

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

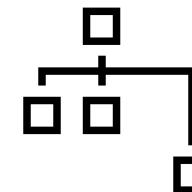
1. What is a Data Model?

1. Structure of a Gen3 Data Model

1. Herding Data Submissions: Data Import and Export

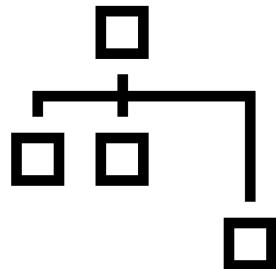
2. Hunting Down Data: Querying and Filtering Data

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



Workspace

What is a Data Model?



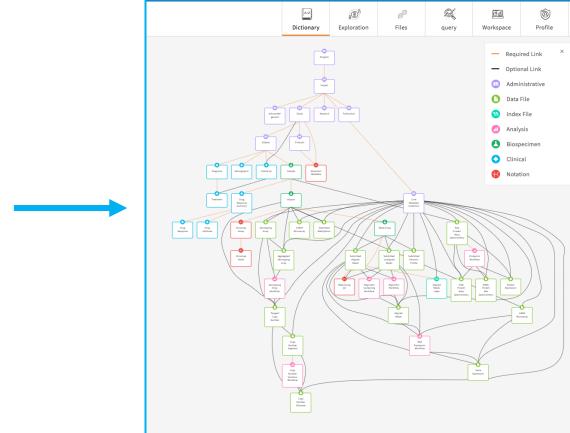
What is a Data Model?

- 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.

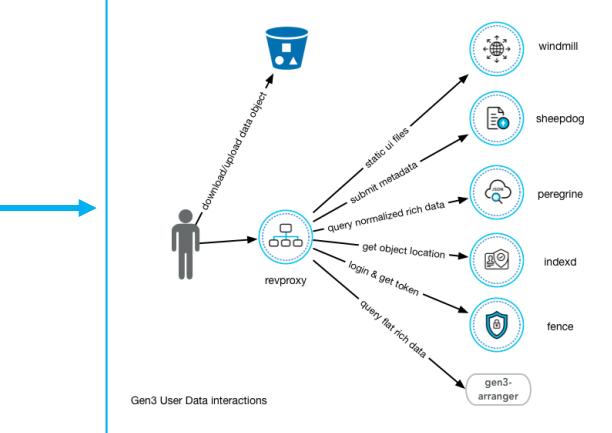
Data Dictionary

Variable Name	Description	Clinical ID	Data Type	MinValue	MaxValue	Values
date_enroll_to_int_data_clinical-new_form	Physical Exam: Date from Enrollment To Intake Form	100	Integer	780		
EDS_electroencephalogram	Physical Exam: Electroencephalogram (EDS)	101	Enum			"Not Obtained", "2-normal", "3-abnormal", "4-refer"
eyes_abnormal	Physical Exam: Eyes Abnormal	102	Enum			"Not Observed", "2-Norm", "3-Mildly impaired", "4-Clearly Impaired"
finger_to_nose_right_hand	Physical Exam: Finger to Nose - Right Hand	103	Enum			"Not Observed", "2-Norm", "3-Mildly impaired", "4-Clearly Impaired"
finger_to_nose_left_hand	Physical Exam: Finger to Nose - Left Hand	104	Enum			"Not Observed", "2-Norm", "3-Mildly impaired", "4-Clearly Impaired"
head_ct_scan	Physical Exam: Head CT Scan	105	Enum			"Not Observed", "2-Norm", "3-A", "4-B"
head_MRI	Physical Exam: Head MRI	106	Enum			"Not Observed", "2-normal", "3-A", "4-B"
hearing_left_ear	Physical Exam: Hearing - Left Ear	107	Enum			"Not Observed", "2-normal", "3-A", "4-B"
hearing_right_ear	Physical Exam: Hearing - Right Ear	108	Enum			"Not Observed", "2-normal", "3-A", "4-B"
heart_exam	Physical Exam: Heart Exam	109	Enum			"Not Observed", "2-normal", "3-Mildly impaired", "4-Clearly Impaired"
head_to_shin_right_log	Physical Exam: Head to Shin - Right Log	110	Enum			"Not Observed", "2-normal", "3-Mildly impaired", "4-Clearly Impaired"
horizontal_eye_movement_left_eye	Physical Exam: Horizontal Eye Movement - Left Eye	111	Enum			"Not Observed", "2-normal", "3-Mildly impaired", "4-Clearly Impaired"
horizontal_eye_movement_right_eye	Physical Exam: Horizontal Eye Movement - Right Eye	112	Enum			"Not Observed", "2-normal", "3-Mildly impaired", "4-Clearly Impaired"
left_arm_sensation	Physical Exam: Left Arm Sensation	113	Enum			"Not Observed", "2-normal", "3-Mildly impaired", "4-Clearly Impaired"
left_arm_stretch	Physical Exam: Left Arm Stretch	114	Enum			"Not Observed", "2-normal", "3-Mildly impaired", "4-Clearly Impaired"
left_leg_sensation	Physical Exam: Left Leg Sensation	115	Enum			"Not Observed", "2-normal", "3-Mildly impaired", "4-Clearly Impaired"
left_leg_strength	Physical Exam: Left Leg Strength	116	Enum			"Not Observed", "2-normal", "3-Mildly impaired", "4-Clearly Impaired"
lungs_dysfunction	Physical Exam: Lungs Dysfunctional	117	Enum			"Not Observed", "2-normal", "3-A", "4-B"
lungs_dysfunctionality	Physical Exam: Lungs Dysfunctional	118	Enum			"Not Observed", "2-normal", "3-A", "4-B"
other_neurodiagnose_1	Physical Exam: Other Neurodiagnosis 1	119	Enum			"Not Observed", "2-normal", "3-A", "4-B"
other_neurodiagnose_2	Physical Exam: Other Neurodiagnosis 2	120	Enum			"Not Observed", "2-normal", "3-A", "4-B"
other_neurodiagnose_3	Physical Exam: Other Neurodiagnosis 3	121	Enum			"Not Observed", "2-normal", "3-A", "4-B"
phys_exam_body_systems_category	Physical Exam: Physical Examination: Body System Category	122	Enum			"Abnormal", "Cardiovascular", "Hepat
phys_exam_cervical_spine	Physical Exam: Physical Examination: Cervical Spine	123	Enum			"Abnormal", "Normal"
phys_exam_cervical_spine	Physical Exam: Physical Examination: Cervical Spine Visit	124	Enum			"Abnormal", "Normal"
phys_exam_result	Physical Exam: Physical Examination Result	125	Enum			"Abnormal", "Normal"
reflex_left_ankle	Physical Exam: Reflex Left Ankle	126	Enum			"Not Observed", "2-normal", "3-A", "4-B"
reflex_left_knee	Physical Exam: Reflex Left Knee	127	Enum			"Not Observed", "2-normal", "3-A", "4-B"
reflex_left_tendon	Physical Exam: Reflex Left Tendon	128	Enum			"Not Observed", "2-normal", "3-A", "4-B"
reflex_left_wrist	Physical Exam: Reflex Left Wrist	129	Enum			"Not Observed", "2-normal", "3-A", "4-B"
reflex_right_ankle	Physical Exam: Reflex Right Ankle	130	Enum			"Not Observed", "2-normal", "3-A", "4-B"
reflex_right_knee	Physical Exam: Reflex Right Knee	131	Enum			"Not Observed", "2-normal", "3-A", "4-B"
reflex_right_tendon	Physical Exam: Reflex Right Tendon	132	Enum			"Not Observed", "2-normal", "3-A", "4-B"
reflex_right_wrist	Physical Exam: Reflex Right Wrist	133	Enum			"Not Observed", "2-normal", "3-A", "4-B"
right_arm_stretch	Physical Exam: Right Arm Stretch	134	Enum			"Not Observed", "2-normal", "3-Mildly impaired", "4-Clearly Impaired"
right_leg_sensation	Physical Exam: Right Leg Sensation	135	Enum			"Not Observed", "2-normal", "3-Mildly impaired", "4-Clearly Impaired"
right_leg_stretch	Physical Exam: Right Leg Stretch	136	Enum			"Not Observed", "2-normal", "3-Mildly impaired", "4-Clearly Impaired"
right_pincer_response	Physical Exam: Right Pincer Response	137	Enum			"Not Observed", "2-normal", "3-Mildly impaired", "4-Clearly Impaired"
skin_abnormal	Physical Exam: Skin Abnormal	138	Enum			"Not Observed", "2-normal", "3-Mildly impaired", "4-Clearly Impaired"

