Gen3 Data Modeling

Sheepdog& Peregrine

Herding Data Submissions

& Hunting Down Data

Thursday, May 9, 2019
1:00 PM-2:00 PM (CST)

Gen3 Data Modeling

Herding Data Submissions & Hunting Down Data

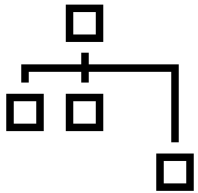
Chris Meyer, Ph.D.

Center for Translational Data Science,
University of Chicago

May 9, 2019

Outline

1. What is a Data Model?



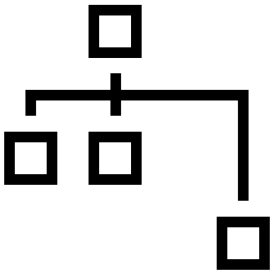
1. Structure of a Gen3 Data Model

1. Herding Data Submissions: Data Import and Export

2. Hunting Down Data: Querying and Filtering Data

1. Demonstration of Query, Export, and Import in Workspace

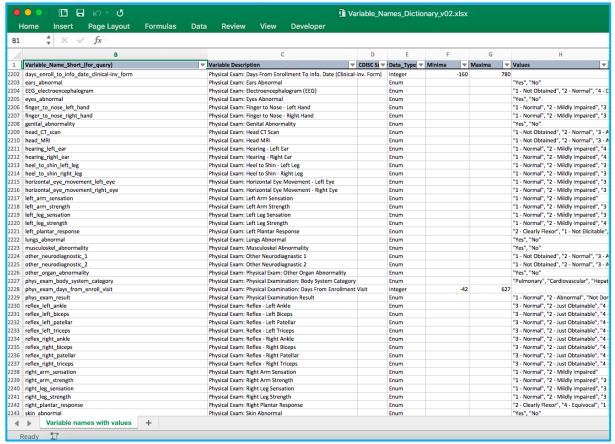
What is a Data Model?



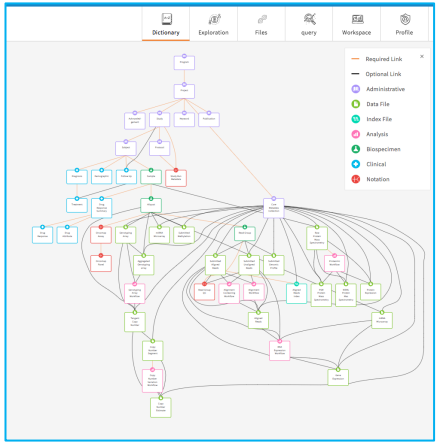
What is a Data Model?

- A ***data model*** organizes terms in a data dictionary and defines how they relate to one another. It is the implementation of a data dictionary and enables Gen3 services to **submit** and **query** data.

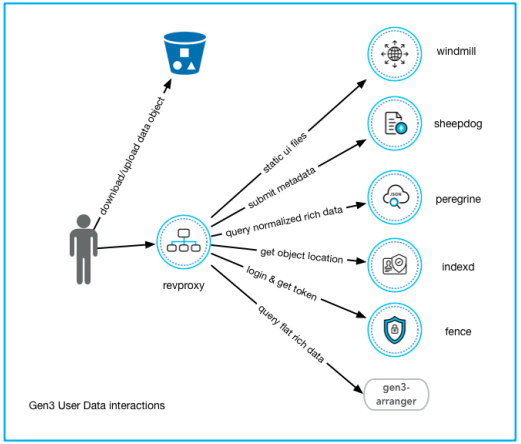
Data Dictionary



Data Model

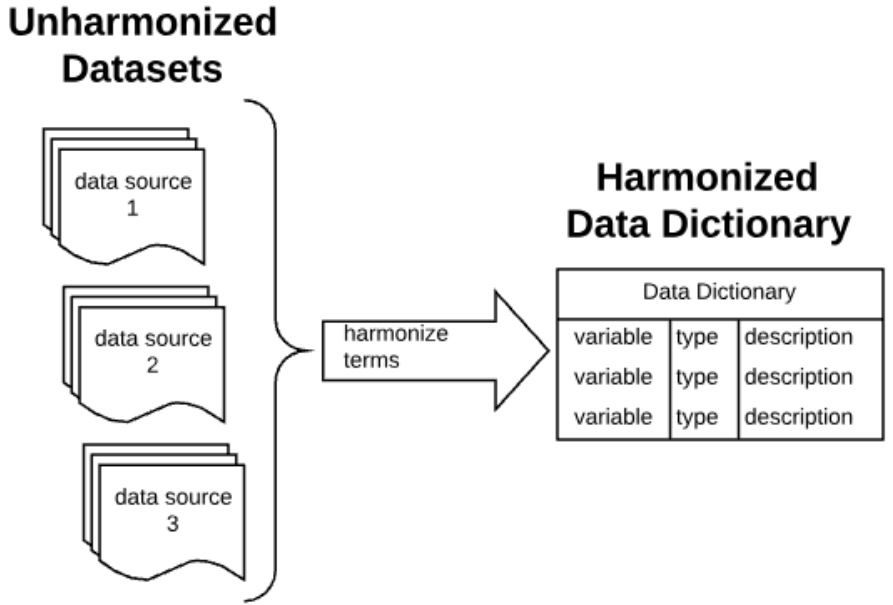


Gen3 Services



What is the Data Dictionary?

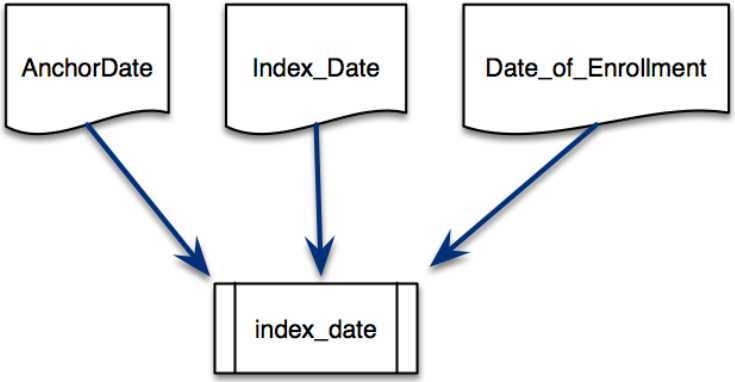
- The ***data dictionary*** defines and describes how research datasets are represented in the database and harmonizes term definitions from different data sources
- ***Data harmonization*** is foundational to the *data commons* concept of sharing data for cross-project analyses.



What is the Data Dictionary?

GEN3 Data Commons

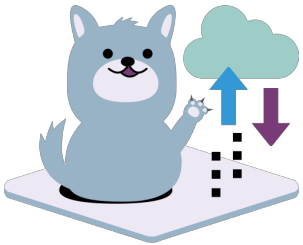
- Dictionaries get everyone on the same page:
  - Define nodes and properties used across different but similar projects in a process called *data harmonization*.
  - Help avoid inconsistencies in data reporting and use across projects.
  - Make data easier to find, subset and analyze by enforcing Data Standards.
  - Support mapping terms to external controlled vocabularies like the NCIt, the National Cancer Institute's Thesaurus.



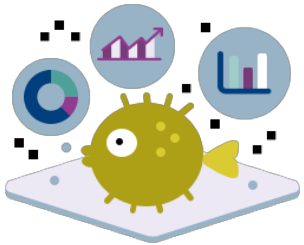
Example: Different studies have their own unique term for "the date a participant enrolled in a clinical trial". Those terms are *harmonized*, or mapped a single term, in the data dictionary.

What is the Data Model?

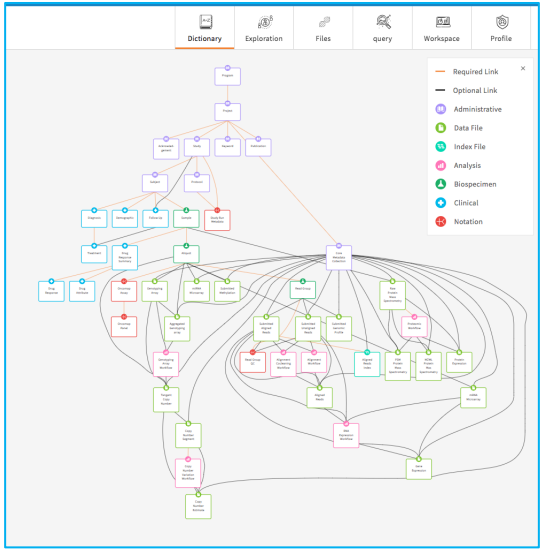
- The data model enables Gen3 services to **import**, **export**, and **query** data.
  - **Data import and export** is accomplished by the *Sheepdog* service, which checks submissions against the data model to ensure all required fields are present and have appropriate values.



