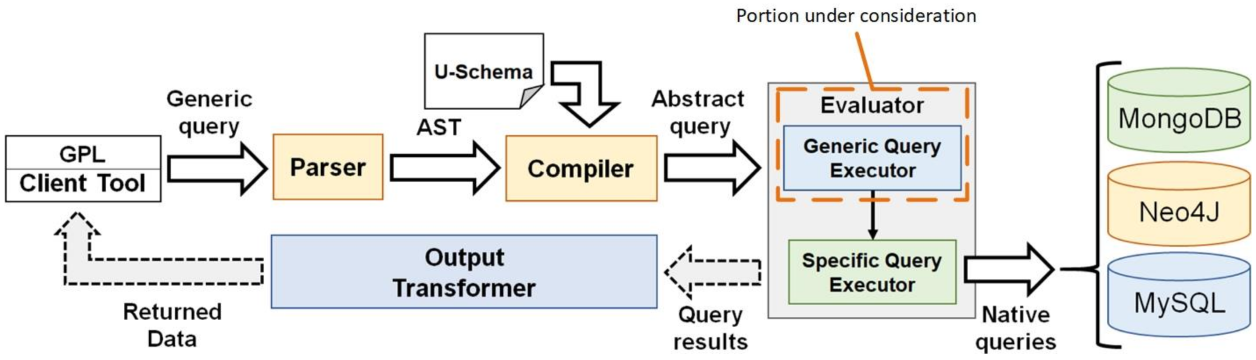
Introduction

At this point, the proliferation of NoSQL databases has been well observed and remarked upon in literature. A common point is that NoSQL systems offer a variety of data models, have no agreed upon standards, and even systems with an equivalent model may support different features or query functions and syntax. [1], [12] These various databases are well suited to meeting application needs where traditional relational databases falter. However, it has also been pointed out that the storage of data in heterogeneous formats and systems can cause difficulties for developers. [2], [4], [5], [13] This is especially true because NoSQL databases often do not require or enforce adherence to a formal schema, while relational databases do, and there can be benefits to either approach. [1], [12] Pursuing a unified system model where heterogeneous data stores can be accessed uniformly is a worthwhile goal that can lead to database utilities such as visualizers, migration tools, standard query languages, and schema enforcers/data standardizers. [5], [12] This project will take a step in that direction, building on the work Candel et al. in [5].

Proposal

In [5], the authors present a unified meta-model, U-Schema, able to represent logical schemas for relational, column store, document store, key-value, and graph databases; U-Schema is also able to represent structural variations among NoSQL entities. They define mappings between each data paradigm and U-Schema and offer examples of the types of tools and benefits that could be realized through the use and implementation of U-Schema.

One of the presented applications of the U-Schema is for a generic query language that can uniformly access databases that operate in the covered paradigms. They model their proposed language as an extension of SQL and give some examples queries. They also provide an overview of the proposed architecture for processing the generic queries, reproduced below with an added annotation.



Generic query architecture based on U-Schema, taken from (TODO: cite)

This project will focus on the "Evaluator" portion of the architecture, particularly on the "Generic Query Executor" piece. The big, motivating idea behind the project is "**How does the generic query executor make decisions on which specific queries to execute, and how can it make *good* decisions?**" The portions related to language definition, parsing, compilation, and translation into specific queries will not be considered, nor will the transformation of query results to output (although this may have some impact on how the evaluator makes decisions).

This project will seek to develop strategies and techniques to answer three general types of question from the perspective of the query evaluator:

1. When the same data is present in multiple databases, which one should I access it from?
2. When data for the same U-Schema entity is spread out between multiple databases, how do I tell where to search to satisfy a query for that entity?
3. How do I join different data from different databases together?

In an even more general manner, these questions can be summarized as "How do I know which data is where, and what the best way to access it is?" This points to the need for the U-Schema query engine to maintain its own meta-dictionary, analogous to the specific data dictionaries of the underlying databases, which tracks what information is stored where in what format and how to access it.

