Querios
John Diggins, Jacob Levine, Henry O’Connor, Joren Carlson

RESTful API for executing complex, AI-driven queries on hybrid columnar- and row-format data stores

Motivation

At DeepCure, we are constantly generating data, computationally and experimentally, and storing them in our databases. The data types can range from simple integers to complex array structures. The stored data can be used to train machine learning models, and/or generate insights for decision making in the drug discovery process. Currently, the various databases sit behind disparate APIs, leading to the segregation of data analysis and preparation routines into multiple separate queries and ETL (extract-transform-load) workflows.

Goal

Have a web application with a RESTful API that can receive, parse and execute complex queries on the DeepCure data warehouse, and return the results. The queries can range from simple SQL to arbitrary UDFs (user-defined functions).

Milestones

%Environment setup and tool familiarization
- Make sure Python is installed with pip (package manager) working.
- Follow tutorial(s) to become familiar with python, flask, and vuejs

%Having a system-level diagram of the end application

%Develop Unit Testing & Code Review policy

%Design minimal query scheme
- Form a standard for how input queries will be formed

%Working minimal alpha query parser
- break complex queries into individual queries
- Create basic tree out of queries to represent dependencies

%Working minimal alpha query name resolver / string builder
- Resolve plain text strings with proper names (prepare query to be integrated with database APIs)

%Working minimal alpha query controller
- The minimal working query controller takes a query tree from the parser and controls which queries are executed and when. It will check the tree as results are returned for root results, meaning the query is complete and ready to be formatted for return to the user.
%Working minimal alpha return data formatter

%Working minimal alpha query executor

%Working frontend alpha version

- no fancy UI
- input for the query
- display results

%Deliver minimal complete backend

- Connect all pieces of the back end

%Integrations with each underlying database

%Deliver final product

- Release minimum viable product

**Deliverables**

For the MVP we will build the following:

- Flask microservices front end
  - Input in the form of SQL syntax query string.
  - Outputs simple table of queried data.
- Query Parser
  - Brakes down input query into tree object
- Query Controller
  - Will fire off the querying process.
  - Determines when we are done querying.
- Query String Builder
  - Takes tree node containing string and returns set of input variables.
  - Fills these variables into an appropriately formatted string.
  - Sends to executor.
- Query Executor
  - Connects to db. Returns error if failure.
  - Executes the query on the db.
  - Returns the results as a pandas df to the controller.
- Format user return
  - Formats resulting df into table for return.