Data Model

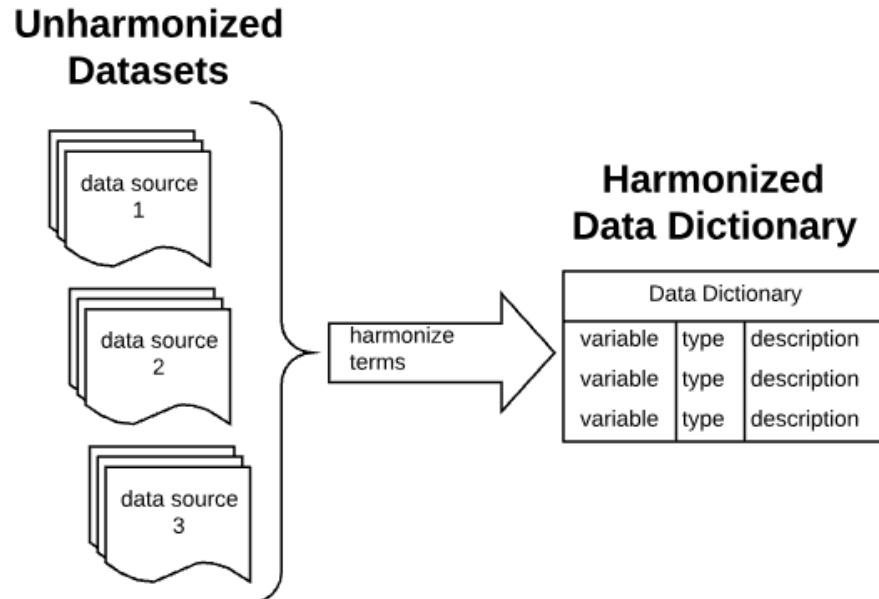


Gen3 Services



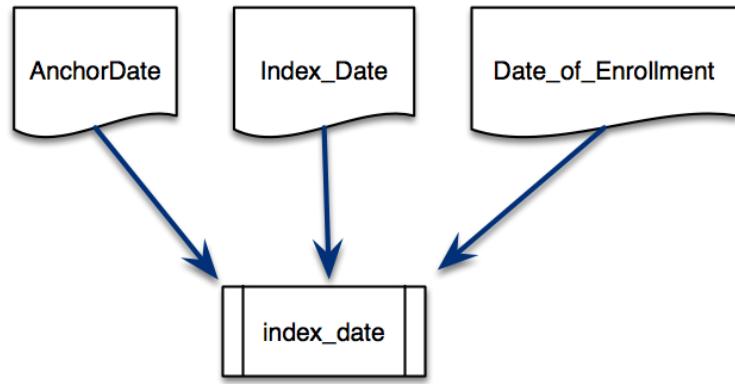
What is the Data Dictionary?

- 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?

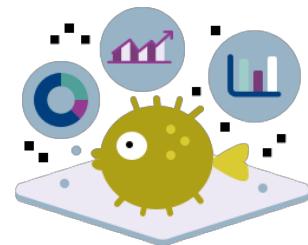
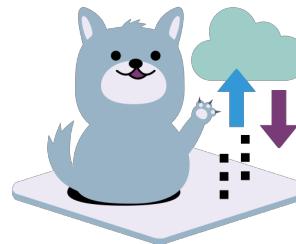
- Dictionaries get everyone on the same page:
 - Define nodes and properties used across different but similar projects in a process called *data harmonization*.
 - Help avoid inconsistencies in data reporting and use across projects.
 - Make data easier to find, subset and analyze by enforcing Data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



