Semistructured Models, Queries and Algebras in the Big Data Era *

[Tutorial Summary]

Yannis Papakonstantinou
Computer Science and Engineering, UCSD
yannis@cs.ucsd.edu

ABSTRACT

Numerous databases promoted as SQL-on-Hadoop, NewSQL and NoSQL support semi-structured, schemaless and heterogeneous data, typically in the form of enriched JSON. They also provide corresponding query languages. In addition to these *genuine JSON databases*, relational databases also provide special functions and language features for the support of JSON columns, typically piggybacking on non-1NF (non first normal form) features that SQL acquired over the years. We refer to SQL databases with JSON support as *SQL/JSON databases*.

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 genuine JSON databases, while the table orientation of SQL/JSON databases often leads to cumbersome syntactic/semantic structures that are contrary to the semistructured nature of JSON. Furthermore, the query languages often fall short of full-fledged semistructured query language capabilities, when compared to the yardstick set by XQuery and prior works on semistructured data (even after superficial model differences are abstracted out).

We survey features, the designers' options and differences in the approaches taken by actual systems. In particular, we first present a SQL backwards-compatible language, named *SQL++*, which can access both SQL and JSON data. SQL++ is expected to be supported by Couchbase's CouchDB and UCI's AsterixDB semistructured databases. Then we expand SQL++ into the *Configurable SQL++*, 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 Configurable SQL++ semantics into the semantics of 10 surveyed languages, hence providing a compact and formal tool to understand the essential semantic differences between different systems. We briefly comment on the utility of formally capturing semantic variations in polystore systems.

Finally we discuss the comparison with prior nested and semistructured query languages (notably OQL and XQuery) and describe a key aspect of query processor implementation: set-oriented semistructured query algebras. In particular, we transfer into the JSON era lessons from the semistructured query processing research of the 90s and 00s and combine them with insights on current JSON databases. Again, the tutorial presents the algebras' fundamentals while it abstracts away modeling differences that are not applicable.

1. INTRODUCTION

This is a summary of the SIGMOD 2016 tutorial. *The reader is referred to* ***http://db.ucsd.edu/TutorialSIGMOD16/*** *for the complete material, which includes the full presented slideware and extended companion papers.*

2. TOPIC AND AUDIENCE OF THIS TUTORIAL

Numerous databases promoted as SQL-on-Hadoop, NewSQL and NoSQL support Big Data applications. These databases generally support the 3Vs.(i) Volume: amount of data (ii) Velocity: speed of data in and out (iii) *Variety*: semi-structured, schemaless and heterogeneous data, which is the focus of this tutorial. Due to the Variety requirement, many databases have adopted semi-structured data models, which are generally slightly different subsets of enriched JSON. The databases provide corresponding query languages.

In addition to these *genuine JSON databases* that utilize variants of JSON as their data model, relational databases also provide special functions and language features for the support of JSON columns, often piggybacking on non-1NF (non first normal form) features that SQL acquired over the years. We refer to SQL databases with JSON support as *SQL/JSON databases*.

In total, the tutorial's evolving companion survey discusses the following as of April 2016 (and potentially more by the time the reader accesses it).

1. Apache Hive [34] (description of Hive largely applicable to Cloudera Impala [11] also)
2. IBM Jaql [7]
3. Apache Pig [26]

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

SIGMOD'16, June 26-July 01, 2016, San Francisco, CA, USA

ISBN 978-1-4503-3531-7/16/06…$15.00

DOI: http://dx.doi.org/10.1145/2882903.2912573

4. Apache Cassandra CQL [22]
5. MongoDB [25]
6. Couchbase's N1QL [13, 9]
7. JSONiq [17]
8. AsterixDB's AQL [6]
9. Google Big Query (aka Dremel [23])
10. Mongo JDBC [35] (a JDBC driver provided by the UnityJDBC middleware for SQL-compliant access to MongoDB)
11. SQL-92 - as the anchor of SQL compliance

Myriads of developers and researchers currently use genuine JSON databases as well as SQL/JSON databases. Database builders and researchers 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. This tutorial provides a deep understanding of the current data models and query languages of genuine JSON and SQL/JSON databases, hence enabling comparisons.

The tutorial does not limit itself to the current status of JSON querying: The database builders and researchers need to draw lessons from the rich body of past research on nested [19, 32, 2], object-oriented [4] and semistructured data models and querying [14, 8, 31], which have been a topic of intense database research: They were first researched in the form of labeled graphs in the mid-90s. Then semistructured data research boomed in the form of XML and its labeled tree abstraction. The current industrial boost to semistructured data emerged primarily from startups, often on the mobile and web space, that utilize Javascript and JSON.[1] Many of the important language design issues of the first eras must be recalled in the new era of semistructured data.

