

# 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

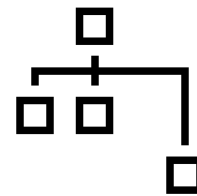
## *Herding Data Submissions & Hunting Down Data*

Chris Meyer, Ph.D.

Center for Translational Data Science,  
University of Chicago

May 9, 2019

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

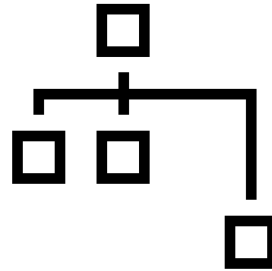


1. Demonstration of Query, Export, and Import in Workspace



Workspace

# What is a Data Model?

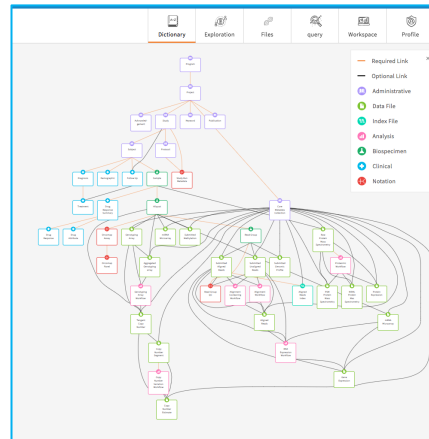


- A **data model** organizes terms in a data dictionary and defines how they relate to one another. It is the implementation of a data dictionary and enables Gen3 services to **submit** and **query** data.

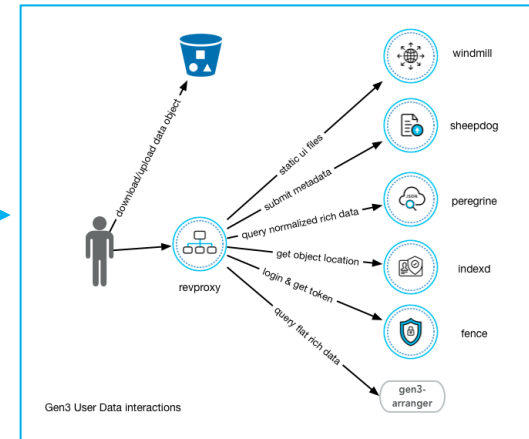
## Data Dictionary

ID	Variable Name	Description	Unit	Values
2200	eyes_abnormal	Physical Exam: Eyes from Endment to Info. Data (Clinical On-Sym)	Enum	100
2201	eyes_abnormal	Physical Exam: Eyes Abnormal	Enum	"No", "Yes"
2202	eyes_abnormal	Physical Exam: Electroretinogram (ERG)	Enum	"1", "Not Obtainable", "2", "Normal", "3", "C"
2203	eyes_abnormal	Physical Exam: Eyes Abnormal	Enum	"No", "Yes"
2204	finger_to_nose_right_hand	Physical Exam: Finger to Nose - Right Hand	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2205	finger_to_nose_right_hand	Physical Exam: Finger to Nose - Right Hand	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2206	finger_to_nose_right_hand	Physical Exam: Finger to Nose - Right Hand	Enum	"No", "Yes"
2207	finger_to_nose_right_hand	Physical Exam: Finger to Nose - Right Hand	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2208	general_abnormality	Physical Exam: General Abnormality	Enum	"No", "Yes"
2209	head_ct_scan	Physical Exam: Head CT Scan	Enum	"1", "Not Obtainable", "2", "Normal", "3", "A"
2210	head_ct_scan	Physical Exam: Head CT	Enum	"1", "Normal", "2", "Normal", "3", "A"
2211	hearing_left_ear	Physical Exam: Hearing - Left Ear	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2212	hearing_left_ear	Physical Exam: Hearing - Left Ear	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2213	hearing_left_ear	Physical Exam: Hearing - Right Ear	Enum	"1", "Normal", "2", "Mildly Impaired", "4"
2214	hearing_left_ear	Physical Exam: Hearing - Right Ear	Enum	"1", "Normal", "2", "Mildly Impaired", "4"
2215	hearing_right_ear	Physical Exam: Hearing - Left Ear	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2216	hearing_right_ear	Physical Exam: Hearing - Right Ear	Enum	"1", "Normal", "2", "Mildly Impaired", "4"
2217	horizontal_eye_movement_left_eye	Physical Exam: Horizontal Eye Movement - Left Eye	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2218	horizontal_eye_movement_left_eye	Physical Exam: Horizontal Eye Movement - Right Eye	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2219	left_arm_sensation	Physical Exam: Left Arm Sensation	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2220	left_arm_strength	Physical Exam: Left Arm Strength	Enum	"1", "Normal", "2", "Mildly Impaired", "4"
2221	left_arm_strength	Physical Exam: Left Arm Strength	Enum	"1", "Normal", "2", "Mildly Impaired", "4"
2222	left_arm_strength	Physical Exam: Left Leg Sensation	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2223	left_arm_strength	Physical Exam: Left Leg Strength	Enum	"1", "Normal", "2", "Mildly Impaired", "4"
2224	left_arm_strength	Physical Exam: Left Leg Strength	Enum	"1", "Normal", "2", "Mildly Impaired", "4"
2225	left_arm_strength	Physical Exam: Left Plantar Response	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2226	left_arm_strength	Physical Exam: Left Plantar Response	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2227	left_arm_strength	Physical Exam: Left Plantar Response	Enum	"No", "Yes"
2228	left_arm_strength	Physical Exam: Left Plantar Response	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2229	left_arm_strength	Physical Exam: Left Plantar Response	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2230	left_arm_strength	Physical Exam: Left Plantar Response	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2231	left_arm_strength	Physical Exam: Left Plantar Response	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2232	left_arm_strength	Physical Exam: Left Plantar Response	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2233	left_arm_strength	Physical Exam: Left Plantar Response	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2234	left_arm_strength	Physical Exam: Left Plantar Response	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2235	left_arm_strength	Physical Exam: Left Plantar Response	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2236	left_arm_strength	Physical Exam: Left Plantar Response	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2237	left_arm_strength	Physical Exam: Left Plantar Response	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2238	left_arm_strength	Physical Exam: Left Plantar Response	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2239	left_arm_strength	Physical Exam: Left Plantar Response	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2240	left_arm_strength	Physical Exam: Left Plantar Response	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2241	left_arm_strength	Physical Exam: Left Plantar Response	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2242	left_arm_strength	Physical Exam: Left Plantar Response	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2243	left_arm_strength	Physical Exam: Left Plantar Response	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2244	left_arm_strength	Physical Exam: Left Plantar Response	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2245	left_arm_strength	Physical Exam: Left Plantar Response	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2246	left_arm_strength	Physical Exam: Left Plantar Response	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2247	left_arm_strength	Physical Exam: Left Plantar Response	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2248	left_arm_strength	Physical Exam: Left Plantar Response	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2249	left_arm_strength	Physical Exam: Left Plantar Response	Enum	"1", "Normal", "2", "Mildly Impaired", "3"
2250	left_arm_strength	Physical Exam: Left Plantar Response	Enum	"1", "Normal", "2", "Mildly Impaired", "3"