  - **Database queries** are facilitated by the *Peregrine*, *Arranger*, and *Guppy* services. Queries must conform to the data model for successful data retrieval.



Structure of a Gen3 Data Model



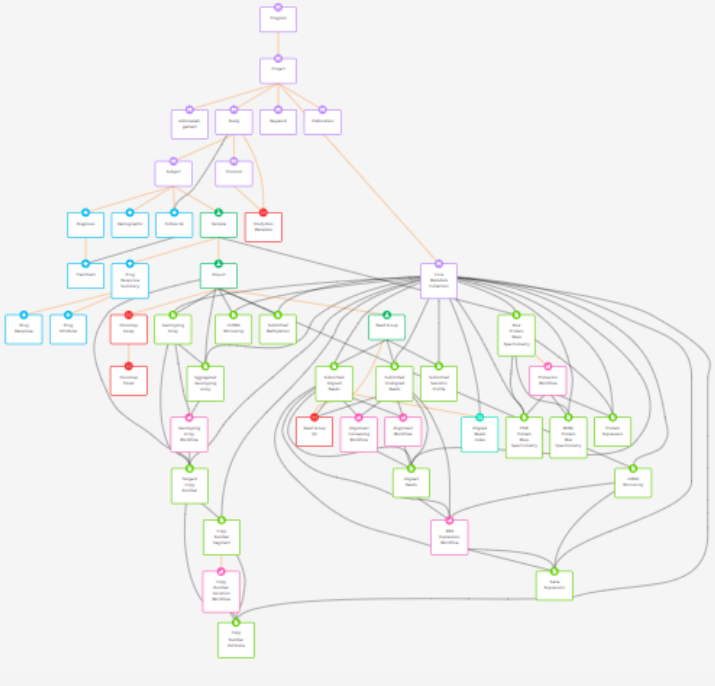
Structure of the Gen3 Data Model

- The Gen3 Data Model is a graph-like relational model consisting of interrelated ***nodes*** that store certain related ***properties.***

administrative

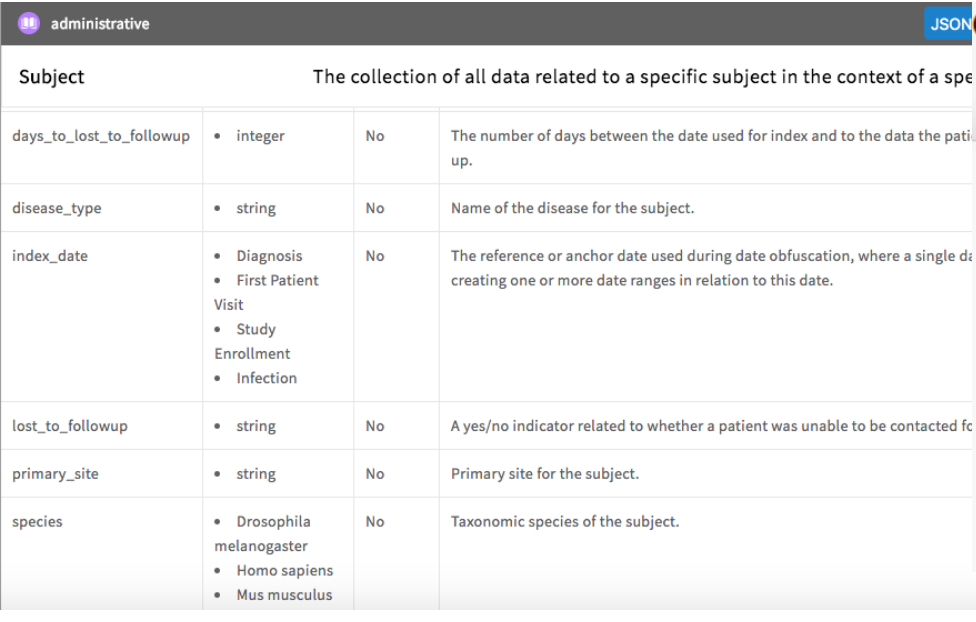
Subject — The collection of all data related to a specific subject in the context of a specific study.

Property	Type	Required	Description
type	• string	★Required	No Description
submitter_id	• string • null	★Required	No Description
studies	• array • object	★Required	No Description
days_to_lost_to_followup	• integer	No	The number of days between the date used for index and to the data the patient was lost to follow-up.
disease_type	• string	No	Name of the disease for the subject.
index_date	• Diagnosis • First Patient Visit • Study Enrollment • Infection	No	The reference or anchor date used during date obfuscation, where a single date is obscurred by creating one or more date ranges in r
lost_to_followup	• string	No	A yes/no indicator related to whether a patient was unable to be contacted for follow-up.
primary_site	• string	No	Primary site for the subject.
species	• Drosophila melanogaster • Homo sapiens • Mus musculus • Mustela putorius furo • Rattus rattus • Sus scrofa • Canis Domesticus	No	Taxonomic species of the subject.
strain	• string	No	A lower-level taxonomic rank used in microbiology or virology, plants and rodents, usually at the intraspecific level (within a species)
tissue_source_site_code	• string	No	A clinical site that collects and provides patient samples and clinical metadata for research use.



Structure of the Gen3 Data Model

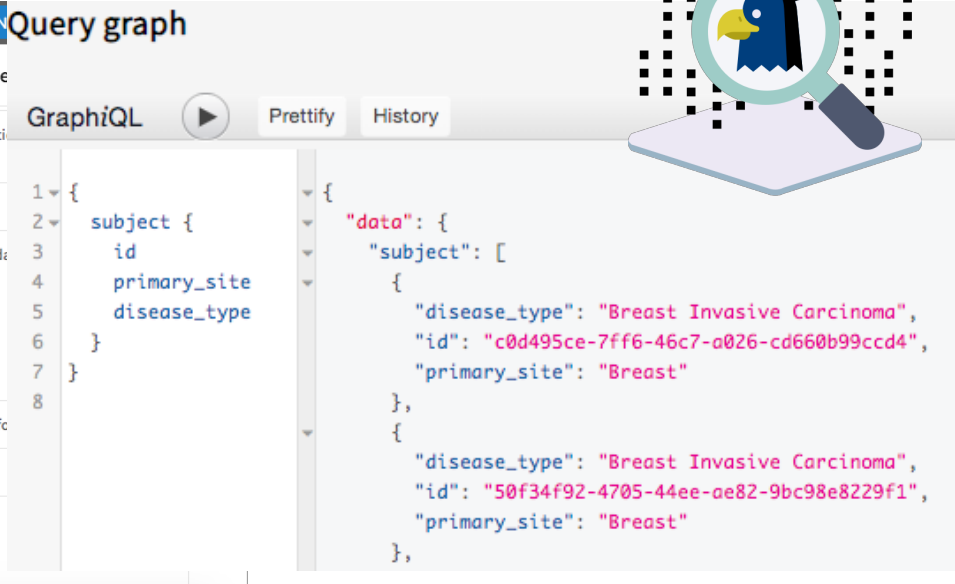
- Structured Data are imported and exported as key-value pairs by Sheepdog.
- The data element keys are termed ***properties*** in Gen3***.***
- Property values can be queried using GraphQL, which is accomplished via the Peregrine, Arranger, or Guppy services.
- Sets of values in a node are called ***records*** or ***entities***, which are assigned unique IDs (UUIDs).



administrative JSON

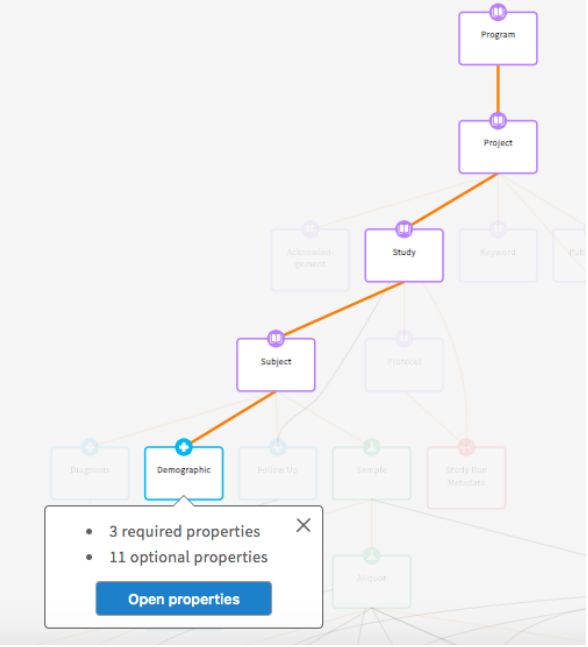
Subject — The collection of all data related to a specific subject in the context of a spe

days_to_lost_to_followup	• integer	No	The number of days between the date used for index and to the data the pati… up.
disease_type	• string	No	Name of the disease for the subject.
index_date	• Diagnosis • First Patient Visit • Study Enrollment • Infection	No	The reference or anchor date used during date obfuscation, where a single da… creating one or more date ranges in relation to this date.
lost_to_followup	• string	No	A yes/no indicator related to whether a patient was unable to be contacted fo…
primary_site	• string	No	Primary site for the subject.
species	• Drosophila melanogaster • Homo sapiens • Mus musculus	No	Taxonomic species of the subject.



