semantics for type mismatches, etc). Towards this goal the tutorial stresses the Configurable aspect of the presented SQL extension, which is essentially a guery language generator. Depending on the *configuration options* that are chosen for various features, different capabilities are assumed and different semantics emerge.

One particular example where configuration options capture differences concisely, is the behavior of paths of the various query languages in the absence of information. For example, consider a JSON object **`{a:1, b:2}`** and a path that navigates into the absent path **`c`**. Languages differ on what is the result of **`c`**. Is it an error? Is it a special value? If it is a special value, how does it behave in other features of the query language? Is the query writer given control on what special value may emerge or whether an error will be thrown? A configuration option captures these differences precisely.

By appropriate choices of configuration options, the Configurable SQL++ semantics morphs into the semantics of other query languages. Hence, the audience will be able to understand the essential differences between the various query languages, without being swamped by their superficial syntactic differences. Given the time constraints, the tutorial will present a few examples of issues and classifications, leaving the complete surveying for an online survey that the authors will have set up.

We expect that some of the results listed in the feature matrices describing configuration options will change in the next years as the space evolves rapidly. Despite the forthcoming changes, we expect the configuration-based aspect of our tutorial to remain a standing tool in understanding the space, since by understanding each database's capabilities in terms of applicable options, the reader can focus on the fundamental differences of the databases.

Step 3: Additional Features We will also discuss features, many of which coming from XQuery, that have not been captured by configuration options. These will prompt a more open-ended discussion of language designs and trade-offs. A notable one is type coercion - the approaches and the pros and cons.

Step 4: Languages with Operational Semantics Tied to the Computation Model Recently we have witnessed a trend towards development of high-level graph query languages that are deliberately not purely declarative, instead featuring an operational semantics tied to the underlying computation model, which is typically a Bulk Synchronous Parallel (BSP) instance presented as a variation of the Map-Reduce programming paradigm. Prominent examples are Gremlin and GSQL. We discuss this class of queries.

Open Issues Finally, we emphasize open issues in the expansion from structured to semistructured querying and briefly discuss interoperability challenges induced by language differences.

4. ADDITIONAL INFORMATION

Prerequisite Knowledge The attendees must have solid knowledge of SQL-92, since it is the baseline upon which the semistructured aspects are then added. Also solid knowledge of relational algebra. Knowledge of SQL-2003 and/or XPath are a plus but not required.

Alin Deutsch is a Professor of Computer Science and Engineering at UCSD. His research pertinent to this tutorial centers around the design and optimization of semi-structured query languages and on programming paradigms for graph data analytics. Alin is the recipient of an ACM PODS Test of Time Award, an ACM SIGMOD Top-3 Best Paper Award (together with tutorial co-author Yannis Papakonstantinou), an Alfred P. Sloan fellowship, and an NSF Career award. He has served as Senior Scientist of TigerGraph Inc., a start-up that offers an engine capable of real-time analytics on web-scale graph data. The analytic tasks are expressed in a high-level graph query language called GSQL, in whose design Alin was involved.

Yannis Papakonstantinou is a Professor of Computer Science and Engineering at UCSD. A common theme of his research is the extension of database platforms and query processors beyond centralized relational databases and into semistructured databases, integrated views of distributed databases and web services, textual data and queries involving keyword search. His research has received more than 14,500 citations, according to Google Scholar, most of which refer to his work on semistructured data, semistructured query processing and related middleware. In addition to his academic activity in middleware, semistructured data and query processing, Yannis was the Chief Scientist of Enosys Software, which built and commercialized an early Enterprise Information Integration platform for structured and semistructured data, utilizing XML and XQuery. The

Enosys Software was OEM'd and sold under the BEA Liquid Data and BEA Aqualogic brand names. Yannis currently serves also as a consultant to Amazon Web Services.

5. REFERENCES

[1] S. Abiteboul, P. C. Fischer, and H.-J. Schek, editors. *Nested Relations and Complex Objects*, volume 361 of *Lecture Notes in Computer Science*. Springer, 1989.