Similarly, when it comes to implementations, we discuss set-at-a-time algebras for semistructured and object-oriented data, in the interest of transferring into the new space time-tested lessons on algebra-based relational query processing as well as lessons from set-at-a-time algebras from past research on nested relational algebra, OQL and XQuery [10, 37, 12, 5, 18, 30, 24, 3, 1, 36, 33, 21, 29, 20, 16, 38].

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

3. CHALLENGES IN COMPREHENDING THE SPACE OF SEMISTRUCTURED DBS

A first challenge is that the evolving query languages have many variations. 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.[2] 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 [19, 32, 2], XQuery, and other XML query languages [31, 14, 8].

Part of the confusion around current genuine JSON 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.S

Similarly, part of the confusion and the large semantics behind SQL/JSON is due to the retrofit of SQL for JSON columns, while certain limitations of SQL (such as the need of a schema) remain in place. Again, the tutorial does not require knowledge of non-1NF, often proprietary, features that have been added to SQL. Rather it only requires textbook SQL-92 language and teaches the non-1NF 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.

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

Step 1: Extending the relational model and SQL for semistructured 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. Therefore, the first step of this tutorial towards explaining the large space of current and past works is the introduction of *Configurable SQL++*. Neglecting temporarily the "configurable" aspect (discussed in Step 2), one may think of SQL++ as a semi-structured query language, based on extending SQL with a minimal number of features.

[1]Their JSON motivation is essentially the same one that the early semistructured research works had: flexible, schemaless data. Douglas Crockford's book "Javascript: The Good Parts" characteristically muses about JSON that "the less we need to agree on in order to interoperate, the more easily we can interoperate".

[2]Most genuine semistructured databases also fall significantly short of full-fledged SQL capabilities, which is not surprising since many commercial JSON databases started as key-value and document-oriented databases.

SQL++ is *backwards compatible* with the "textbook" SQL-92 and mostly backwards compatible with other relevant features of the SQL standard. (The cases where it is not, such as differences in type coercion, lend themselves to interesting discussed comparisons.) Hence SQL++ can be easily understood by SQL programmers and researchers.

The enabler of the relatively easy extension from SQL to genuine JSON database capability is that the semistructured SQL++ data model is a superset of both JSON and the SQL data model. One should think of JSON objects as tuples. Then the SQL++ model can be thought of as expanding JSON with bags (as opposed to having JSON arrays only) and enriched values, i.e., atomic values that are not only numbers and strings (vendors have already made this extension). Vice versa, one may think of SQL++ as expanding SQL with JSON features: arrays, heterogeneity, and the possibility that any value may be an arbitrary composition of the array, bag and tuple constructors, hence enabling arbitrary nested structures, such as arrays of arrays. Consequently, the SQL++ query language inputs and outputs SQL++ data. Notice that there was no such straightforward correspondence between the XML model and relational, primarily due to the lack of explicit tuples in XML.

Then we incorprorate into SQL++ salient features of past full-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). For example, in the spirit of XQuery, JSONiq and OQL, SQL++ is a fully composable and semi-structured language, hence being able to input and output nested and heterogeneous structures. In this tutorial, a new student/researcher of semistructured data, who missed the OQL and XQuery eras, will be able to absorb the essential teachings of 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.

Minimal semantics changes Instead of inventing (or reinventing) multiple syntactic/semantic features for the same fundamental functions (something that, as we show, occassionally happens in SQL/JSON databases), the extension from SQL to SQL++ is most often achieved simply by *removing* semantic restrictions of SQL. Here are a few examples:

1. Unlike SQL's `FROM` clause variables, which bind to tuples only, the FROM clause variables of SQL++ bind to any JSON element.
2. SQL++ is fully composable in the sense that subqueries can appear anywhere, potentially creating nested results when they appear in the `SELECT` clause.
3. While correlation of the subqueries in a `FROM` clause is not allowed in SQL-92, the SQL++ subqueries of a `FROM` clause may be freely correlated with (earlier defined) variables of the same `FROM` clause.
4. The groups created by the `GROUP BY` are directly usable in nested queries - as opposed to SQL's approach where they may only participate in aggregate functions in very limited and particular ways. Interestingly, the SQL++ approach ends up explaining in a simple, formal way even SQL's grouping and aggregation.
5. Unlike SQL, the SQL++ semantics do not require schema or any homogeneity on the input data.