Structure of the Gen3 Data Model

- Properties are organized into ***nodes***, which are categories of structured data.
- Each node must have a relationship to at least one other node.
- The root node is *program* and must have the *project* node as its child.



clinical JSON TSV Close

Demographic	Data for the characterization of the patient by means of segementing the population (e.g.,characterization by age, sex, or race).

Data File

Property	Type	Required	Description	Term
type	• string	★ Required	No Description	
submitter_id	• string • null	★ Required	No Description	
subjects	• array • object	★ Required	No Description	
age_range	• string	No	Range of ages for the subject. The age range should not include ages over 89 years.	
breed	• American cocker spaniel	No	A stock of animals or plants within a species having a distinctive appearance and typically having been developed by deliberate selection.	

Structure of the Gen3 Data Model

GEN3 Data Commons

- The data model is a JSON created from node schemas in the YAML format.
- Each node is defined in a single schema.
- The schema contains the following:
  - A node ***id*** used for data query/submission.
  - A ***category*** used to group nodes conceptually.
  - A ***description*** which describes the node's contents
  - List of ***links*** defining relationship to other nodes.
  - List of ***required*** properties.
  - List of ***properties***.

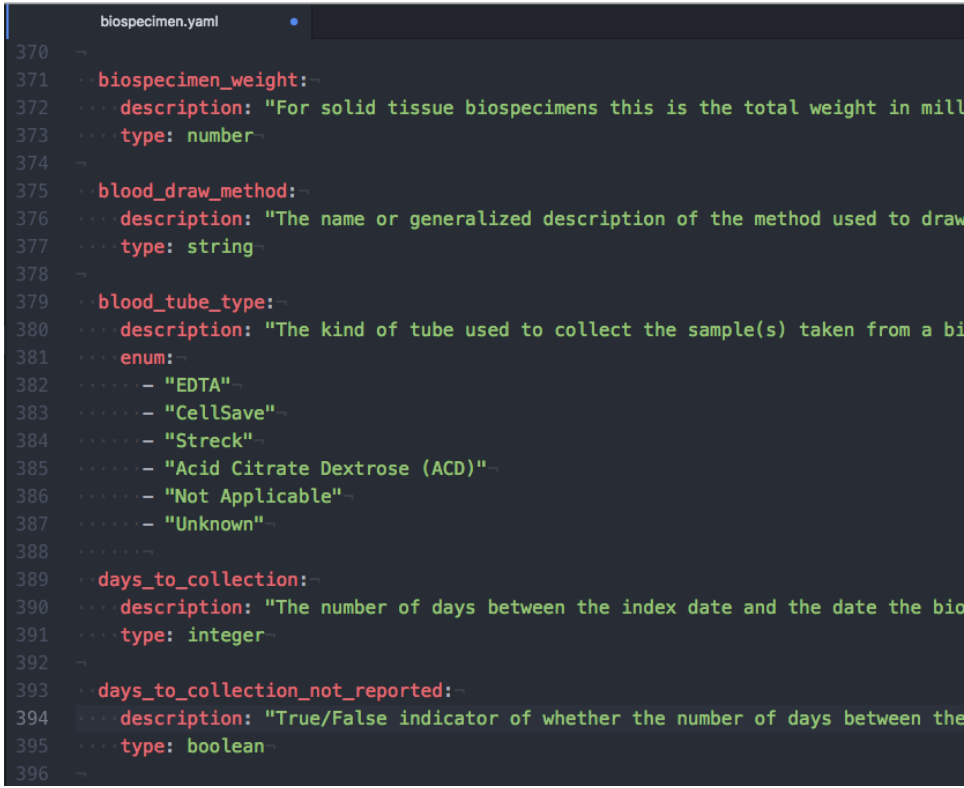
demographic.yaml

```yaml
$schema: "http://json-schema.org/draft-04/schema#"

id: "demographic"
title: Demographic
type: object
namespace: https://nci-crdc-demo.datacommons.io/
category: clinical
program: '*'
project: '*'
description: >
  Data for the characterization of the patient by means of
additionalProperties: false
submittable: true
validators: null

systemProperties:
  - id
  - project_id
  - state
  - created_datetime
  - updated_datetime

links:
  - name: subjects
    backref: demographics
    label: describes
    target_type: subject
    multiplicity: one_to_one
    required: true
```

Structure of the Gen3 Data Model

GEN3 Data Commons

- Property definitions include:
  - **property name** (e.g., "*blood_tube_type*:")
  - ***description***
  - ***type***
    - **string**
    - **enum** (enumerated values)
    - **integer** (whole numbers)
    - **number** (floats / numbers w decimel)
    - **boolean** (True/False)
    - **array** (a list of strings)



Structure of the Gen3 Data Model

- Limitations can be put on acceptable property values:
  - **Minimum/maximum** for integers/numbers.
  - **Enumerations** are limited strings.
  - Strings can be required to match patterns**.**
- Submitted records that do not conform fail.

demographic.yaml

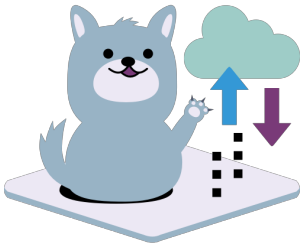
```yaml
cause_of_death:
  description: >
    Text term to identify the cause of patient death with respect to cancer.
  enum:
    - "Cancer Related"
    - "Not Cancer Related"
    - "Unknown"

days_to_birth:
  description: >
    The number of days between the index date and the date of patient birth.
    If the number of days is greater than 32872 (89 years), then please use
    'days_to_birth_gt89'.
  type: integer
  maximum: 32872
  minimum: 0

days_to_birth_gt89:
  description: >
    Indicate if the number of days between the index date and the date of
    patient birth is greater than 32872 (89 years).
  enum:
    - "Yes"
    - "No"
```

_definitions.yaml

```yaml
id: _definitions

UUID:
    term:
        $ref: "_terms.yaml#/UUID"
    type: string
    pattern: "^[a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{12}$"
```

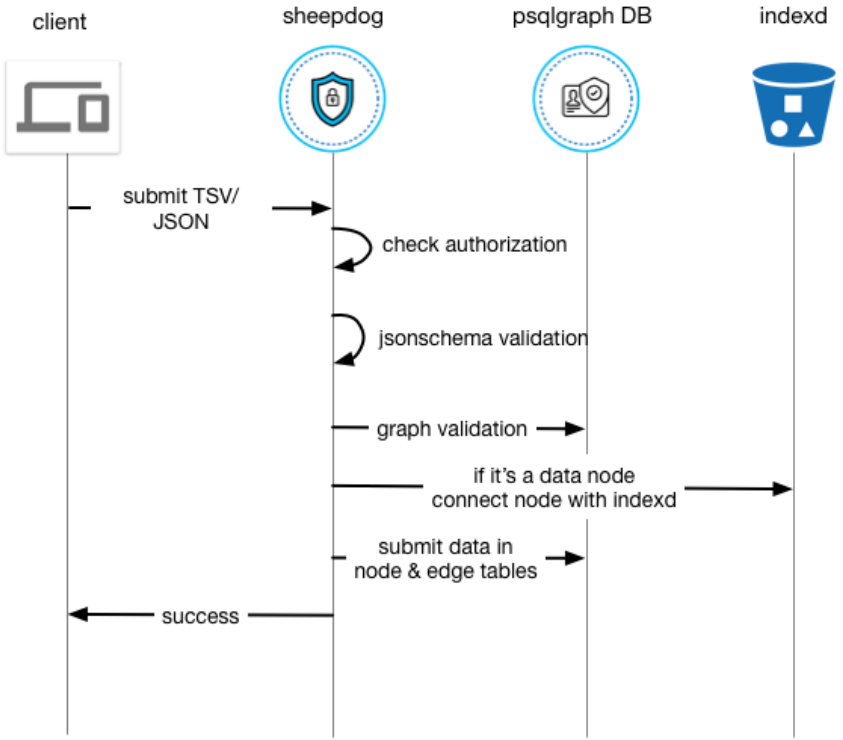
Herding Data Submissions

The Submission Service

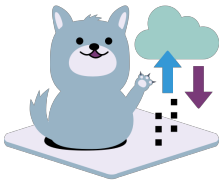


Herding Data Submissions

- The Sheepdog service shepherds submissions of structured data into the graph database.
- Sheepdog checks validity of each record in a data upload against the data dictionary to ensure all required fields are present and have appropriate data values.
- Sheepdog also supports export of structured data records in TSV or JSON formats.