So, a primary goal of this project will be the exploration and definition of a meta-dictionary for a U-Schema based system. This dictionary should support making decisions that can answer the types of question presented above. In addition, we are interested in the evaluator making "good" decisions, which in this case means evaluating queries in an efficient manner (in terms of number of computational, data access, and data sending operations). If there are multiple ways to process a query, we would like the evaluator to choose the most efficient option. This may involve figuring out how to interface with built-in cost-estimation features and query plan evaluators already present in each underlying database. This process may also be aided through the use of meta-indices at the U-Schema abstraction level.

The final implementation will work with some "hardcoded" example queries that fall under the umbrella of the types of question listed above. It will use a meta-dictionary to decide on possible ways to answer these queries, and a rudimentary cost-estimation mechanism to help measure which way might be the most efficient. Ideally, one or two indexing data structures will also be implemented on the U-Schema meta-level, and they will demonstrably improve query efficiency from a cost-estimation standpoint. Finally, all of this will require coming up with a set of test data, whether that is procured from other sources or generated. This data will preferably be represented in at least three different types of data paradigm (document, relational, and graph) and interdependent. Data for some entities should be spread out across databases and some data should be duplicated between databases.

Background

As noted above, this project expands on the work already done by Candel et al. in [5]. However, their work is just one of many that proposes a unified metamodeling approach to handle data that is stored under different paradigms; other examples include [2], [4], [8] and [14]. The U-Schema proposed by Candel et al. seems to be one of the most cohesive, complete options, and as such seems like the best work on which to base this project; the authors review many other similar proposals such as [2], [3], and [13], but give convincing arguments in favor of their work in each case. It's especially beneficial that they propose the generic schema architecture.

An interesting aspect of the proposal in [5] is that it relies on native queries executed against specific database instances, as opposed to grouping all data together in one unified store under the auspices of U-Schema. While some authors, such as in [9], have advocated for that approach, and some NoSQL systems such as OrientDB[1] use this, I think there are still some advantages to the architecture as presented in [5]. Mainly, it is already somewhat common for a single application to make use of multiple databases of different paradigms, as microservice architecture (and the corresponding "database per service" pattern[2]) is popular. So, a U-Schema based universal query engine could likely be deployed on top of existing applications, offering the benefits of a unified query interface without having to do a full refactor for the sake of using a new database. In addition, if it made more sense for an individual component to interact with a specific database, as opposed to the U-Schema meta-level, it could still do so unimpeded.

[5] is not the first work that proposes a universal platform for querying data from heterogeneous databases. A very notable example is the SOS System proposed and implemented by Atzeni et al. in [2]. This system supports uniform access between a document store (MongoDB), a column store (HBase), and a key-value store (Redis). However, the functionality is limited to "put", "get", and "delete" operations, and their model essentially reduces each database to a key-value store. The SOS Platform does not seem to offer nearly the power or flexibility of U-Schema. Similar work [4] is also very limited, as it can involve manual interaction with the users to select which data store to pull data from, and supports only simple SELECT-style queries.

Some work in [8] and [14] seemed promising at first, despite the fact that I had reservations about the publishers themselves. Each of these papers purports to offer a unified query framework for relational and NoSQL databases, complete with inter-database joins and optimizations at the meta-level (i.e. before specific queries are executed against specific databases). Both papers depended heavily on the Apache Calcite project, which may itself prove to be a fruitful area of exploration for the purposes of optimization and tracking meta-data for my project. However, both [8] and [14] rely on the user submitting native SQL queries, which Calcite parses and then uses to determine what native databases contain the queried values. This process is very vague on how exactly Calcite, operating under a relational model, communicates with databases that use different paradigms; the authors offer no sort of meta-model. As such, it is hard to envision these as complete solutions.

Finally, query optimization has, of course, been well studied already. In the relational context, [6] provides a good summary of techniques, and sources such as [7] go into greater depth. For the most part, the key is that relational algebra affords the capability for some operations to be reordered or nested to achieve greater efficiency, with or without the presence of indices. For NoSQL systems, [1] discussed how coming up with an appropriate domain-specific data model can speed up query processing, and online sources such as [10] and [11] discuss other query optimization techniques. It will be important to keep all of this in mind as I work on developing a cost estimator for U-Schema generic queries and attempt to implement techniques that increase query efficiency.