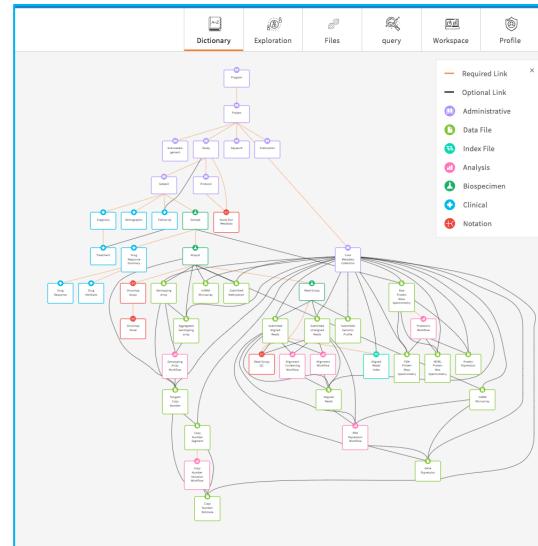
Example: Different studies have their own unique term for “the date a participant enrolled in a clinical trial”. Those terms are *harmonized*, or mapped a single term, in the data dictionary.

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.



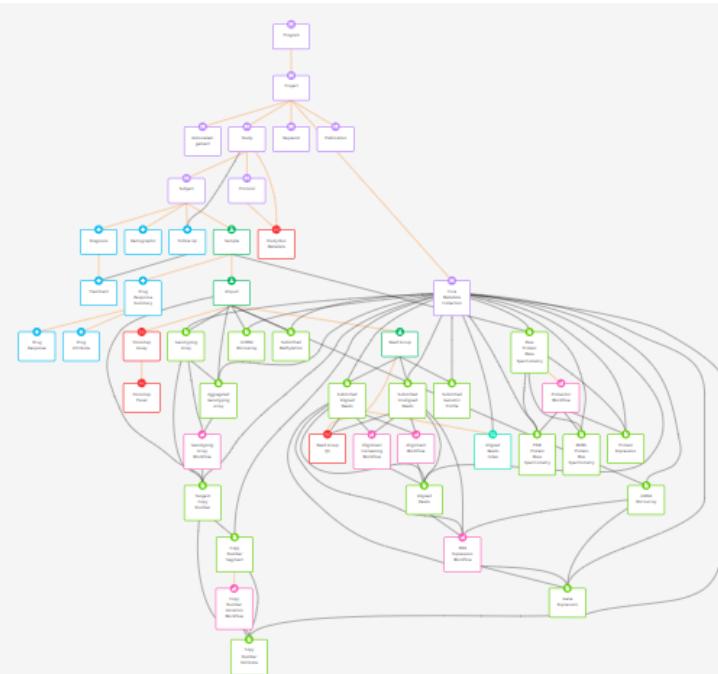
Structure of a Gen3 Data Model



Structure of the Gen3 Data Model

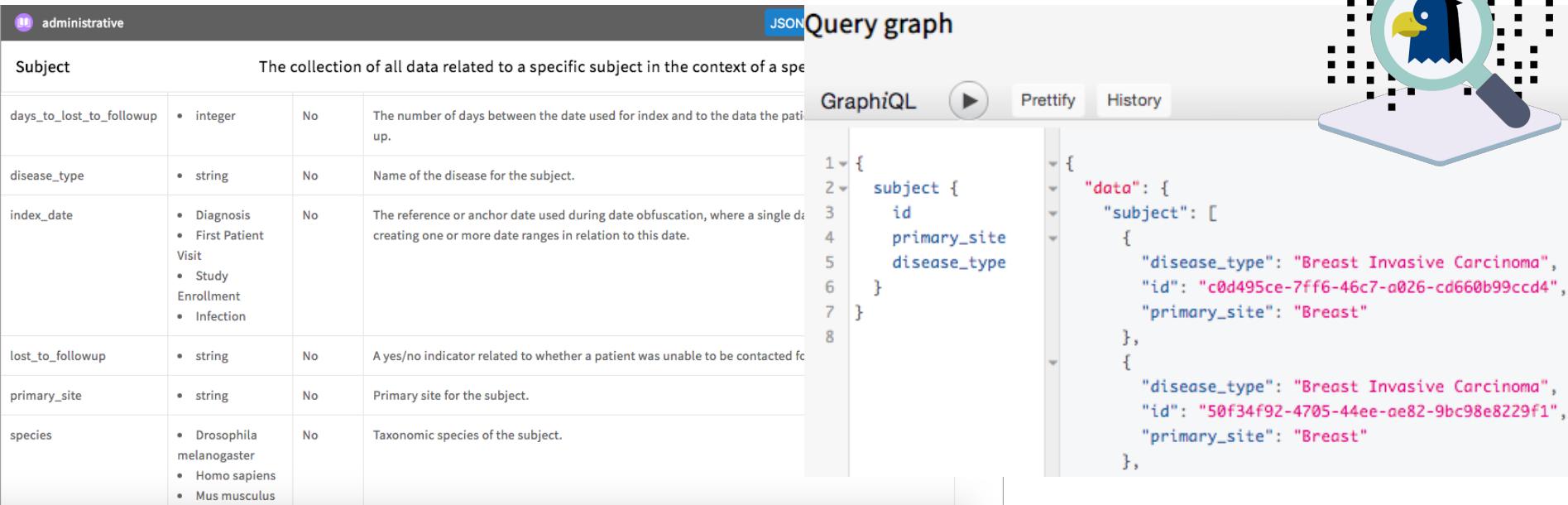
- The Gen3 Data Model is a graph-like relational model consisting of interrelated **nodes** that store certain related **properties**.

Subject			
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 date 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 obscured by creating one or more date ranges in relation to the original date.
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).



The screenshot shows a GraphQL query interface with the following components:

- Table on the left:** A table titled "Subject" with columns for "days_to_lost_to_followup", "disease_type", "index_date", "lost_to_followup", "primary_site", and "species". Each row contains a description of the property's purpose.
- GraphQL Query (top right):** A query starting with "subject {". The JSON output below it shows two records for "Breast Invasive Carcinoma" with IDs "c0d495ce-7ff6-46c7-a026-cd660b99cccd4" and "50f34f92-4705-44ee-ae82-9bc98e8229f1".
- JSON Output (right side):** The JSON response from the GraphQL query, showing the requested data structure.
- Icon on the right:** An illustration of a blue bird with a magnifying glass, symbolizing search or discovery.