Gen3 rich data submission

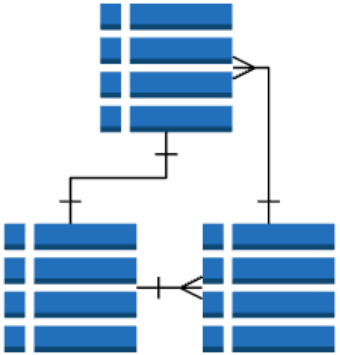


Herding Data Submissions: Types of Data

GEN3 Data Commons

- ***Data files*** must be downloaded to view its content, which is not accessible via API queries. Examples are images, tabulated data spreadsheets, or DNA sequencing reads.

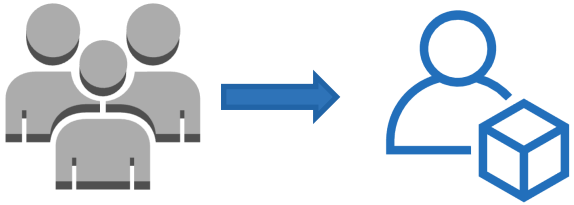
- ***Structured data*** **(**AKA ***metadata***) consists of records containing variable key-value pairs, which can be queried and modified via the API or viewed in Gen3 data exploration tools.



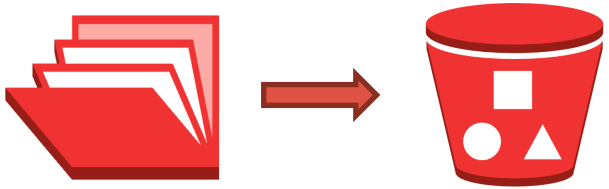
Herding Data Submissions: Steps for Data Import

GEN3 Data Commons

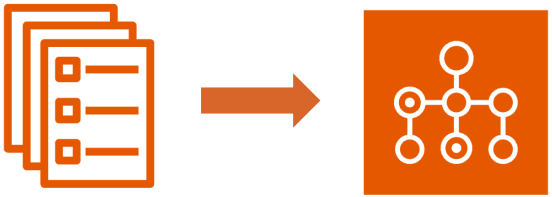
1. User Authorization



1. Data File Upload:
    a. Prepare Project in Submission Portal
    b. Upload Data Files to Object Storage
    c. Map Uploaded Files to a Data File Node



1. Structured Data Submission:
    a. Submit Structured Data
    b. Link Data File Records to their structured data



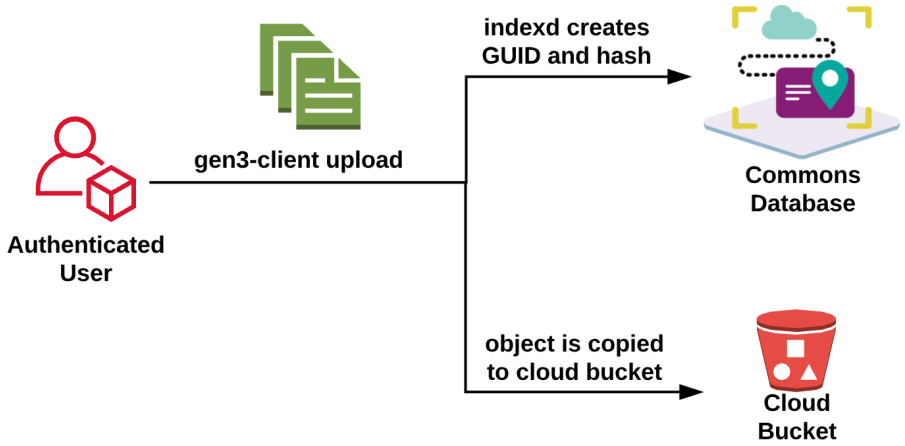
File Upload: the Data Lake

GEN3 Data Commons

- The ***Linked Data Lake*** paradigm:
  - Data files are uploaded to object storage (AWS s3 bucket). Users don’t see bucket contents.
  - Indexd assigns a unique identifier called ***GUID*** to each file. Users access files via GUIDs.
  - Files in the ***data lake*** are ***linked*** to structured data using ***GUIDs***

File Upload: Use the ‘gen3-client’ to Upload Files

- The ***gen3-client*** is a command-line tool for uploading and downloading data files
  - The client is configured with your credentials and sends files to an s3 bucket
  - A unique GUID is minted for each file
  - Indexd creates records linking the s3 locations of files with data_file records in the data model



File Upload: Steps to Upload Files

GEN3 Data Commons

1. Configure the gen3-client with Credentials Downloaded from Windmill
    - `**gen3-client configure --profile=**profile_name **--apiendpoint=**https://nci-crdc-demo.datacommons.io/ **--cred=**~/Downloads/credentials.json`

```
~/Documents/Notes/DCF/demo> gen3-client configure --profile=dcf_demo --apiendpoint=https://nci-crdc-demo.datacommons.io/ --cred=~/Downloads/dcf-credentials.json
2019/04/03 13:23:50 Local failed log file "/Users/christopher/.gen3/logs/dcf_demo_message_log_20190403132350CDT.log" has opened
2019/04/03 13:23:50 Local succeeded log file "/Users/christopher/.gen3/logs/dcf_demo_succeeded_log.json" has opened
2019/04/03 13:23:50 Local failed log file "/Users/christopher/.gen3/logs/dcf_demo_failed_log_20190403132350CDT.json" has opened

Begin parsing all file paths for "~/Downloads/dcf-credentials.json"
Finish parsing all file paths for "~/Downloads/dcf-credentials.json"
```

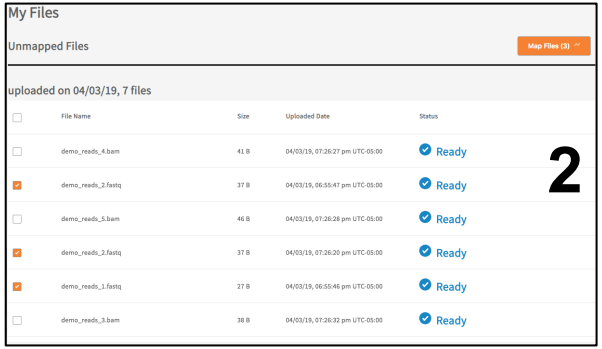
2. Upload files using the profile by passing the client a file location / RegEx
    - `gen3-client upload **--profile=**profile_name **--upload-path=**path/to/file.txt`