[2] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The lorel query language for semistructured data. *Int. J. on Digital Libraries*, 1(1):68–88, 1997.

[3] S. Abiteboul and V. Vianu. Regular path queries with constraints. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 12-14, 1997, Tucson, Arizona, USA*, pages 122–133, 1997.

[4] F. Bancilhon, S. Cluet, and C. Delobel. A query language for the O2 object-oriented database system. In *DBPL*, pages 122–138, 1989.

[5] A. Behm, V. R. Borkar, M. J. Carey, R. Grover, C. Li, N. Onose, R. Vernica, A. Deutsch, Y. Papakonstantinou, and V. J. Tsotras. ASTERIX: towards a scalable, semistructured data platform for evolving-world models. *Distributed and Parallel Databases*, 29(3):185–216, 2011.

[6] K. S. Beyer, V. Ercegovac, R. Gemulla, A. Balmin, M. Y. Eltabakh, C.-C. Kanne, F. Özcan, and E. J. Shekita. Jaql: A scripting language for large scale semistructured data analysis. *PVLDB*, 4(12):1272–1283, 2011.

[7] (BigQuery:) Standard SQL Query Reference. https://cloud.google.com/bigquery/docs/reference/standard-sql/.

[8] A. Bonifati and S. Ceri. Comparative analysis of five XML query languages. *SIGMOD Record*, 29(1):68–79, 2000.

[9] D. Borkar, R. Mayurum, G. Sangudi, and M. Carey. Have your data and query it too: From key-value caching to big data management. In *SIGMOD Conference*, 2016.

[10] Cloudera Impala. http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html.

[11] SQL Queries for Azure Cosmos DB. https://docs.microsoft.com/en-us/azure/cosmos-db/sql-api-sql-query.

[12] Couchbase. http://www.couchbase.com/.

[13] Neo4j's Query Language Cypher. https://neo4j.com/developer/cypher-query-language/.

[14] A. Deutsch, M. F. Fernández, D. Florescu, A. Y. Levy, and D. Suciu. A query language for XML. *Computer Networks*, 31(11-16):1155–1169, 1999.

[15] M. F. Fernández, D. Florescu, A. Y. Levy, and D. Suciu. Declarative specification of web sites with strudel. *VLDB J.*, 9(1):38–55, 2000.

[16] D. Florescu and G. Fourny. JSONiq: The history of a query language. *IEEE Internet Computing*, 17(5):86–90, 2013.

[17] Facebook's GraphQL. http://graphql.org/.

[18] Apache TinkerPop3 Gremlin. http://tinkerpop.apache.org/docs/current/reference/.

[19] TigerGraph's GSQL. http://doc.tigergraph.com/GSQL-Tutorial-and-Demo-Examples.html.

[20] G. Jaeschke and H.-J. Schek. Remarks on the algebra of non first normal form relations. In *PODS*, pages 124–138. ACM, 1982.

[21] A. O. Mendelzon, G. A. Mihaila, and T. Milo. Querying the world wide web. *Int. J. on Digital Libraries*, 1(1):54–67, 1997.

[22] MongoDB. http://www.mongodb.org/.

[23] J. Robie, D. Chamberlin, M. Dyck, and J. Snelson. XQuery 3.0: An XML query language, W3C candidate recommendation, 2013. http://www.w3.org/TR/2013/PR-xquery-30-20131022/.

[24] M. A. Roth, H. F. Korth, and A. Silberschatz. Extended algebra and calculus for nested relational databases. *ACM Trans. Database Syst.*, 13(4):389–417, 1988.

[25] Apache Spark SQL. https://spark.apache.org/sql/.

[26] The RDF Query Language SPARQL. https://jena.apache.org/tutorials/sparql.html.

[27] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Anthony, H. Liu, and R. Murthy. Hive - a petabyte scale data warehouse using Hadoop. In *ICDE*, pages 996–1005, 2010.

[28] The XQuery language. www.w3.org/TR/xquery, 2004.