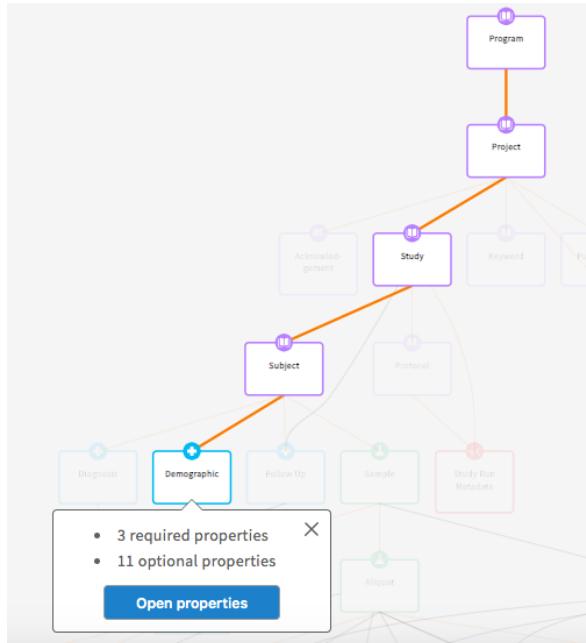
Subject			
days_to_lost_to_followup	• integer	No	The number of days between the date used for index and to the date the patient up.
disease_type	• string	No	Name of the disease for the subject.
index_date	• Diagnosis • First Patient Visit • Study Enrollment • Infection	No	The reference or anchor date used during date obfuscation, where a single date 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.

```
1  {
2   subject {
3     id
4     primary_site
5     disease_type
6   }
7 }
```

```
1 {
2   "data": [
3     {
4       "subject": [
5         {
6           "disease_type": "Breast Invasive Carcinoma",
7           "id": "c0d495ce-7ff6-46c7-a026-cd660b99cccd4",
8           "primary_site": "Breast"
9         },
10        {
11          "disease_type": "Breast Invasive Carcinoma",
12          "id": "50f34f92-4705-44ee-ae82-9bc98e8229f1",
13          "primary_site": "Breast"
14        }
15      ]
16    }
17 }
```

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 X
Demographic		Data for the characterization of the patient by means of segmenting the population (e.g.,characterization by age, sex, or race).			Data File	Term
Property	Type	Required	Description			
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

- 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
1 $schema: "http://json-schema.org/draft-04/schema#"
2
3 id: "demographic"
4 title: Demographic
5 type: object
6 namespace: https://nci-crdc-demo.datacommons.io/
7 category: clinical
8 program: '*'
9 project: '*'
10 description: >
11   Data for the characterization of the patient by means of
12 additionalProperties: false
13 submittable: true
14 validators: null
15
16 systemProperties:
17   - id
18   - project_id
19   - state
20   - created_datetime
21   - updated_datetime
22
23 links:
24   - name: subjects
25     backref: demographics
26     label: describes
27     target_type: subject
28     multiplicity: one_to_one
29     required: true
```

Structure of the Gen3 Data Model

- Property definitions include:

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

```
biospecimen.yaml .  
370   biospecimen_weight:  
371     description: "For solid tissue biospecimens this is the total weight in milligrams."  
372     type: number  
373  
374   blood_draw_method:  
375     description: "The name or generalized description of the method used to draw the specimen."  
376     type: string  
377  
378   blood_tube_type:  
379     description: "The kind of tube used to collect the sample(s) taken from a biospecimen."  
380     type: string  
381     enum:  
382       - "EDTA"  
383       - "CellSave"  
384       - "Streck"  
385       - "Acid Citrate Dextrose (ACD)"  
386       - "Not Applicable"  
387       - "Unknown"  
388  
389   days_to_collection:  
390     description: "The number of days between the index date and the date the biospecimen was collected."  
391     type: integer  
392  
393   days_to_collection_not_reported:  
394     description: "True/False indicator of whether the number of days between the index date and the collection date were not reported."  
395     type: boolean  
396
```

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.

```
_definitions.yaml
```

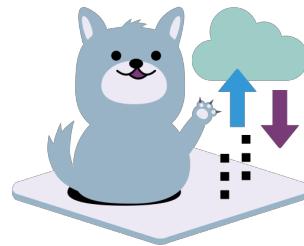
```
1 id: _definitions
2
3 UUID:
4   term:
5     $ref: "_terms.yaml#/UUID"
6   type: string
7   pattern: "^[a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{12}$"
```