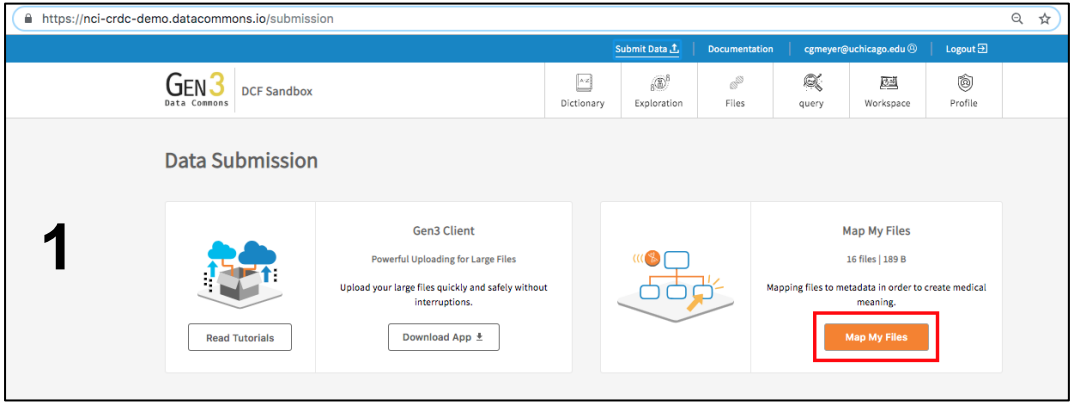
```
~/Documents/Notes/DCF/demo> gen3-client upload --profile=dcf --upload-path=demo_reads_1.fastq
2019/04/03 14:07:11 Local failed log file "/Users/christopher/.gen3/logs/dcf_message_log_20190403140711CDT.log" has opened
2019/04/03 14:07:11 Local succeeded log file "/Users/christopher/.gen3/logs/dcf_succeeded_log.json" has opened
2019/04/03 14:07:11 Local failed log file "/Users/christopher/.gen3/logs/dcf_failed_log_20190403140711CDT.json" has opened

Begin parsing all file paths for "demo_reads_1.fastq"
Finish parsing all file paths for "demo_reads_1.fastq"

The following file(s) has been found in path "demo_reads_1.fastq" and will be uploaded:
        demo_reads_1.fastq
```

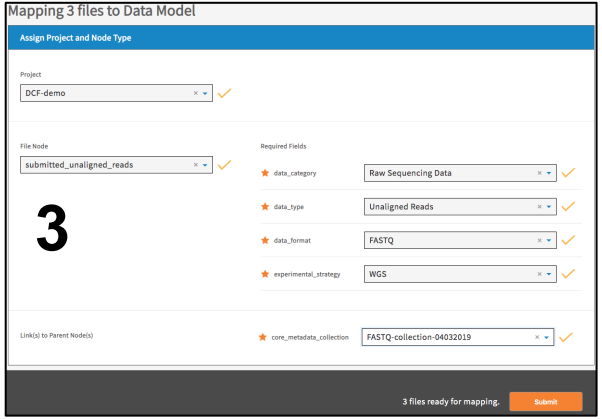
File Upload: Map the Files to the Data Model

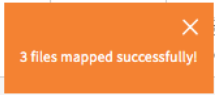
3. The final step in File Upload is **mapping the files to a node** in the model

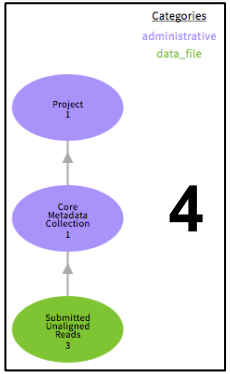
    1. Click “Map my Files” in Windmill.
    2. Choose files via checkbox to map to a particular node.
    3. Assign values to required properties for the files.
    4. Sheepdog creates the structured data records.











Structured Data Upload: Overview

GEN3 Data Commons

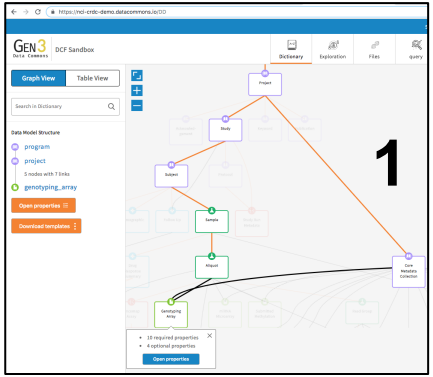
- Now that files are uploaded and mapped to the data model, the rest of the project's ***structured data*** must be submitted and linked to the data file records.
    - Sheepdog ensures ***structured data*** conform to the data model, and values can be queried via Peregrine
    - ***Data files*** on the other hand must be downloaded from object storage to view contents/values.
- Structured Data is submitted node-by-node.
- Typically data is submitted in TSV files (also accept JSON format).
- Sheepdog services checks submissions against the data model and creates one record for each row in a TSV (or entity in a JSON).
- Records are updated if a row has a previously created ***submitter_id*** or ***UUID***

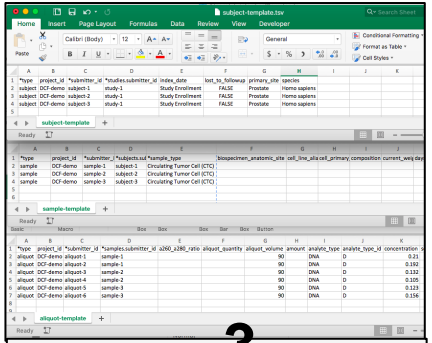
Structured Data Upload: TSV Submission Process

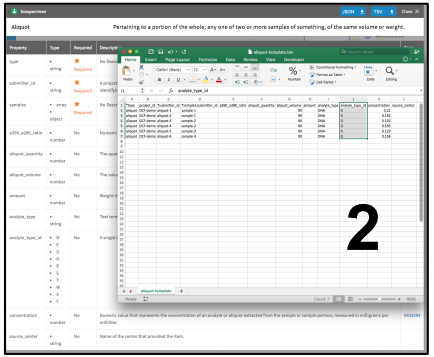
GEN3 Data Commons

- TSV Submission Process

1. Download a template TSV for each desired node in your project.
2. Populate template TSVs with structured data.
3. Submit TSVs in the proper order (top-down, starting with the root node and moving towards “leaf” nodes).
4. Update links in data file TSV to link files to their corresponding, upstream structured data.
5. Sheepdog updates the records.

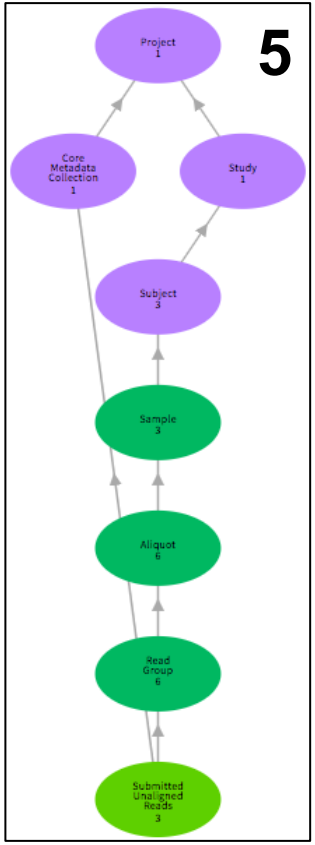






	A	B	C	D
1	type	submitter_id	core_metadata_collections.submitter_id	read_groups.submitter_id
2	submitted_unaligned_reads	DCF-demo_demo_reads_1_e23c	FASTQ-collection-04032019	read-group-1
3	submitted_unaligned_reads	DCF-demo_demo_reads_2_a2a1	FASTQ-collection-04032019	read-group-2
4	submitted_unaligned_reads	DCF-demo_demo_reads_2_6eb7	FASTQ-collection-04032019	read-group-3

4



Structured Data: TSV Submission Troubleshooting

GEN3 Data Commons

- During TSV submission, Sheepdog checks each entity (row in TSV) against the data dictionary.
  - TSVs are submitted in “chunks” of 30 records / rows
  - If any entity / row in the TSV is invalid with respect to the data model, the chunk will fail

Submitting chunk 1 of 1

Failed: 400
Submitted chunk 1 of 1

Errors:

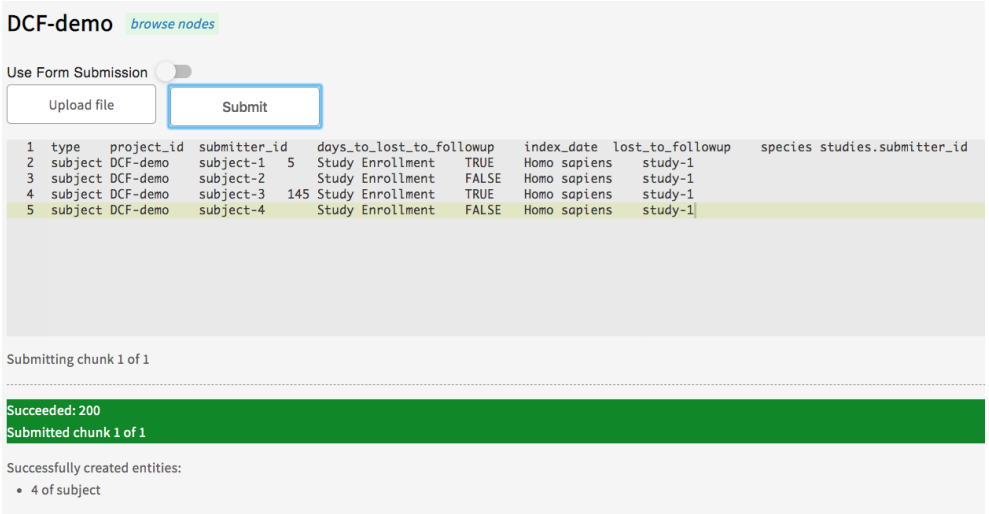

```
[
    {
        "action": null,
        "errors": [
            {
                "keys": [
                    "days_to_lost_to_followup"
                ],
                "message": "'five' is not of type 'integer'",
                "type": "INVALID_VALUE"
            },
            {
                "keys": [
                    "species"
                ],
                "message": "'Homo sapien' is not one of ['Drosophila melanogaster', 'Homo sapiens',
                "type": "ERROR"
            }
        ],
```