## Data Model



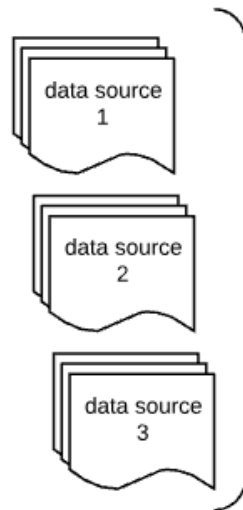
## Gen3 Services



# 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.

## Unharmonized Datasets

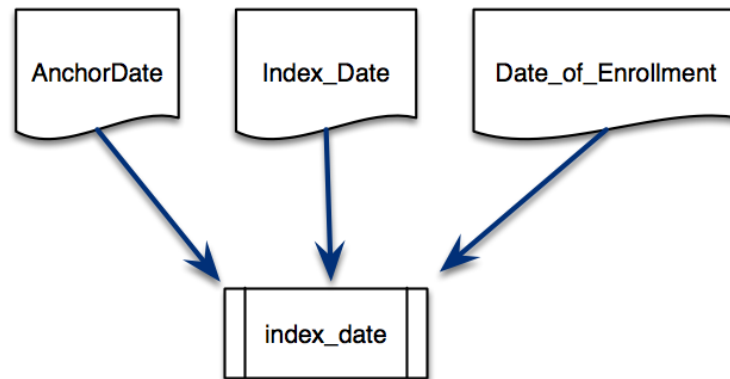


harmonize terms

## Harmonized Data Dictionary

Data Dictionary		
variable	type	description
variable	type	description
variable	type	description

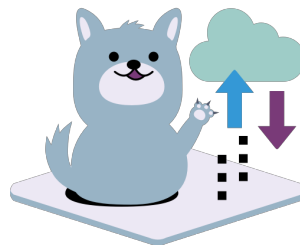
- Dictionaries get everyone on the same page:
  - Define nodes and properties used across different but similar projects in a process called *data harmonization*.
  - Help avoid inconsistencies in data reporting and use across projects.
  - Make data easier to find, subset and analyze by enforcing Data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



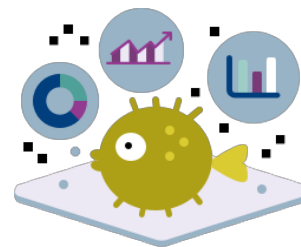
**Example:** Different studies have their own unique term for “the date a participant enrolled in a clinical trial”. Those terms are *harmonized*, or mapped a single term, in the data dictionary.