```
demographic.yaml
```

```
43 cause_of_death:
44   description: >
45     Text term to identify the cause of patient death with respect to cancer.
46   enum:
47     - "Cancer Related"
48     - "Not Cancer Related"
49     - "Unknown"
50
51 days_to_birth:
52   description: >
53     The number of days between the index date and the date of patient birth.
54     If the number of days is greater than 32872 (89 years), then please use
55     'days_to_birth_gt89'.
56   type: integer
57   maximum: 32872
58   minimum: 0
59
60 days_to_birth_gt89:
61   description: >
62     Indicate if the number of days between the index date and the date of
63     patient birth is greater than 32872 (89 years).
64   enum:
65     - "Yes"
66     - "No"
```

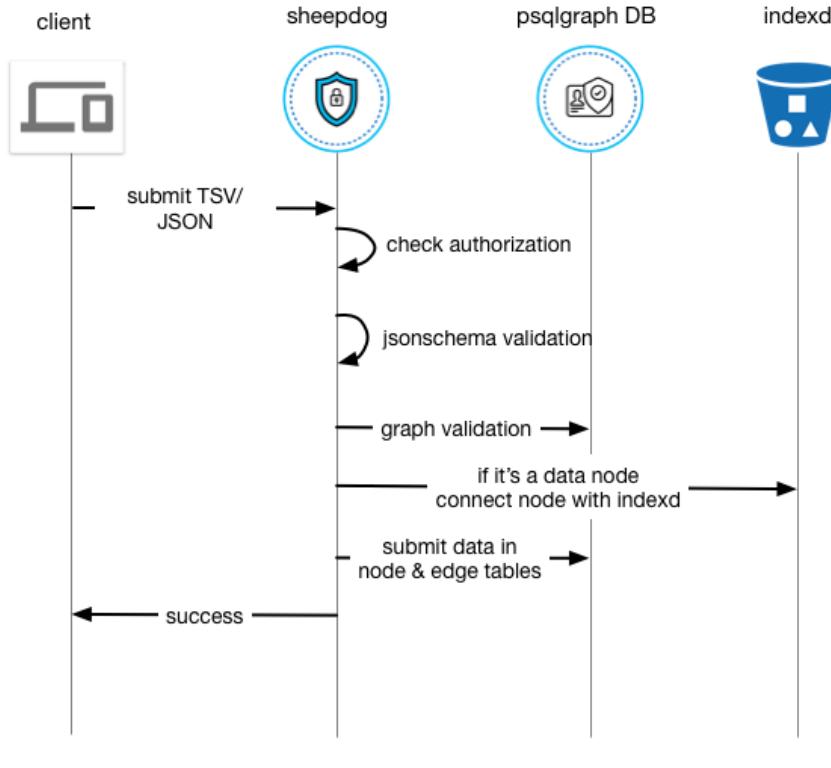
Herding Data Submissions

The Submission Service



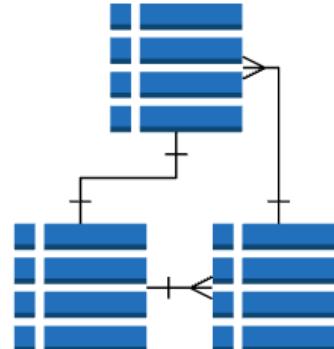
Herding Data Submissions

- 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.



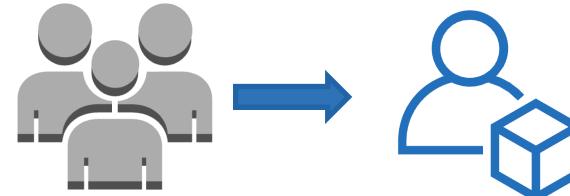
Herding Data Submissions: Types of Data

- **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.



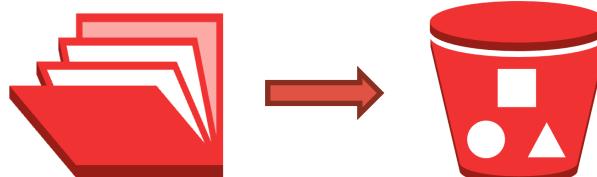
Herding Data Submissions: Steps for Data Import

1. User Authorization



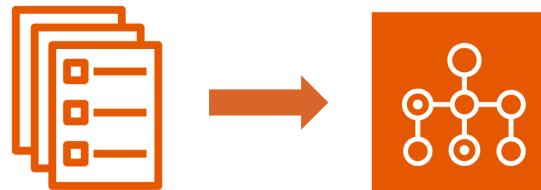
1. Data File Upload:

- 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:

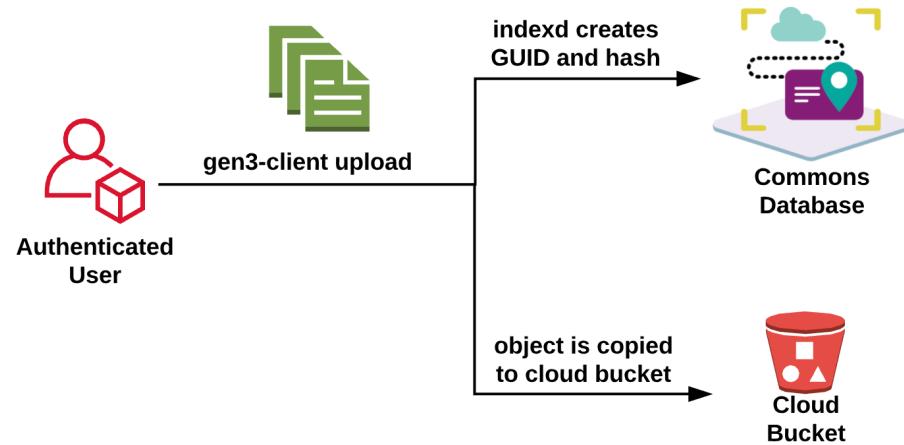
- a. Submit Structured Data
- b. Link Data File Records to their structured data



- 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

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.

1

The screenshot shows the 'Data Submission' section of the GEN3 Data Commons interface. It features three main components: 'Gen3 Client' (with a sub-section 'Powerful Uploading for Large Files'), 'Dictionary' (with a sub-section 'Read Tutorials'), and 'Map My Files' (with a sub-section 'Mapping files to metadata in order to create medical meaning.' and a red box around the 'Map My Files' button). The 'Map My Files' button is the focal point of this step.

2