	A	B	C	D	E	F	G	H
1	type	project_id	submitter_id	days_to_lost_to_followup	index_date	lost_to_followup	species	studies.submitter_id
2	subject	DCF-demo	subject-1	five	Study Enrollment	TRUE	Homo sapien	study-1
3	subject	DCF-demo	subject-2		5/9/19	FALSE	Homo sapiens	study-1
4	subject	DCF-demo	subject-3	145	Study Enrollment	Yes	Homo sapiens	study-1
5	subject	DCF-demo	subject-4		Study Enrollment	FALSE	Homo sapiens	study-one

Structured Data: TSV Submission Troubleshooting

GEN3 Data Commons

- After fixing the errors, the submission is successful and records are created or updated by Sheepdog.
    - If an existing submitter_id or id is submitted, the record is updated instead of created.
    - If data changes, the values are overwritten.

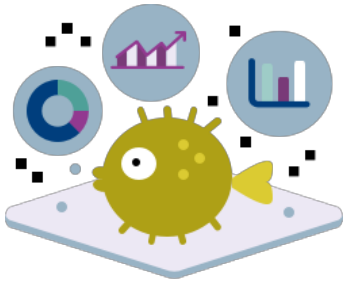


	A	B	C	D	E	F	G	H
1	type	project_id	submitter_id	days_to_lost_to_followup	index_date	lost_to_followup	species	studies.submitter_id
2	subject	DCF-demo	subject-1	5	Study Enrollment	TRUE	Homo sapiens	study-1
3	subject	DCF-demo	subject-2		Study Enrollment	FALSE	Homo sapiens	study-1
4	subject	DCF-demo	subject-3	145	Study Enrollment	TRUE	Homo sapiens	study-1
5	subject	DCF-demo	subject-4		Study Enrollment	FALSE	Homo sapiens	study-1

Hunting Down Data

Querying and Filtering Data



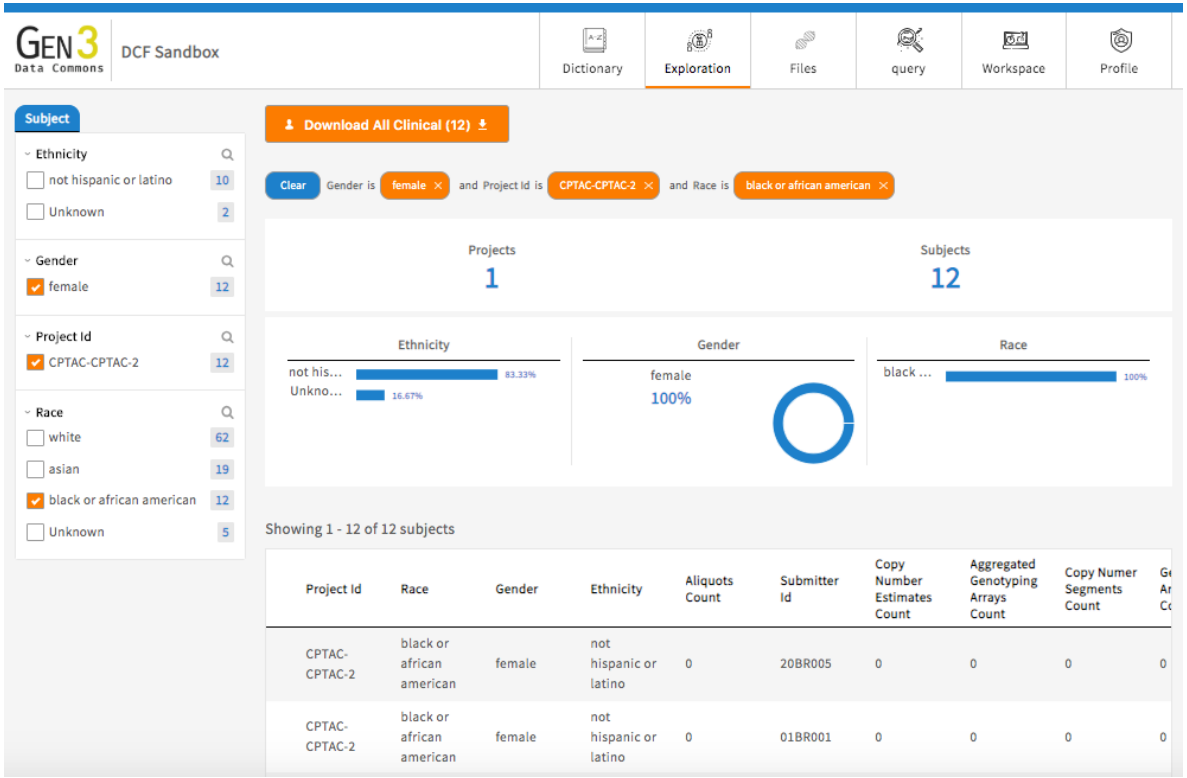



Windmill's Exploration Page

a graphical user interface for cohort selection

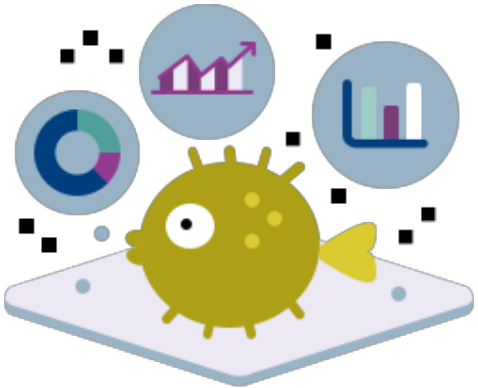
Windmill's Exploration Page

- Cohorts can be selected via a graphical user interface using data facets.
- Once a cohort is selected, a file download manifest can be sent to your Workspace / JupyterHub for easy data file access and analysis



API Queries for Cohort Building

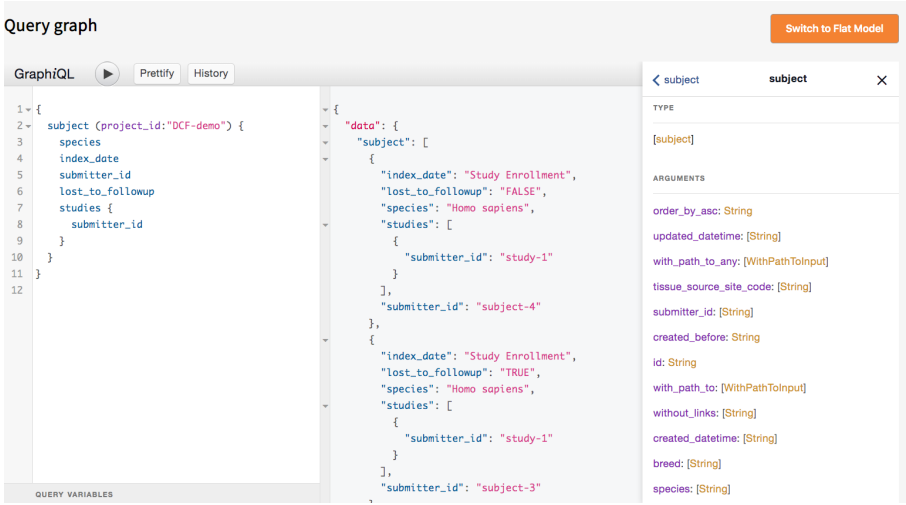
Peregrine, Arranger, and Guppy



GraphiQL Query Building in Windmill

GEN3 Data Commons

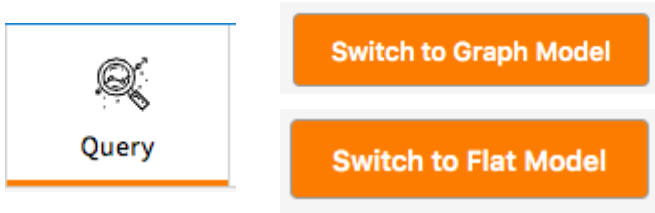
- The GraphiQL interactive query building interface makes queries more intuitive for both Flat and Graph models
  - Built-in documentation
  - Autocomplete for objects, fields, arguments
  - Ability to pass variables



Windmill's Query Page

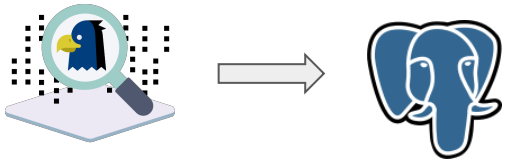
GEN3 Data Commons

- Switch between the Flat and Graph models on Windmill's "Query" Page.