A complete specification of SQL++ can be found in [27]. Notice that features 2 to 4 above are also present in SQL 2003, which includes many 1NF features. In the interest of unifying genuine JSON databases with non-1NF features of SQL, this tutorial explains such features using SQL++, which leads to syntax and semantics succinct and significantly shorter than the SQL 2003 specification. A key methodology that leads to short, succinct semantics is the staging, which reminds of XQuery's specification: First we define an *SQL++ core*. Consequently, additional features and SQL compatibility is achieved as syntactic sugar over the core.

Besides restriction removals, some extensions are also presented. For example, pivoting the tuples' attribute/value pairs to columns is a feature of some databases. Therefore, SQL++ can allow a pair of variables of the `FROM` clause to range over the attribute name and attribute value pairs of input tuples. Similarly, when a `FROM` clause ranges over an array, SQL++ allows a pair of variables to capture the data at an array position and the index of this position. In this way SQL++ seamlessly expands the logic of SQL to "schema" inspection and arrays.

SQL++ shows positive early adoption signs: Couchbase and UCI's AsterixDB are in the process of adopting SQL++.[3] An earlier version of SQL++ has been used by the federated query processor of the UCSD FORWARD application development platform `http://forward.ucsd.edu/sqlpp`.

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

Step 2: Substantial Semantic Differences between Databases However, not all differences are superficial. Furthermore, this tutorial does not suggest that SQL++ (or some close descendant thereof) will become a standard and remove the many variations that are now found in this space. If nothing else, the gap between genuine JSON databases and SQL/JSON is relatively deep as SQL/JSON is very much based on tables and semantics around tables. Furthermore, via communication with genuine JSON database vendors we recognize that a model and query language standard is premature for such a young and fast-evolving area. 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 it pertains to the handling of semi-structured aspects (semantics for missing attributes, heterogeneous types, etc), which are not captured by the SQL backwards compatibility. Towards this goal the tutorial utilizes an extension of SQL++ into Configurable SQL++, which is essentially a guery language generator. Depending on the options that are chosen for various features, different capabilities are assumed and different semantics emerge.

The Configurable SQL++ explains the multiple options that language designers have and the options they have chosen in current systems. The Configurable SQL++ includes *configuration options* that describe

1. which features are supported and

[3]The tutorial's author has collaborated with the AsterixDB and Couchbase teams on specifying SQL++.

2. (for the supported features) different options of language semantics that formally capture the variations of existing database query languages.

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 comparisons will be mostly around the SQL standard, three well known databases (Cassandra CQL, MongoDB, Couchbase N1QL) and one academia-originating system (AsterixDB AQL). Selected features will be explained as particular settings of various configuration options. The reader can find the "configuration options" surveying approach, as well as coverage of additional databases, in the companion documents of this tutorial at **http://db.ucsd.edu/TutorialSIGMOD16/**.

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 Configurable SQL++ 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. To further facilitate understanding of Configurable SQL++ and the effect of the various configuration options, we provide a web-accessible reference implementation of Configurable SQL++ at `http://forward.ucsd.edu/sqlpp`.

Finally, with respect to configurability, we note the potential applicability of the formal capture (by the Configurable SQL++) of capability and semantic differences in the area of polystore querying and query rewriting [15].

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

Step 4: Set-at-a-time Algebras It is desirable that the processing of semistructured queries by genuine JSON databases leverages existing SQL query processing techniques. A distinguishing feature of SQL query processing is the utilization of algebraic plans that consist of set-at-a-time operators. The tutorial shows next that SQL++ query processing can be based on a *SQL++ algebra*, which is a set-at-a-time algebra that extends the relational algebra. The point is particularly relevant to genuine JSON databases, as they evolve from key-value stores to full-blown databases.

The SQL++ algebra is representative of multiple algebras that were proposed and used in the context of semistructured and nested data. It combines ideas that had first appeared in the context of set-at-a-time algebras for XQuery and OQL. Nested relational algebras are also explained as a subset of the SQL++ algebra.

In the interest of capturing multiple algebraic approaches, we present a non-minimal set of algebraic operators, along with algebraic equivalences that explain how certain operators can be replaced by others. The provided equivalences explain how the various approaches connect to each other, while the commentary highlights the tradeoffs.

Open Issues Finally, we emphasize open issues in the expansion from structured to semistructured querying (dealing with lack of typing, potential expansions to include graphs) and briefly discuss interoperability challenges induced by language differences.

5. OTHER DATA

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 XQuery are a plus but not required.

Relevant Aspects of Presenter's Bio 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 12,500 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, eventually acquired in 2003 by BEA Systems.

Acknowledgments The presenter thanks his coauthors of [28] and [27], on which most of this tutorial presentation is based on.

