GA4GH Discovery Search Update

December 2, 2019

What has changed in the last six months?

Use cases

Use cases encompass diverse needs, including multimodal queries, access levels, fuzzy matching, aggregation, data types

Harmful variants of high quality in genes related to phenotype of subject

Variant Analysis

VICC
Variant Interpretation
for Cancer Consortium

Genomics datasets from subjects with phenotype

Data Discovery

EUROPEAN
GENOME-PHENOME
ARCHIVE

ENA
European Nucleotide Archive

Existence of a specific variant of interest

Variant Discovery
(Beacon)

elixir

Other patients with similar phenotype and genotype

Patient Discovery
(Matchmaker)

Matchmaker
Exchange

Subjects with clinical indication, measure, and genetic feature

Clinical Trial
Recruitment

AUTISM
SPEAKS®

What Are We Trying to Accomplish?

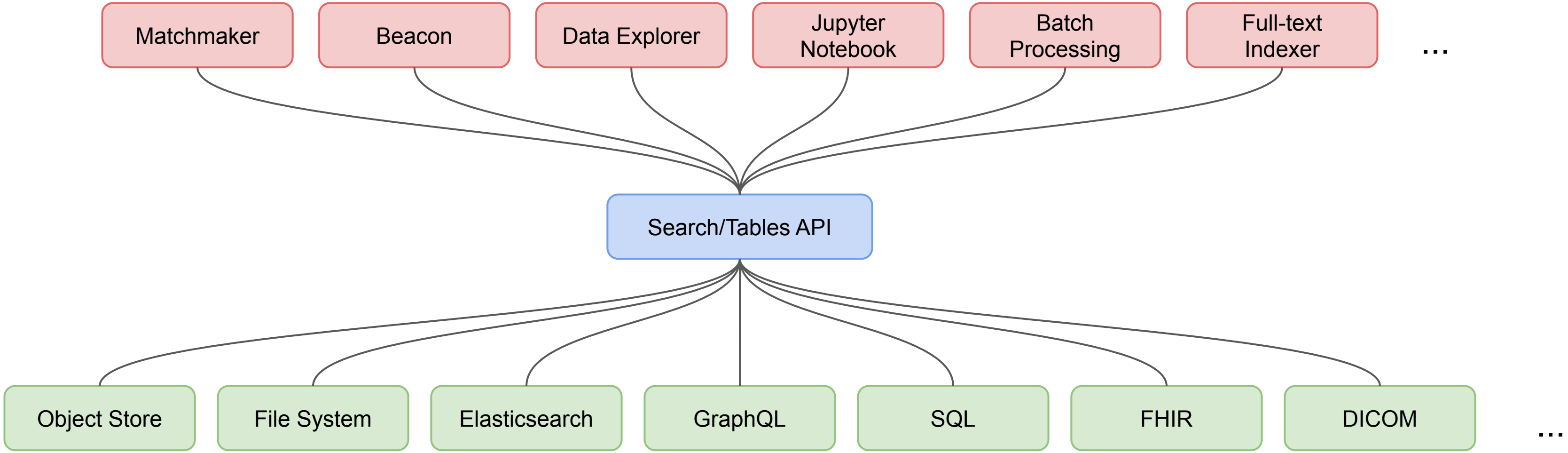
- Enable instant access to data,
    - Respecting patient consent and privacy regulations
    - Allowing discovery of new data sets
    - Without presupposing the use case
    - In an environment where tools work together seamlessly

Not all of these things necessarily belong in a Search standard,
but it’s important to keep them in mind.

Overview

Instead of making N × M connections from applications to data sources, implement N Search/Tables clients and M Search/Tables connectors.

$$N + M < N \times M$$



Search/Tables REST Resources

Get list of tables	GET /tables
Get structural info about a table	GET /table/*{table_name}*/info
Get a first page of data in a table	GET /table/*{table_name}*/data
Next page via HATEOAS (just an example; server can pick any path and client follows a relative link)	GET /table/*{table_name}*_pages/3
Search (execute SQL Query)	POST /search

GET /tables

```
{
  "tables": [
    {
      "name": "subjects",
      "description": "Subjects of the Personal Genome Project Canada",
      "data_model": {
        "$ref": "https://personalgenomes.ca/schema/subject.json"
      }
    },
    ...
  ]
}
```

GET /table/*subjects*/info

```
{
  "name": "subjects",
  "description": "Subjects of the Personal Genome Project Canada",
  "data_model": {
    "$ref": "https://personalgenomes.ca/schema/subject.json"
  }
}
```

GET /table/*subjects*/data

```
{
  "data_model": {
    "$ref": "https://personalgenomes.ca/schema/subject.json"
  },
  "data": [
    {"id": "PGPC-1", "birth_date": "1973-07", "sex": "F"},
    {"id": "PGPC-10", "birth_date": "1963-11", "blood_type": "A+", "sex": "M"},
    ...
  ],
  "pagination": {
    "next_page_url": "subjects_2"
  }
}
```