- These use different endpoints that query different databases:
  - Graph Model hits the PostgreSQL DB

    vpodc.org/api/v0/submission/graphql/



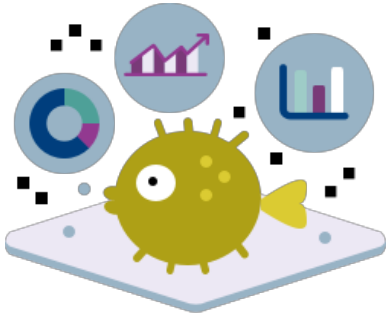
  - Flat Model hits the ElasticSearch DB

    vpodc.org/api/v0/flat-search/search/graphql

Gen3 Query Overview

- Graph Model
  - **Peregrine** searches the **PostgreSQL** (graph database).
  - **Peregrine** translates GraphQL query to SQL.

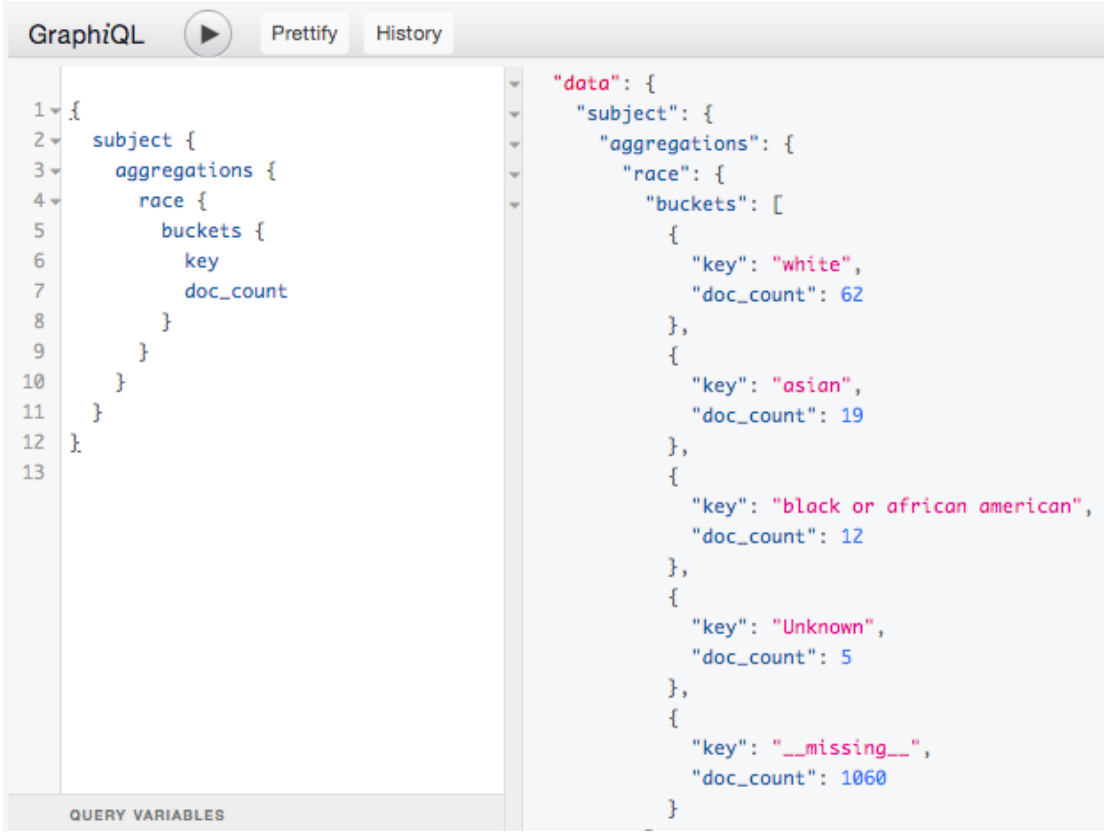
- Flat Model
  - **Arranger/Guppy** searches the **ElasticSearch DB.**
  - Arranger translates GraphQL to ElasticSearch query.
  - ES queries support Aggregations.
  - **Guppy** facilitates easier GraphQL-like queries of ElasticSearch DB.



Flat Model: Aggregation Query

Flat Model queries support *Aggregations* for *string* and *numeric* fields:

- For strings:
  - bin counts - the number of records that have each **key**.
- For numeric fields:
  - summary statistics - minimum, maximum, average, count and sum.



The GraphQL Endpoints

GEN3 Data Commons

- Queries can be sent to both flat and graph API endpoints programmatically.

```
In [22]: project_id = 'DCF-demo'
    ...: node = 'subject'
    ...:
    ...: props = ['index_date','species','submitter_id']
    ...: properties = ' '.join(map(str,props))
    ...:
    ...: query_txt = """query Test { %s (first:0, project_id: "%s") {%s}} """ % (node, project_id, properties)
    ...: query = {'query': query_txt}
    ...:
    ...: graphql_endpoint = api + 'api/v0/submission/graphql/'
    ...: resp = requests.post(graphql_endpoint, json=query, auth=auth).text # Get id from submitter_id
    ...: data = json.loads(resp)
    ...:
    ...: data
    ...:
Out[22]:
{'data': {'subject': [{'index_date': 'Study Enrollment',
    'species': 'Homo sapiens',
    'submitter_id': 'subject-4'},
   {'index_date': 'Study Enrollment',
    'species': 'Homo sapiens',
    'submitter_id': 'subject-3'},
   {'index_date': 'Study Enrollment',
    'species': 'Homo sapiens',
    'submitter_id': 'subject-2'},
   {'index_date': 'Study Enrollment',
    'species': 'Homo sapiens',
    'submitter_id': 'subject-1'}]}}
```

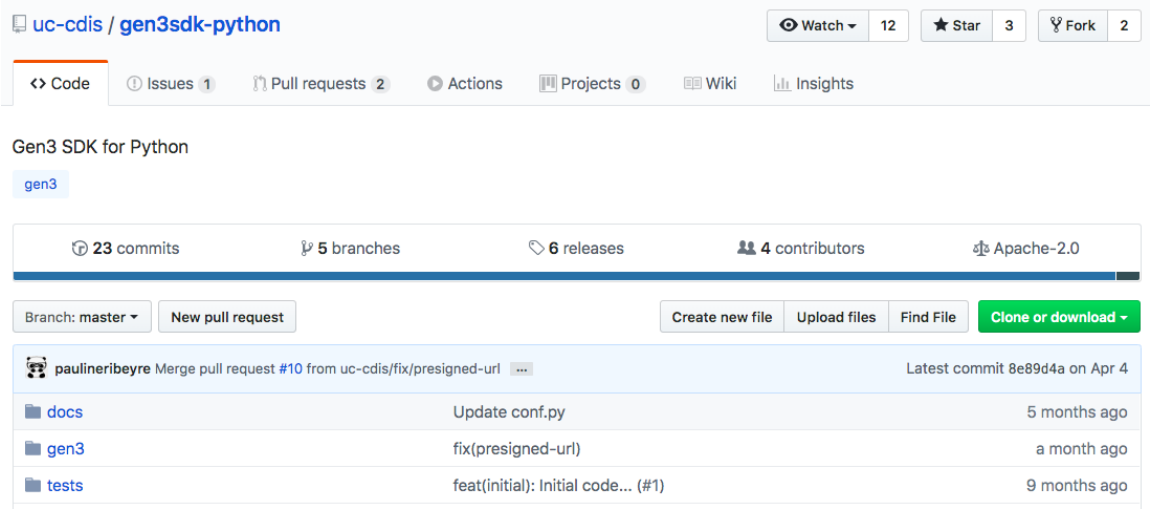
Data Import and Access in the Gen3 Workspace

Import, Export and Query in the Gen3 Workspace JupyterHub

Data Import and Access in JupyterHub

GEN3 Data Commons

- Data can be exported programmatically in, for example, a Python notebook using **the gen3-sdk**, which is an open-source suite of functions for interacting with Gen3 APIs.
- Import the gen3sdk in Python using “import gen3”
- The gen3sdk code lives on GitHub: https://github.com/uc-cdis/gen3sdk-python



Workspace JupyterHub Demonstration

GEN3 Data Commons

- Now, we will take a look at the Gen3 Workspace, featuring data query, export, and import in JupyterHub

Future of Services

Guppy GraphQL: introduction

GEN3 Data Commons

- *Simple* GraphQL schema to explore Flat model.
- But *powerful*, support everything Arranger does and more:
  - Histogram with bin aggregation for numbers;
  - No 10000 results limit;
  - JSON-based filters.