- 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.

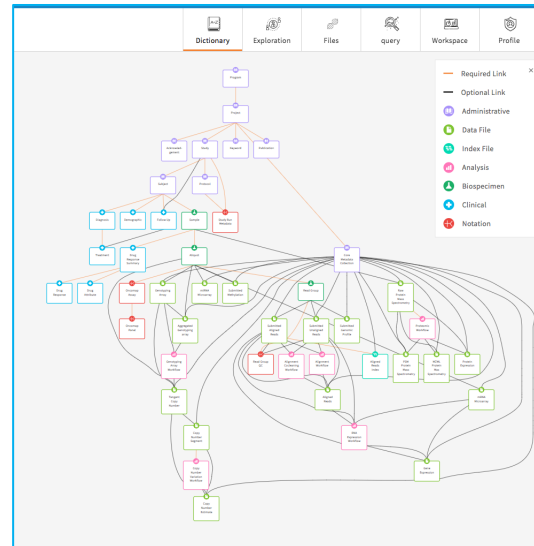


- **Database queries** are facilitated by the *Peregrine*, *Arranger*, and *Guppy* services. Queries must conform to the data model for successful data retrieval.





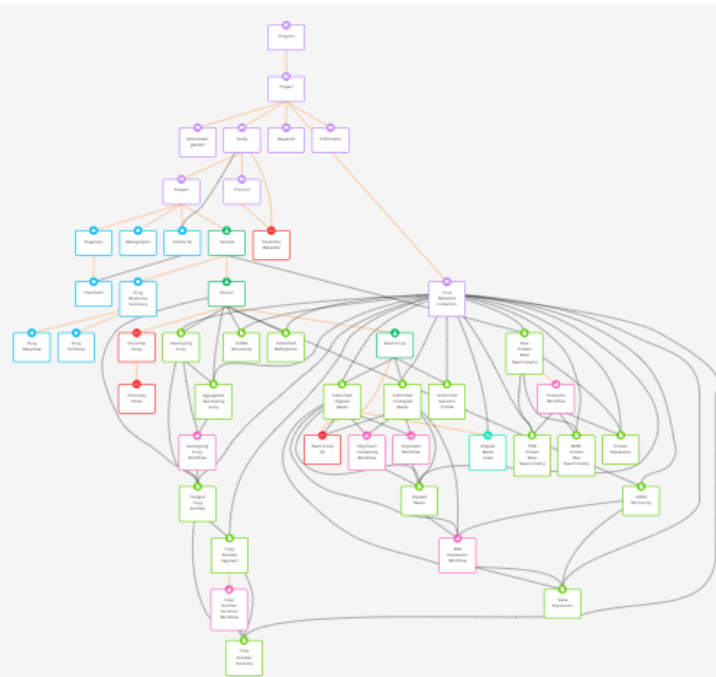
# Structure of a Gen3 Data Model



# 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 obscured by creating one or more date ranges in
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 Query graph			
Subject	The collection of all data related to a specific subject in the context of a spe		
days_to_lost_to_followup	<ul style="list-style-type: none"><li>integer</li></ul>	No	The number of days between the date used for index and to the data the pati up.
disease_type	<ul style="list-style-type: none"><li>string</li></ul>	No	Name of the disease for the subject.
index_date	<ul style="list-style-type: none"><li>Diagnosis</li><li>First Patient Visit</li><li>Study Enrollment</li><li>Infection</li></ul>	No	The reference or anchor date used during date obfuscation, where a single d creating one or more date ranges in relation to this date.
lost_to_followup	<ul style="list-style-type: none"><li>string</li></ul>	No	A yes/no indicator related to whether a patient was unable to be contacted fo
primary_site	<ul style="list-style-type: none"><li>string</li></ul>	No	Primary site for the subject.
species	<ul style="list-style-type: none"><li>Drosophila melanogaster</li><li>Homo sapiens</li><li>Mus musculus</li></ul>	No	Taxonomic species of the subject.