The screenshot shows the 'My Files' page with a table of uploaded files. Each file has a checkbox next to it. A large number '2' is overlaid on the right side of the table.

3

The screenshot shows the 'Mapping 3 files to Data Model' page. It includes a 'Project' dropdown set to 'DCF-demo', a 'File Node' dropdown set to 'submitted_unaligned_reads', and a 'Required Fields' section with four entries: 'data_category' (Raw Sequencing Data), 'data_type' (Unaligned Reads), 'data_format' (FASTQ), and 'experimental_strategy' (WGS). A large number '3' is overlaid on the left side of the form.

4

The screenshot shows a hierarchical diagram of the 'Core Metadata Collection'. At the top is 'Project 1', which points down to 'Core Metadata Collection 1', which in turn points down to 'Submitted Unaligned Reads 3'. To the right, an orange box displays the message '3 files mapped successfully!' and a 'Submit' button. A large number '4' is overlaid on the right side of the diagram.

Structured Data Upload: Overview



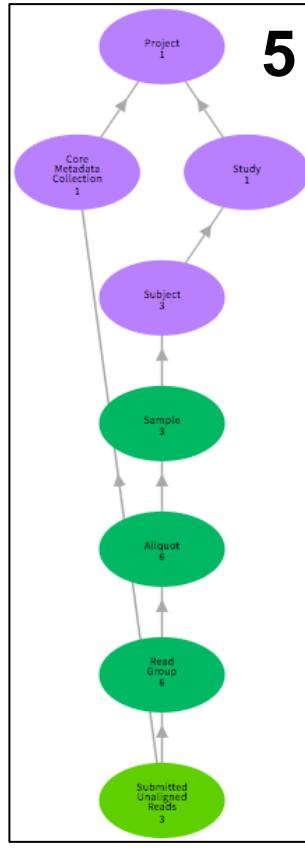
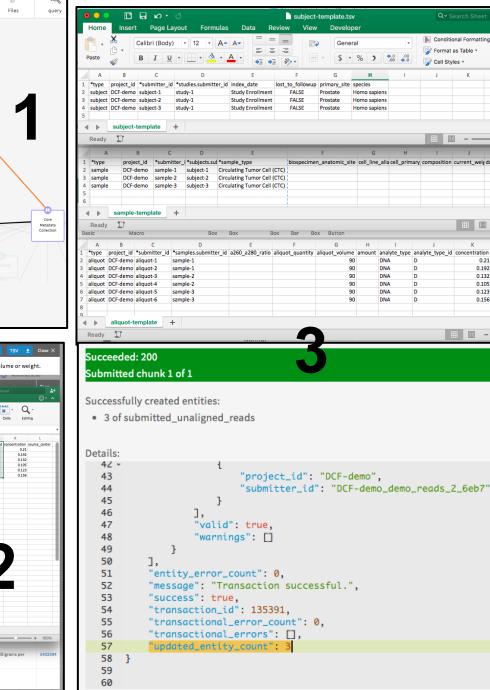
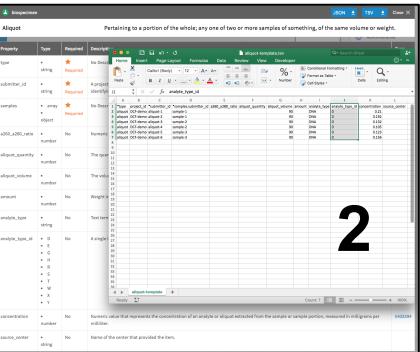
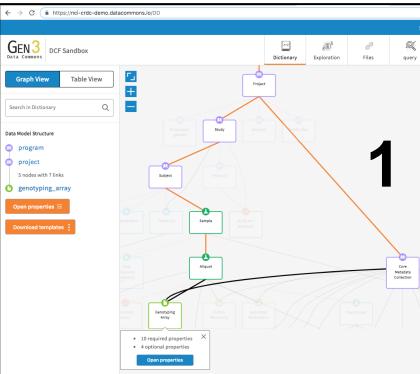
- Now that files are uploaded and mapped to the data model, the rest of the project's ***structured data*** must be submitted and linked to the data file records.
 - Sheepdog ensures ***structured data*** conform to the data model, and values can be queried via Peregrine
 - Data files*** on the other 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.



Structured Data: TSV Submission Troubleshooting

- 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:
1~ []
2~ {
3~   "action": null,
4~   "errors": [
5~     {
6~       "keys": [
7~         "days_to_lost_to_followup"
8~       ],
9~       "message": "'five' is not of type 'integer'",
10~      "type": "INVALID_VALUE"
11~    },
12~    {
13~      "keys": [
14~        "species"
15~      ],
16~      "message": "'Homo sapien' is not one of ['Drosophila melanogaster', 'Homo sapiens',
17~      "type": "ERROR"
18~    }
19~  ].
```