GET https://personalgenomes.ca/schema/subject.json

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "https://personalgenomes.ca/schema/subject.json",
  "type": "object",
  "required": [ "id" ],
  "properties": {
    "id": { "$ref": "https://schemablocks.org/schemas/ga4gh/v0.x.y/individual.json#properties/id" },
    "sex": { "$ref": "https://schemablocks.org/schemas/ga4gh/v0.x.y/individual.json#properties/sex" },
    "birth_date": {
      "$ref": "https://personalgenomes.ca/schema/BirthDate.json",
      "description": "Birth year and month in ISO 8601 format (yyyy-mm)",
      "pattern": "^[0-9]{4}-(0[1-9]|1[0-2])$",
    },
    "blood_type": {
      "$id": "https://personalgenomes.ca/schema/BloodType.json",
      "$schema": "http://json-schema.org/draft-07/schema#",
      "description": "ABO Blood Type with Rhesus factor",
      "enum": [ "A+", "A-", "AB+", "AB-", "B+", "B-", "O+", "O-" ],
      "type": "string"
    }
  }
}
```

JSON Schema Draft 7

Note that Draft 2019-09 is out and we will likely adopt it. See their changelog for details.

Example Data Record (PGPC Subject)

```
{
  "id": "PGPC-56",
  "sex": "PATO:0020001",
  "birth_date": "1970-04",
  "blood_type": "O+"
}
```

Tables API Summary

- Represents data with rich attribute metadata
- Enables data interchange
- Provides the preconditions for useful federation across datasets:
  - Uniform representation
  - A language for describing and identifying attributes
- Works as a wire format and a storage format
  - In both cases, gzip can keep the data size minimal
- Supports pagination
- Complements use-case specific standards without presupposing a use case itself

How to get started with Search

Get Familiar With Tables and JSON Schema

Static Tables

- Try describing a few datasets you know well using JSON Schema
- Use the open source tables-api-cli to convert the data
- Upload to a webserver or cloud bucket
- That’s a Tables implementation!

Dynamic Tables

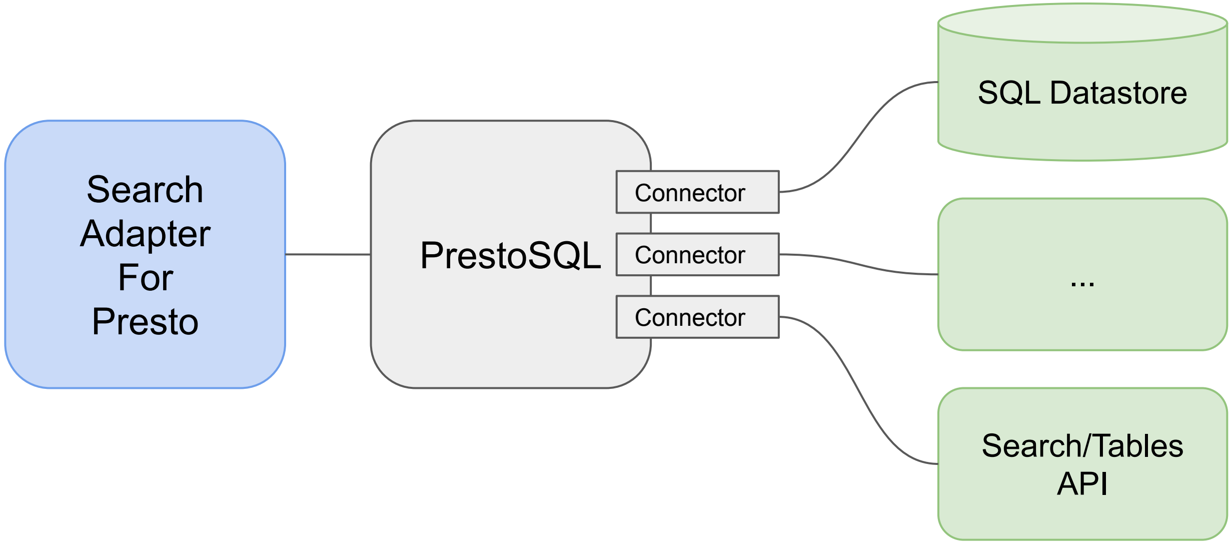
- Create the schema as above
- Serve the data using a web service that translates to JSON on the fly

Get Familiar With Presto SQL

- Open source (ASL 2.0)
- Follow the getting started instructions on their website
- Connect to a local datastore
  - Recommend a SQL datastore first, eg. PostgreSQL or MySQL
  - Also try a non-SQL datasource like ElasticSearch or MongoDB
- Start on your own machine
- Try setting up a cluster

Reasonable paths to having early search support on live data sets in ~6 months

GA4GH Search Adapter - Presto SQL - Anything



Interop Across Data Sets

- Start at the attribute level
- Focus on codifying readily identifiable attributes such as Subject ID, Sample ID, Sex
- Lean on existing efforts: in the JSON Schema, refer to identifiers from SchemaBlocks, HPO, Phenopackets, even FHIR and CDISC
- Recommendation: publish the JSON schemas separately from the Tables and refer to them with $ref

Access and Authorization

- We recommend Search/Tables API implementations require OAuth 2 bearer tokens
- For internal implementations, tokens can be issued by the local authorization server
- For federated implementations, we recommend users obtain OAuth tokens from the target system using GA4GH Passports

How to engage the Discovery Search workstream

Contact Info

- Biweekly calls - everyone welcome
  - Next call is Wednesday, December 11 at 11:00 am Toronto Time
  - Contact Rishi Nag to get added to the calendar event
- GitHub Repository - create/comment on issues and pull requests
- Either way, please let us know what’s working and what isn’t

Unexplored Future Directions

- Full-text search has the potential to provide a lot of discovery value
    - Over metadata values
        - column names, descriptions
    - Over data values themselves
    - With fuzzy term matching
        - Term matching could be enhanced by ontologies
    - Could be built on top of Tables API but might be worth including in the specification
- Make provenance explicit in the Table data structure
    - Useful for audits and reproducibility