Query graph

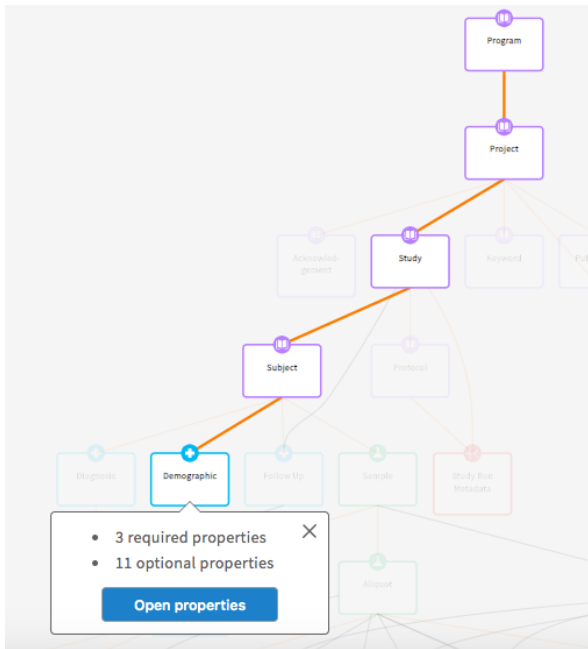
GraphiQL ▶ Prettify History

```
1 {
2   subject {
3     id
4     primary_site
5     disease_type
6   }
7 }
8
```

```
{
  "data": {
    "subject": [
      {
        "disease_type": "Breast Invasive Carcinoma",
        "id": "c0d495ce-7ff6-46c7-a026-cd660b99ccd4",
        "primary_site": "Breast"
      },
      {
        "disease_type": "Breast Invasive Carcinoma",
        "id": "50f34f92-4705-44ee-ae82-9bc98e8229f1",
        "primary_site": "Breast"
      }
    ]
  }
}
```



- 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

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.	

- 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
```

- 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 →
371   - biospecimen_weight:~
372     - description: "For solid tissue biospecimens this is the total weight in mill
373     - type: number~
374 →
375   - blood_draw_method:~
376     - description: "The name or generalized description of the method used to draw
377     - type: string~
378 →
379   - blood_tube_type:~
380     - description: "The kind of tube used to collect the sample(s) taken from a bi
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 bio
391     - type: integer~
392 →
393   - days_to_collection_not_reported:~
394     - description: "True/False indicator of whether the number of days between the
395     - type: boolean~
396 →
```

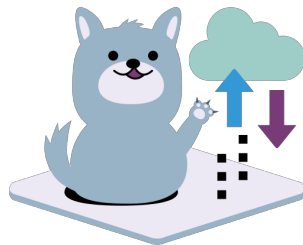
- Limitations can be put on acceptable property values:
  - **Minimum/maximum** for integers/numbers.
  - **Enumerations** are limited strings.
  - Strings can be required to match patterns.
- Submitted records that do not conform fail.

```
demographic.yaml
43 →
44 → cause_of_death:~
45 →   description: >~
46 →     Text term to identify the cause of patient death with respect to cancer.
47 →   enum:~
48 →     - "Cancer Related"~
49 →     - "Not Cancer Related"~
50 →     - "Unknown"~
51 →
52 → days_to_birth:~
53 →   description: >~
54 →     The number of days between the index date and the date of patient birth.
55 →     If the number of days is greater than 32872 (89 years), then please use~
56 →     'days_to_birth_gt89'.~
57 →   type: integer~
58 →   maximum: 32872~
59 →   minimum: 0~
60 →
61 → days_to_birth_gt89:~
62 →   description: >~
63 →     Indicate if the number of days between the index date and the date of~
64 →     patient birth is greater than 32872 (89 years).~
65 →   enum:~
66 →     - "Yes"~
67 →     - "No"~
68 →
```