	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
5	subject	DCF-demo	subject-4		Study Enrollment	FALSE	Homo sapiens	study-one

Structured Data: TSV Submission Troubleshooting

- 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.

DCF-demo [browse nodes](#)

Use Form Submission

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	

Submitting chunk 1 of 1

Succeeded: 200
Submitted chunk 1 of 1

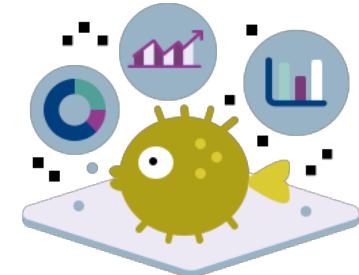
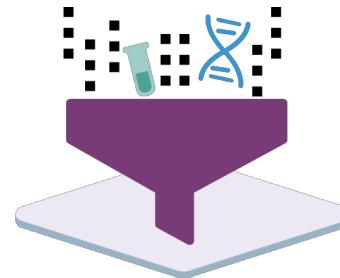
Successfully created entities:

- 4 of subject

	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	

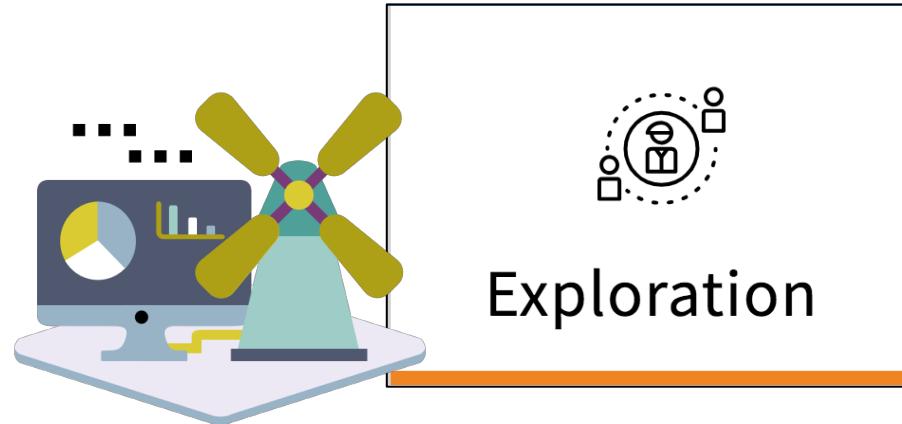
Hunting Down Data

Querying and Filtering Data



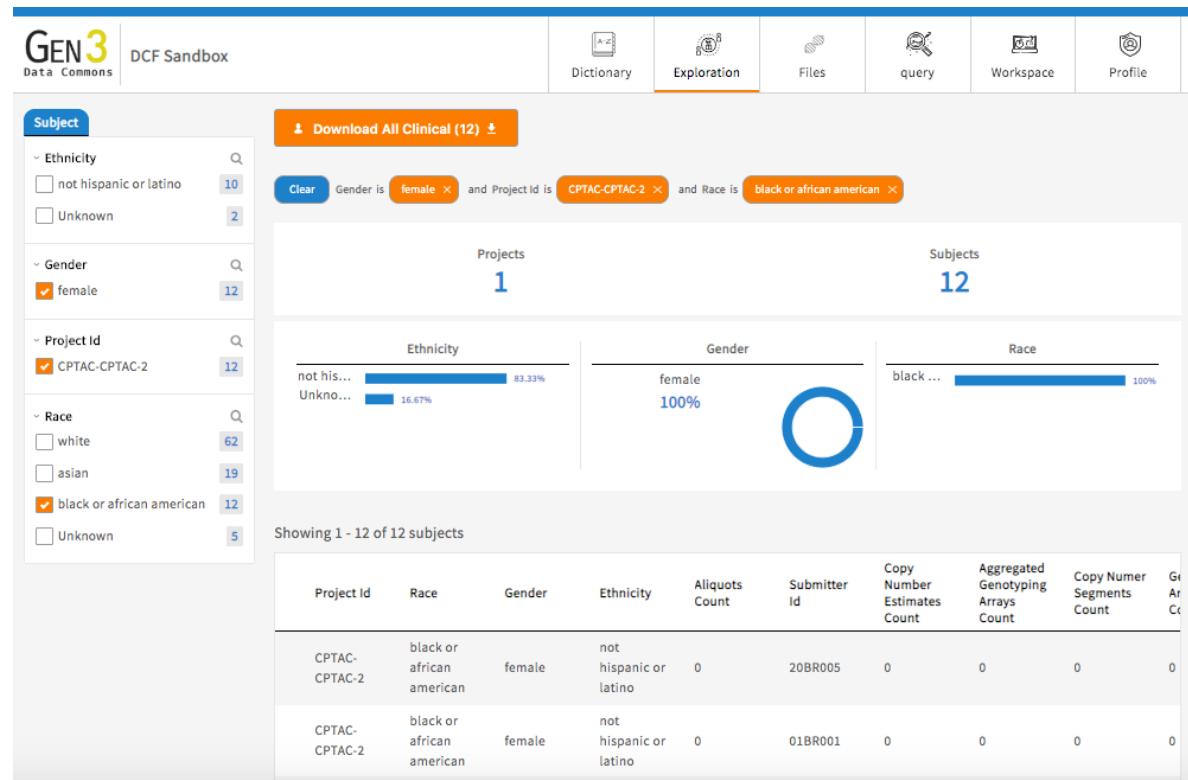
Windmill's Exploration Page

a graphical user interface for cohort selection



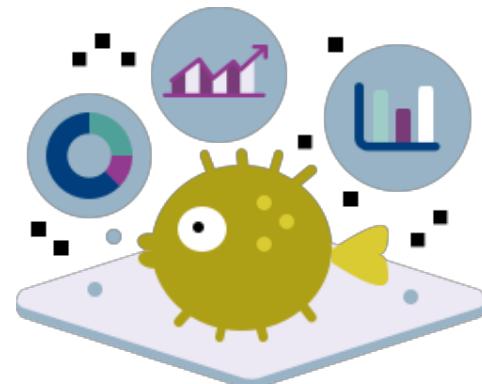
Windmill's Exploration Page

- 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

- 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

Query graph

GraphiQL ▶ Prettify History

```
1 ~ {  
2   subject (project_id:"DCF-demo") {  
3     species  
4     index_date  
5     submitter_id  
6     lost_to_followup  
7     studies {  
8       submitter_id  
9     }  
10   }  
11 }  
12
```

QUERY VARIABLES

subject

subject

TYPE

[subject]

ARGUMENTS

order_by_asc: String

updated_datetime: [String]

with_path_to_any: [WithPathToInput]

tissue_source_site_code: [String]

submitter_id: [String]

created_before: String

id: String

with_path_to: [WithPathToInput]

without_links: [String]

created_datetime: [String]

breed: [String]

species: [String]

Switch to Flat Model



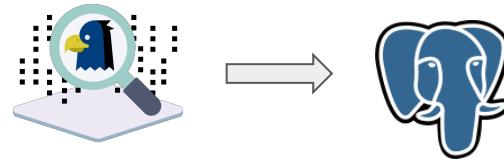
Windmill's Query Page

- 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/



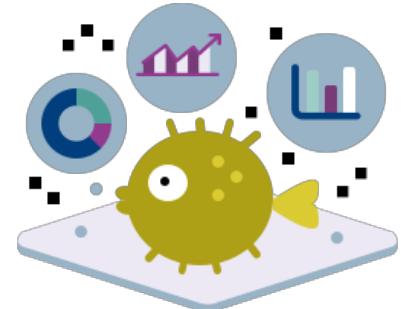
- Flat Model hits the ElasticSearch DB
vpodc.org/api/v0/flat-search/search/graphql



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 screenshot shows a GraphiQL interface with two panes. The left pane contains a GraphQL query, and the right pane displays the resulting JSON data.

```
1 {  
2   subject {  
3     aggregations {  
4       race {  
5         buckets {  
6           key  
7           doc_count  
8         }  
9       }  
10    }  
11  }  
12}  
13
```

The right pane shows the JSON response:

```
"data": {  
  "subject": {  
    "aggregations": {  
      "race": {  
        "buckets": [  
          {  
            "key": "white",  
            "doc_count": 62  
          },  
          {  
            "key": "asian",  
            "doc_count": 19  
          },  
          {  
            "key": "black or african american",  
            "doc_count": 12  
          },  
          {  
            "key": "Unknown",  
            "doc_count": 5  
          },  
          {  
            "key": "__missing__",  
            "doc_count": 1060  
          }  
        ]  
      }  
    }  
  }  
}
```