References

[1] Atzeni, P., Bugiotto, F., Cabibbo, L., & Torlone, R. (2020). Data modeling in the NoSQL world. Computer Standards & Interfaces, 67, 103149. https://doi.org/10.1016/j.csi.2016.10.003

[1] https://orientdb.org/
[2] https://microservices.io/patterns/data/database-per-service.html

[2] Atzeni, P., Bugiotti, F., Rossi, L. (2012). Uniform Access to Non-relational Database Systems: The SOS Platform. In: Ralyté, J., Franch, X., Brinkkemper, S., Wrycza, S. (eds) Advanced Information Systems Engineering. CAiSE 2012. Lecture Notes in Computer Science, vol 7328. Springer, Berlin, Heidelberg. https://doi-org.proxybz.lib.montana.edu/10.1007/978-3-642-31095-9_11

[3] Atzeni P., Gianforme G., Cappellari P. (2009) A Universal Metamodel and Its Dictionary. In: Hameurlain A., Küng J., Wagner R. (eds) Transactions on Large-Scale Data- and Knowledge-Centered Systems I. Lecture Notes in Computer Science, vol 5740. Springer, Berlin, Heidelberg. https://doi-org.proxybz.lib.montana.edu/10.1007/978-3-642-03722-1_2

[4] Ágnes Vathy-Fogarassy, Tamás Hugyák. (2017) Uniform data access platform for SQL and NoSQL database systems. Information Systems, 69, p. 93-105. https://doi.org/10.1016/j.is.2017.04.002.

[5] Candel, C. J., Sevilla Ruiz, D., & García-Molina, J. J. (2022). A unified metamodel for NoSQL and relational databases. Information Systems, 104, 101898. https://doi.org/10.1016/j.is.2021.101898

[6] Chaudhuri, S. (1998). An overview of query optimization in Relational Systems. Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems - PODS '98. https://doi.org/10.1145/275487.275492

[7] Garcia-Molina, H., Ullman, J. D., & Widom, J. (2009). Query Execution. In Database systems: The complete book (pp. 701–758). essay, Pearson Prentice Hall.

[8] Karanjekar, J. B., & Chandak, M. B. (2017). Uniform query framework for relational and NoSQL databases. *Computer Modeling in Engineering & Sciences, 113*(2), 177-187. http://dx.doi.org/10.3970/cmes.2017.113.177

[9] Liu Z.H., Lu J., Gawlick D., Helskyaho H., Pogossiants G., Wu Z. (2019) Multi-model Database Management Systems - A Look Forward. In: Gadepally V., Mattson T., Stonebraker M., Wang F., Luo G., Teodoro G. (eds) Heterogeneous Data Management, Polystores, and Analytics for Healthcare. DMAH 2018, Poly 2018. Lecture Notes in Computer Science, vol 11470. Springer, Cham. https://doi.org/10.1007/978-3-030-14177-6_2

[10] Murthy, K. (2019, May 8). Approaches to query optimization in NoSQL - DZone database. DZone. Retrieved April 4, 2022, from https://dzone.com/articles/approaches-to-query-optimization-in-nosql-1

[11] Murthy, K. (2016, Dec. 16). A Deep Dive Into Couchbase N1QL Query Optimization - DZone database. DZone. Retrieved April 4, 2022, from https://dzone.com/articles/a-deep-dive-into-couchbase-n1ql-query-optimization

[12] Sevilla Ruiz D., Morales S.F., García Molina J. (2015) Inferring Versioned Schemas from NoSQL Databases and Its Applications. In: Johannesson P., Lee M., Liddle S., Opdahl A., Pastor López Ó. (eds) Conceptual Modeling. ER 2015. Lecture Notes in Computer Science, vol 9381. Springer, Cham. https://doi-org.proxybz.lib.montana.edu/10.1007/978-3-319-25264-3_35

[13] Wang, A. (2016, February 28). Unified Data Modeling for relational and NoSQL databases. InfoQ. Retrieved January 28, 2022, from https://www.infoq.com/articles/unified-data-modeling-for-relational-and-nosql-databases/

[14] Zhang, H., Zhang, C., Hu, R., Liu, X., & Dai, D. (2021). Unified SQL query middleware for heterogeneous databases. *Journal of Physics: Conference Series, 1873*(1) doi:http://dx.doi.org/10.1088/1742-6596/1873/1/012065