Many of the language-related aspects of the proposed tutorial have been presented by Yannis Papakonstantinou during the past year at talks at the following companies and universities that develop NoSQL databases, SQL-on-Hadoop databases or middleware that operates over them: Amazon, Couchbase, MapR, Informatica (which develops middleware for data warehousing and virtual databases over SQL, NoSQL, Hadoop-on-SQL and NewSQL databases), Pivotal. The above talks led to very useful feedback.

6. REFERENCES

[1] S. Abiteboul and N. Bidoit. Non first normal form relations: An algebra allowing data restructuring. *JCSS*, 33(3):361–393, 1986.

[2] 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.

[3] A. Arion, V. Benzaken, I. Manolescu, Y. Papakonstantinou, and R. Vijay. Algebra-based identification of tree patterns in XQuery. In *FQAS*, pages 13–25, 2006.

[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] C. Beeri and Y. Tzaban. SAL: An algebra for semistructured data and XML. In *WebDB*, 1999.

[6] 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.

[7] 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.

[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] Z. Chen, H. Jagadish, L. Lakshmanan, and S. Paparizos. From tree patterns to generalized tree patterns: On efficient evaluation of XQuery. In *VLDB*, 2003.

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

[12] S. Cluet and G. Moerkotte. Nested queries in object bases. Technical report, 1995.

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

[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] J. Duggan, A. J. Elmore, M. Stonebraker, M. Balazinska, B. Howe, J. Kepner, S. Madden, D. Maier, T. Mattson, and S. B. Zdonik. The bigdawg polystore system. *SIGMOD Record*, 44(2):11–16, 2015.

[16] T. Fiebig, S. Helmer, C. Kanne, G. Moerkotte, J. Neumann, R. Schiele, and T. Westmann. Anatomy of a native XML base management system. *VLDB J*, 11(4), 2002.

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

[18] D. Florescu, C. Hillery, D. Kossmann, P. Lucas, F. Riccardi, T. Westmann, M. Carey, and A. Sundararajan. The BEA streaming XQuery processor. *VLDB Journal*, 13, 2004.

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

[20] H. V. Jagadish, S. Al-Khalifa, A. Chapman, L. V. S. Lakshmanan, A. Nierman, S. Paparizos, J. M. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, and C. Yu. TIMBER: A native XML database. *VLDB Journal*, 11(4):274–291, 2002.

[21] H. V. Jagadish, L. V. S. Lakshmanan, D. Srivastava, and K. Thompson. TAX: A tree algebra for XML. In *DBPL*, 2001.

[22] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *Operating Systems Review*, 44(2):35–40, 2010.

[23] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. Dremel: Interactive analysis of web-scale datasets. *PVLDB*, 3(1):330–339, 2010.

[24] P. Michiels, G. A. Mihaila, and J. Siméon. Put a Tree Pattern in Your Algebra. In *ICDE*, 2007.

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

[26] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig Latin: a not-so-foreign language for data processing. In *SIGMOD Conference*, pages 1099–1110, 2008.

[27] K. W. Ong, Y. Papakonstantinou, and R. Vernoux. SQL++: An expressiveness benchmark of SQL-on-Hadoop, NoSQL and NewSQL databases, 2015. http://db.ucsd.edu/wp-content/uploads/pdfs/375.pdf.

[28] K. W. Ong, Y. Papakonstantinou, and R. Vernoux. The SQL++ query language: Configurable, unifying and semi-structured. *CoRR*, abs/1405.3631, 2015. http://arxiv.org/abs/1405.3631.

[29] Y. Papakonstantinou, V. R. Borkar, M. Orgiyan, K. Stathatos, L. Suta, V. Vassalos, and P. Velikhov. XML queries and algebra in the Enosys integration platform. *Data Knowl. Eng.*, 44(3):299–322, 2003.

[30] C. Re, J. Siméon, and M. Fernández. A complete and efficient algebraic compiler for XQuery. In *ICDE*, 2006.

[31] 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/.

[32] 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.

[33] C. Sartiani and A. Albano. Yet another query algebra for XML data. In *IDEAS*, 2002.

[34] 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.

[35] JDBC driver for MongoDB. http://www.unityjdbc.com/mongojdbc/mongosqltranslate.php.

[36] S. D. . Viglas, L. Galanis, D. J. DeWitt, D. Maier, and J. F. Naughton. Putting XML query algebras into context. Available at: http://www.cs.wisc.edu/niagara/Publications.html.

[37] X. Zhang, B. Pielech, and E. Rundensteiner. Honey, I shrunk the XQuery!: An XML algebra optimization approach. In *WIDM*, 2002.

[38] X. Zhang and E. Rundensteiner. XAT: XML algebra for Rainbow system. Technical Report. WPI-CS-TR-02-24, July 2002.