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

# Herding Data Submissions

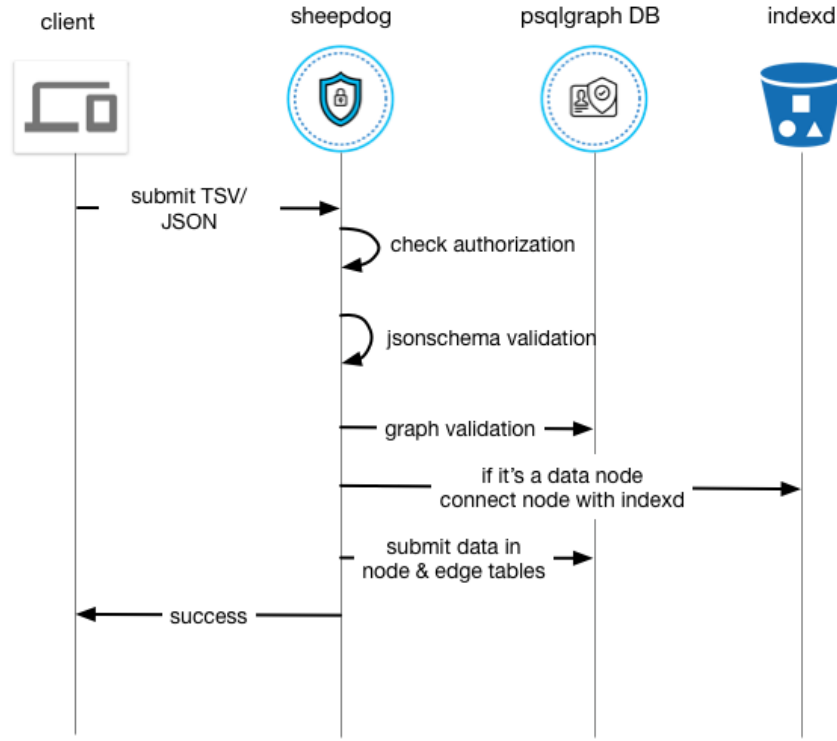
*The Submission Service*



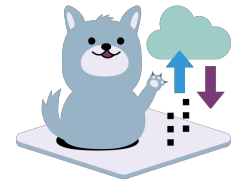


# Herding Data Submissions

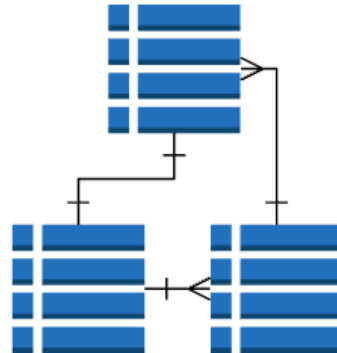
- The Sheepdog service shepherds submissions of structured data into the graph database.
- Sheepdog checks validity of each record in a data upload against the data dictionary to ensure all required fields are present and have appropriate data values.
- Sheepdog also supports export of structured data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



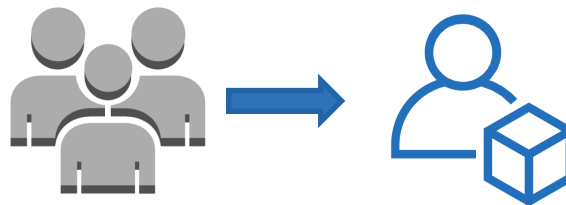
Gen3 rich data submission



- **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.

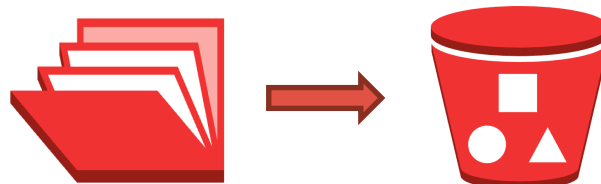


## 1. User Authorization



## 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:

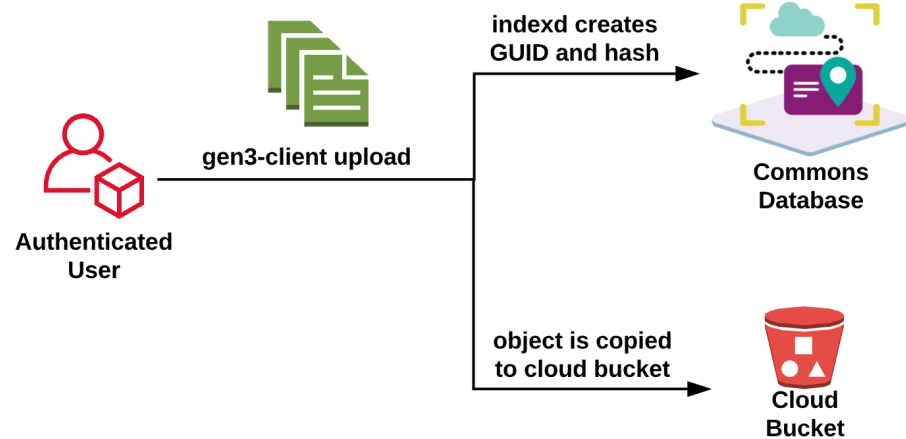
- a. Submit Structured Data
- b. Link Data File Records to their structured data