Future plans:

- Tiered access;
- Support searching by ontology values and it's synonyms;
- SQL syntax for filters;
- Full-text search.

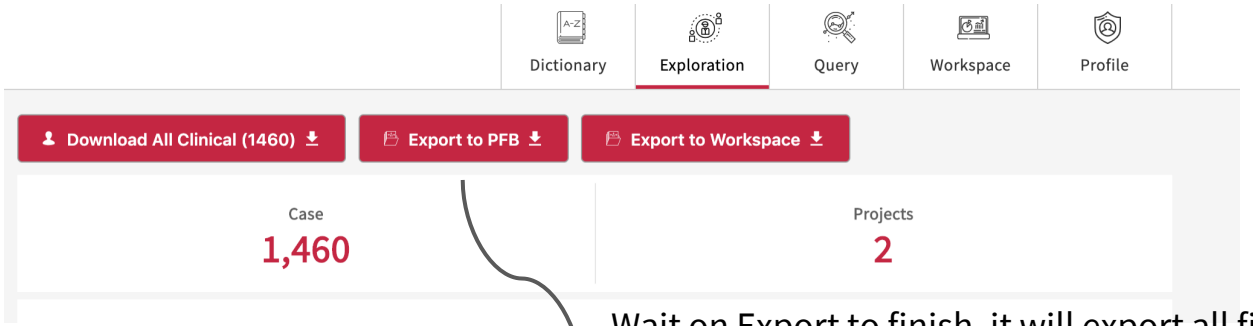
This:

```
{
  subject {
    race
  }
}
```

Not this:

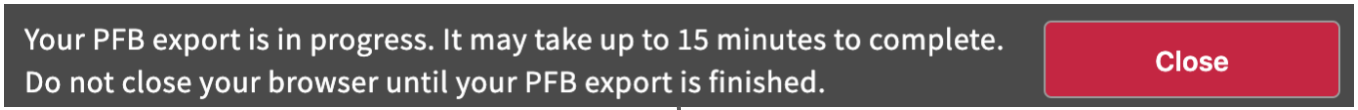
```
{
  subject {
    hits {
      total
      edges {
        node {
          id
          race
        }
      }
    }
  }
}
```

Export clinical data to PFB

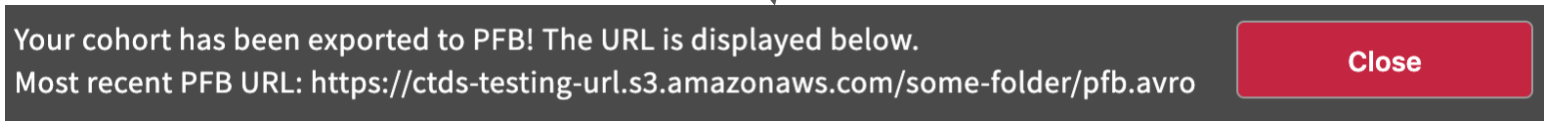


Portable Format
for
Biomedical Data

Wait on Export to finish, it will export all filtered and available clinical data



After some time it will provide a copyable
URL to PFB export of the clinical data



Learn More

- github.com/uc-cdis
- gen3.org
- Gen3 Community on Slack
- support@datacommons.io
- ctds.uchicago.edu

Selected Data Commons Using Gen3

GEN3 Data Commons

NIH NATIONAL CANCER INSTITUTE
Cancer Research Data Commons

Gabriella Miller Kids First
PEDIATRIC RESEARCH PROGRAM

Da ta STAGE

NIAID DATA HUB PILOT

NIH National Heart, Lung, and Blood Institute

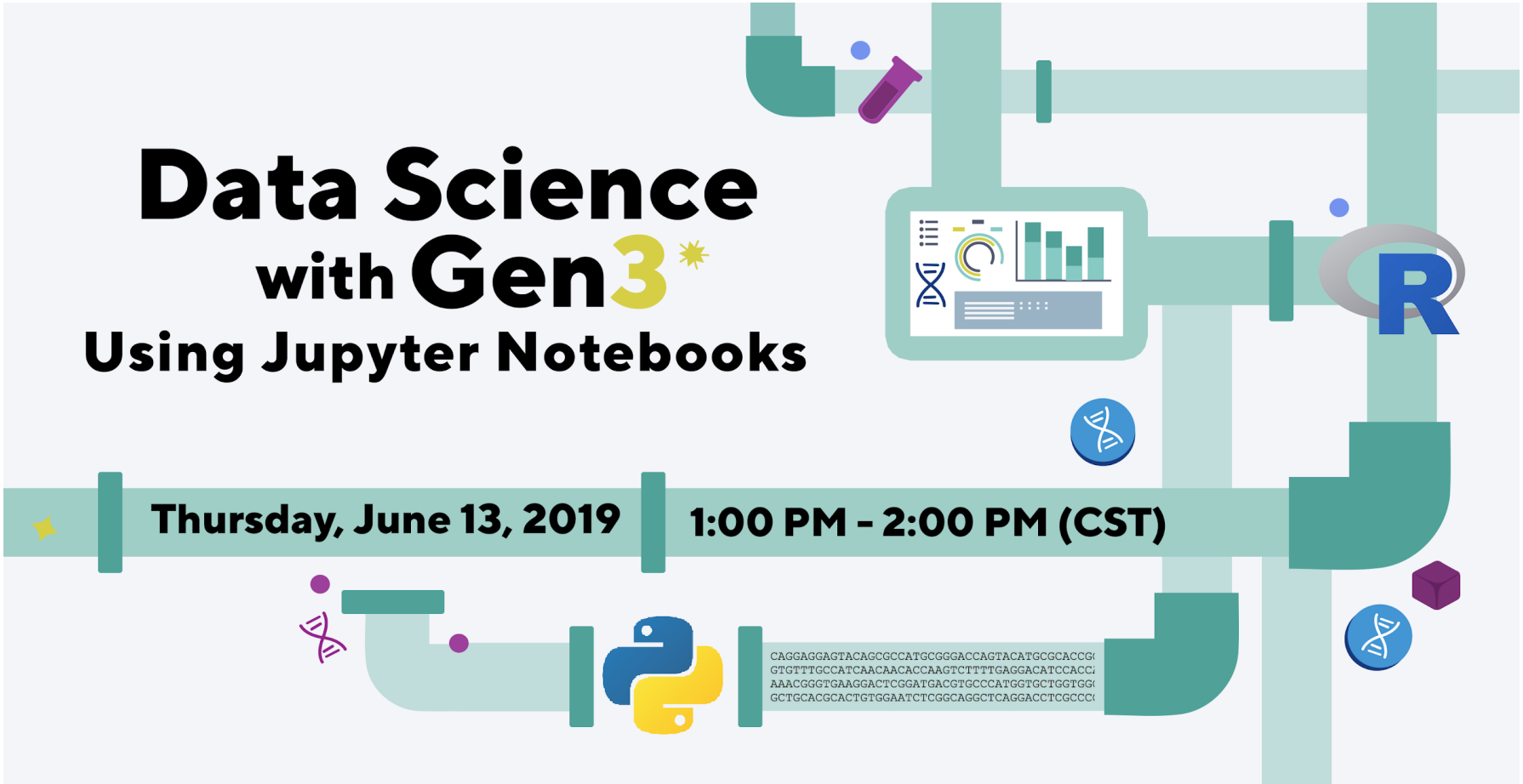
AnVIL

BRAINCOMMONS

NIH National Human Genome Research Institute

Next Webinar

GEN3 Data Commons



Questions?

GEN3 Data Commons