The GraphQL Endpoints

- 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',
'signer_id': 'subject-4'},
{'index_date': 'Study Enrollment',
'species': 'Homo sapiens',
'signer_id': 'subject-3'},
{'index_date': 'Study Enrollment',
'species': 'Homo sapiens',
'signer_id': 'subject-2'},
{'index_date': 'Study Enrollment',
'species': 'Homo sapiens',
'signer_id': 'subject-1'}]}}
```



Data Import and Access in the Gen3 Workspace

Import, Export and Query in the Gen3 Workspace JupyterHub



Workspace

Data Import and Access in JupyterHub

- 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>

The screenshot shows the GitHub repository page for `uc-cdis/gen3sdk-python`. The repository name is at the top left. To the right are buttons for Watch (12), Star (3), Fork (2), and Insights. Below the header are navigation links: Code (selected), Issues (1), Pull requests (2), Actions, Projects (0), Wiki, and Insights. The main content area has a heading "Gen3 SDK for Python" and a search bar containing "gen3". Below the search bar are statistics: 23 commits, 5 branches, 6 releases, 4 contributors, and Apache-2.0 license. A "Clone or download" button is prominent. At the bottom, there's a list of recent commits:

Author	Commit Message	Date
paulineribeyre	Merge pull request #10 from uc-cdis/fix/presigned-url	Latest commit 8e89d4a on Apr 4
docs	Update conf.py	5 months ago
gen3	fix(presigned-url)	a month ago
tests	feat(initial): Initial code... (#1)	9 months ago

Workspace JupyterHub Demonstration

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

The screenshot shows a JupyterHub workspace interface. At the top, there is a navigation bar with tabs for "Files" (selected), "Running", and "Clusters". Below the navigation bar, a message says "Select items to perform actions on them." On the left, there is a file browser with a sidebar showing "0" items selected. The main area lists two files: "webinar_demo" (a folder) and "dcf-credentials.json". To the right of the file list are filters for "Name" (sorted by name), "Last Modified" (sorted by last modified time), and "File size".

Name	Last Modified	File size
webinar_demo	27 minutes ago	
dcf-credentials.json	31 minutes ago	773 B

Future of Services

Guppy GraphQL: introduction

- *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.

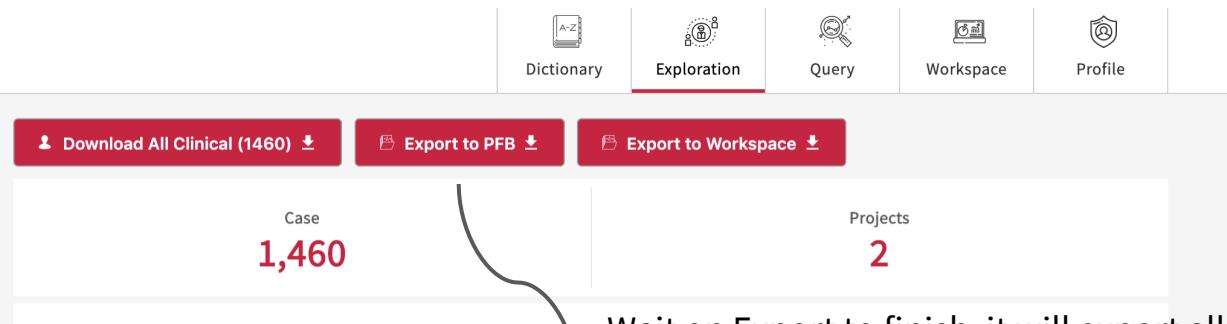
This:

```
{  
  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

Your PFB export is in progress. It may take up to 15 minutes to complete.
Do not close your browser until your PFB export is finished.

Close

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

Your cohort has been exported to PFB! The URL is displayed below.
Most recent PFB URL: <https://ctds-testing-url.s3.amazonaws.com/some-folder/pfb.avro>

Close

Learn More



- github.com/uc-cdis



- gen3.org



- Gen3 Community on Slack



- support@datacommons.io



- ctds.uchicago.edu

Selected Data Commons Using Gen3



Next Webinar

GEN3 Data Commons

Data Science with Gen3*

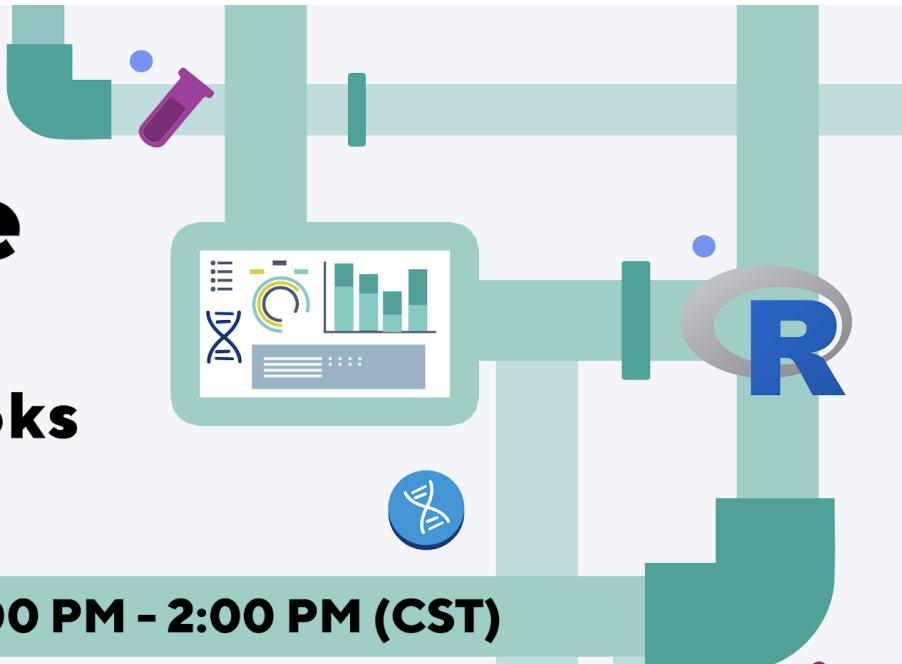
Using Jupyter Notebooks

Thursday, June 13, 2019

1:00 PM - 2:00 PM (CST)



CAGGAGGAGTACAGCCCCATGCGGGACCACTACATGCGCACCG
GTGTTGGCCATCAACACACCAAGTCTTTTGAGGACATCCACC
AAACGGGTGAAGGACTCGGATGACCTGCCCATGGTCCTGGTGC
GCTGCACGCACTGTGGATCTCGCAGGCTCAGGACCTCGCC



Questions?



GEN3 Data Commons