- 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



## 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
```

## 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.

My Files

Unmapped Files Map Files (3)

uploaded on 04/03/19, 7 files

<input type="checkbox"/>	File Name	Size	Uploaded Date	Status
<input type="checkbox"/>	demo_reads_4.bam	41 B	04/03/19, 07:26:27 pm UTC-05:00	Ready
<input checked="" type="checkbox"/>	demo_reads_2.fastq	37 B	04/03/19, 06:55:47 pm UTC-05:00	Ready
<input type="checkbox"/>	demo_reads_3.bam	46 B	04/03/19, 07:26:28 pm UTC-05:00	Ready
<input checked="" type="checkbox"/>	demo_reads_1.fastq	37 B	04/03/19, 07:26:20 pm UTC-05:00	Ready
<input checked="" type="checkbox"/>	demo_reads_1.fastq	27 B	04/03/19, 06:55:46 pm UTC-05:00	Ready
<input type="checkbox"/>	demo_reads_3.bam	38 B	04/03/19, 07:26:32 pm UTC-05:00	Ready

2

3 files mapped successfully!

Mapping 3 files to Data Model

Assign Project and Node Type

Project: DCF-demo ✓

File Node: submitted\_unaligned\_reads ✓

Required Fields:

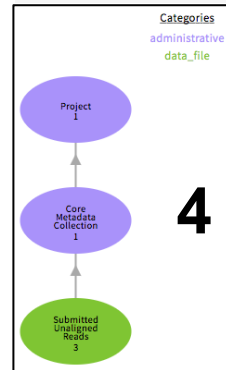
- data\_category: Raw Sequencing Data ✓
- data\_type: Unaligned Reads ✓
- data\_format: FASTQ ✓
- experimental\_strategy: WGS ✓
- core\_metadata\_collection: FASTQ-collection-04032019 ✓

LINK(s) to Parent Node(s):

3 files ready for mapping. Submit

3

Categories  
administrative  
data\_file



4

https://nci-crdc-demo.datacommons.io/submission

GEN<sup>3</sup> Data Commons DCF Sandbox

Dictionary Exploration Files query Workspace Profile

### Data Submission

**1**

Gen3 Client  
Powerful Uploading for Large Files  
Upload your large files quickly and safely without interruptions.  
Read Tutorials Download App ↴

Map My Files  
16 files | 189 B  
Mapping files to metadata in order to create medical meaning.  
Map My Files

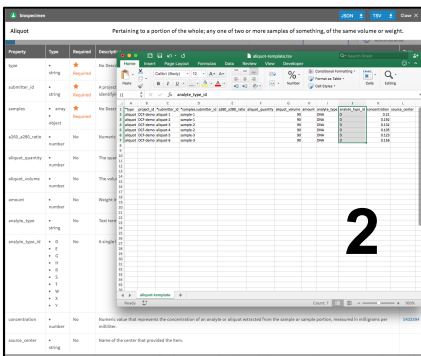
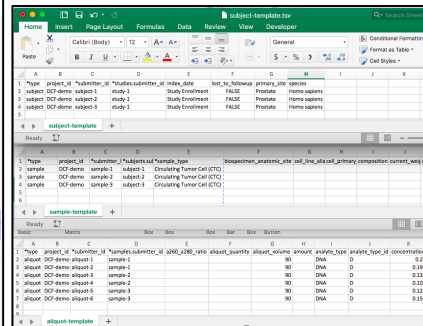
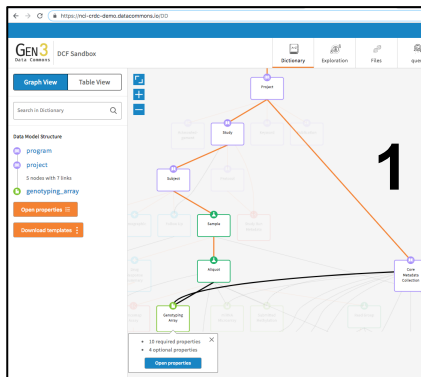
- Now that files are uploaded and mapped to the data model, the rest of the project's **structured data** must be submitted and linked to the data file records.
  - Sheepdog ensures **structured data** conform to the data model, and values can be queried via Peregrine
  - **Data files** on the other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



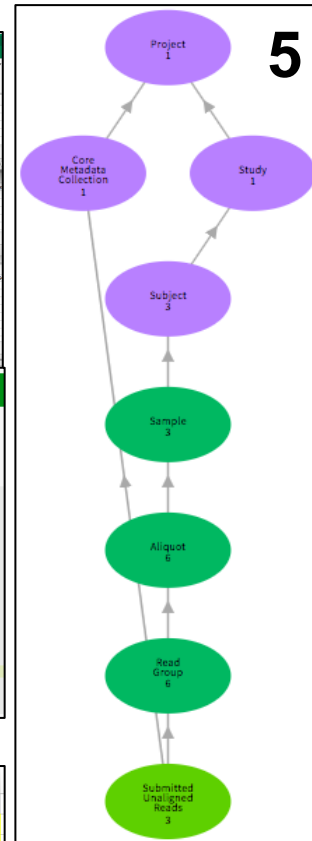
# Structured Data Upload: TSV Submission Process

## ● 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



- 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', 'Homo sapien']",
17 -         "type": "ERROR"
18 -       }
19 -     ]
20 -   }
21 - ]
```

	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

- 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

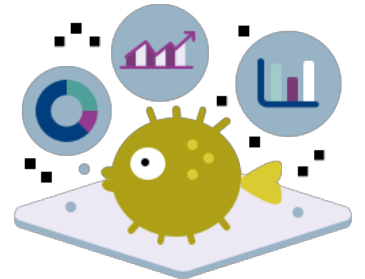
Successfully created entities:

- 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

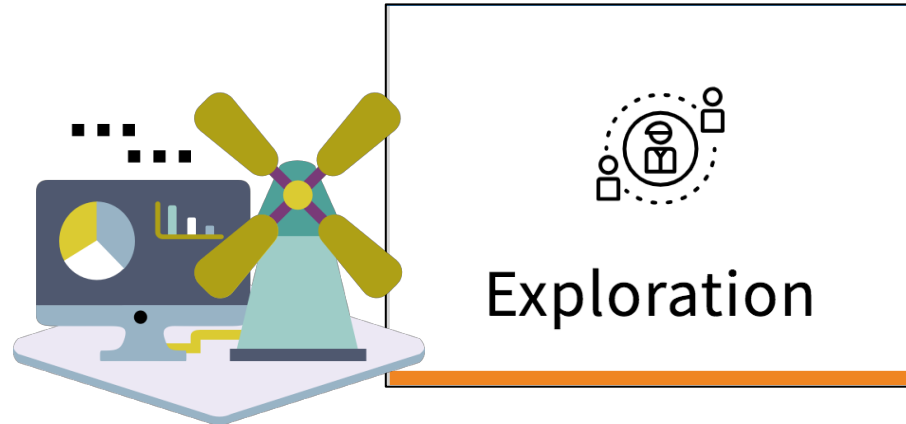
# Hunting Down Data

*Querying and Filtering Data*



# Windmill's Exploration Page

*a graphical user interface for cohort selection*



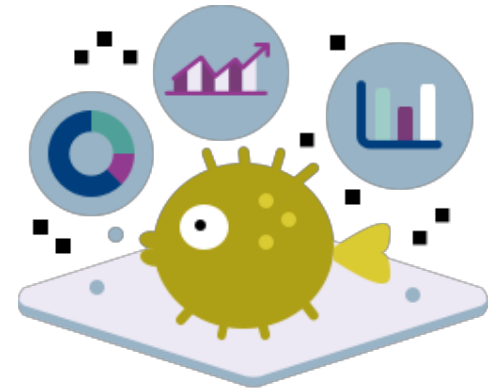
- Cohorts can be selected via a graphical user interface using data facets.
- Once a cohort is selected, a file download manifest can be sent to your Workspace / JupyterHub for easy data file access and analysis

The screenshot displays the GEN3 Data Commons Exploration interface. The top navigation bar includes 'Dictionary', 'Exploration', 'Files', 'query', 'Workspace', and 'Profile'. The main area shows a 'Subject' facet panel on the left with filters for Ethnicity, Gender, Project Id, and Race. The selected filters are: Gender is female (12), Project Id is CPTAC-CPTAC-2 (12), and Race is black or african american (12). The main content area shows a 'Download All Clinical (12)' button and a search bar with the selected filters. Summary statistics show 1 Project and 12 Subjects. Three charts are displayed: Ethnicity (not hispanic or latino: 83.33%, Unknown: 16.67%), Gender (female: 100%), and Race (black or african american: 100%). Below the charts, a table shows the first two subjects:

Project Id	Race	Gender	Ethnicity	Alliquots Count	Submitter Id	Copy Number Estimates Count	Aggregated Genotyping Arrays Count	Copy Number Segments Count	Gr Ar Cc
CPTAC-CPTAC-2	black or african american	female	not hispanic or latino	0	20BR005	0	0	0	0
CPTAC-CPTAC-2	black or african american	female	not hispanic or latino	0	01BR001	0	0	0	0

# API Queries for Cohort Building

*Peregrine, Arranger, and Guppy*



- The GraphQL 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

GraphQL ▶ Prettify History Switch to Flat Model

```
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- }
```

```
{
  "data": {
    "subject": [
      {
        "index_date": "Study Enrollment",
        "lost_to_followup": "FALSE",
        "species": "Homo sapiens",
        "studies": [
          {
            "submitter_id": "study-1"
          }
        ]
      },
      {
        "submitter_id": "subject-4"
      },
      {
        "index_date": "Study Enrollment",
        "lost_to_followup": "TRUE",
        "species": "Homo sapiens",
        "studies": [
          {
            "submitter_id": "study-1"
          }
        ]
      },
      {
        "submitter_id": "subject-3"
      }
    ]
  }
}
```

QUERY VARIABLES

< subject subject X

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]

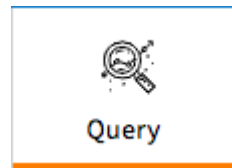
breed: [String]

species: [String]



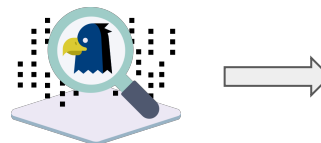


- Switch between the Flat and Graph models on Windmill's "Query" Page.

