Name:	James P Fairbanks
Affiliation:	Georgia Tech Research Institute
Email:	james.fairbanks@gtri.gatech.edu
Presentation type (s):	**Software Demo, Poster, Lightning Talk**

QueryGarden: growing healthy applications in well prepared SQL.

James P. Fairbanks, PhD
Georgia Tech Research Institute, Atlanta, GA

Abstract

Many software packages within the OHDSI ecosystem rely on SQL query generation, which is fraught with security risks and compatibility issues. We introduce an ahead of time query server, QueryGarden, to enable access to queries over HTTP to reduce the risk of SQL injection attacks, improve the lifecycle management of analytical queries and provide both batch processing for training analytics models as well as single patient queries for near real time applications. The Query Garden is a place to plant queries for analytics that grow into stable, secure, and user friendly applications.

Introduction

The OHDSI software ecosystems uses queries as the primary method for storing business logic of health data analysis. ETL into the OMOP CDN [1] is performed primarily by ingesting data from CSV files into a set of staging tables that are then transformed by issuing SQL queries to load the data into production tables. ATLAS [2] uses a human friendly web editor for generating SQL queries as cohort definitions. Achilles uses a collection of SQL queries to determine the data quality of an OMOP database, while the PatientLevelPrediction R package uses SQL queries to define the features that will be extracted for making personalized predictions for patients. All these tools have in common the need for SQL query generation that is beyond the power of standard query parameterization. Data driven applications need to leverage databases in order to guarantee availability and persistence of data. The OHDSI ecosystem has wisely standardized on a set of SQL relational database (RDBMS) systems to use when constructing these applications. However, the management of these queries is not sufficient to meet the diverse and complex needs of the OHDSI community.

Most software applications include a database application layer for storing and retrieving data and executing queries. For these applications the needs of Create, Read, Update, Delete (CRUD) are sufficient. Since the servers and clients written for these applications would prefer a database independent programming interface, they typically use an object-relational mapping (ORM [3]) which translates between the schema of the database and the objects of programming language used to manipulate these databases. The primary benefit of these ORM tools is the cross-platform, database independence and easy of use gained by programming against the native interface of objects or structures in the programming language. The ORM library handles the conversion of application code into SQL statements to perform the CRUD tasks. The primary drawback to these tools is their limitations with respect to the analytical capabilities of the SQL languages. Not all ORM tools provide access to the entirety of the SQL language for a particular RDBMS. So when performing analytical queries, the application must handle all of the complexity of raw SQL embedded in their application, which is precisely what they were trying to avoid with ORM tools in the first place. The OHDSI applications such as PatientLevelPrediction require the power of raw SQL queries in order to efficiently compute the patient features needed for the application. Most ORM tools will not be

sufficient for the analytical applications in the OHDSI ecosystem.

One reason that SQL queries are hard to manage for the OHDSI community is that the OMOP CDM has a large degree of variability. There are multiple versions of the CDM with variable table and schema names. These properties along with heterogeneity in the CDM prevent the use of standard SQL parameters for most queries. SQL query parameters are designed to maximize security and prevent the injection of malicious SQL as parameters passed from user facing code. This security first design means that the structure of the query cannot be changed as a parameter and thus many of the changes required by the OMOP CDM cannot be performed. OHDSI/SQLRender tool handles this problem using a runtime manipulation of strings with a custom DSL for generating queries. However, this approach leads to security vulnerabilities and inefficient execution.

The Query Garden approach combines query generation with SQL query parameters to avoid SQL injection vulnerabilities by generating the queries as a resource that is deployed with the application and accessed at run-time. These queries are specified using a standard templating language such as Jinja, Mustaches or the Go standard text templates and filled in at deployment time with all of the structure modifying changes that cannot be done with SQL query parameters. The queries can contain query parameters that are substituted at run-time. The variable values are specified in a YAML [4] document that associates with each key a string value that will be replaced in the query. This YAML document includes scopes which are collections of variables that are used when instantiating a template. This design allows for reuse of templates across many queries, which reduces maintenance costs. The queries are also stored as templates and scopes as files on disk which enables them to be tracked in version control. By tracking the scopes and templates separately, the administrator of a Query Garden based service can easily verify changes to the queries for security risks. Before deploying an application that uses the queries, the scopes and templates are combined to form the rendered queries. These rendered queries are also saved to disk for manual inspection. At this point they are plain SQL queries (with query parameters) that can be analyzed using any existing tool for static or dynamic analysis of SQL queries, for example query plans can be created using the EXPLAIN or ANALYZE keywords in an RDBMS GUI client. The ability to profile SQL queries outside of the application that uses them is important for benchmarking and optimizing the queries. Once the queries have been rendered, they can be deployed as static resources with the application that uses them. This relieves the application from depending on any particular method for generating queries, which enhances interoperability as any tool that exports queries can be used with any application that requires them. For example clinical experts can construct queries using ATLAS which can be deployed along with queries exported by a computational phenotyping tool.

The Query Garden approach is designed to meet the needs of an application for near real time patient level prediction. The models need to be trained using queries that operate on cohorts and then deployed using queries that operate on patients. This requires many queries, one per feature of the feature matrix as well as a scalable system for managing these queries. Since the cohort level and patient level queries need to share many parameters, the templates and scopes are shared between them. Clinical experts can enter new queries by writing templates or scopes to add new features. Development and maintenance of this application are greatly aided by a simple system for statically generating the queries.

Conclusion

In conclusion, the Query Garden approach for statically rendered queries enables secure, interoperable deployment of analytical queries. This approach is complementary to an ORM based approach for building applications that depend on business logic in the form of simple database queries. Separating the construction of the queries from their integration into applications allows for better debugging, profiling, and auditing of the queries and can improve the performance and security of many OHDSI applications.

References

1. OHDSI/CommonDataModel contributors. OMOP CDM specification https://github.com/OHDSI/CommonDataModel/
2. OHDSI contributors. OHDSI list of analytics tools https://www.ohdsi.org/analytic-tools/
3. Maria E Orlowska, Hui Li, Chengfei Liu. On Integration of Relational and Object-Oriented Database Systems SOFSEM 1997: SOFSEM'97: Theory and Practice of Informatics pp 295-312
4. Oren Ben-Kiki, Clark Evans, Ingy döt Net, The YAML Specification, http://yaml.org/spec/ 2009