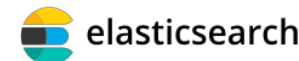


- These use different endpoints that query different databases:

- Graph Model hits the PostgreSQL DB  
[vpodc.org/api/v0/submission/graphql/](http://vpodc.org/api/v0/submission/graphql/)



- Flat Model hits the ElasticSearch DB  
[vpodc.org/api/v0/flat-search/search/graphql](http://vpodc.org/api/v0/flat-search/search/graphql)



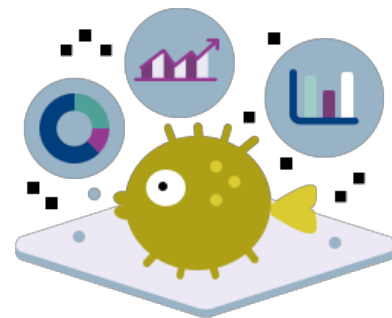
- Graph Model

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



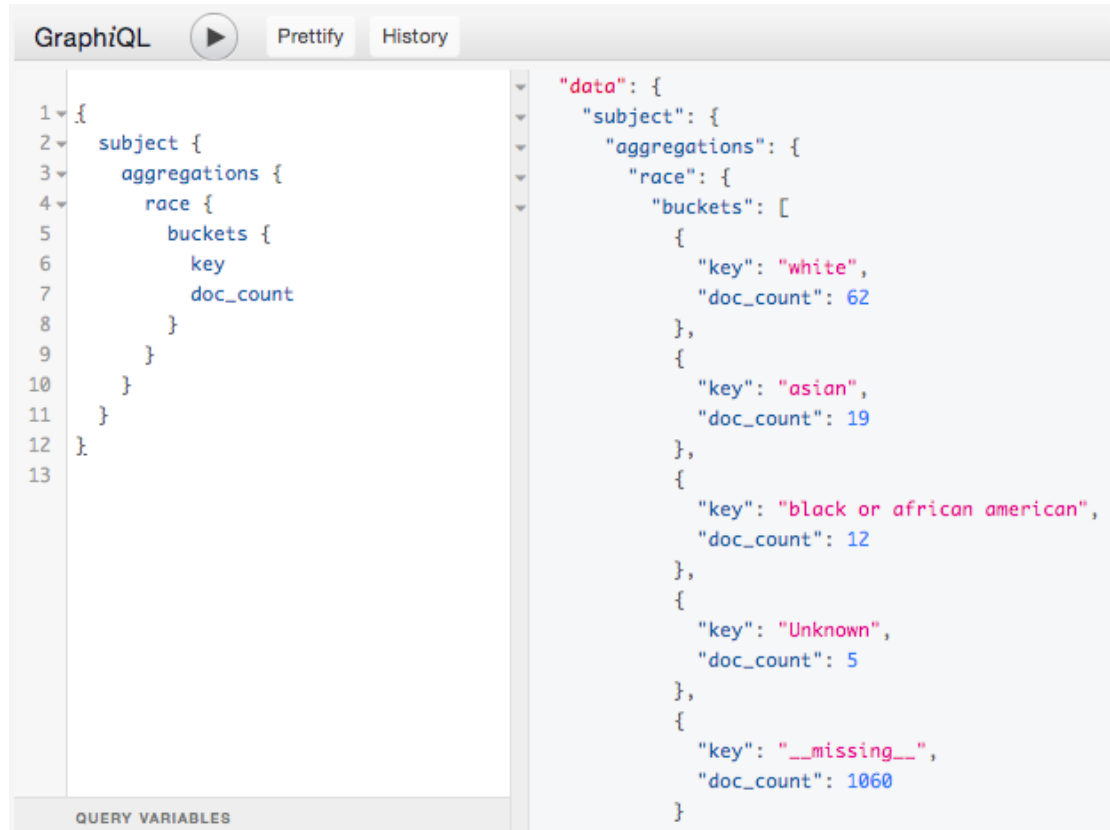
- Flat Model

- **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 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 GraphQL IDE interface with a query on the left and its JSON response on the right. The query is a flat model aggregation query for the 'race' field. The response shows the aggregated data for each race category.

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

```
"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
          }
        ]
      }
    }
  }
}
```

QUERY VARIABLES

- 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*



Workspace

- 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`. At the top, it displays the repository name, a 'Watch' button with 12 users, a 'Star' button with 3 stars, and a 'Fork' button with 2 forks. Below this is a navigation bar with tabs for 'Code', 'Issues 1', 'Pull requests 2', 'Actions', 'Projects 0', 'Wiki', and 'Insights'. The main heading is 'Gen3 SDK for Python' with a 'gen3' tag. A summary bar shows 23 commits, 5 branches, 6 releases, 4 contributors, and Apache-2.0 license. Below this are buttons for 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find File', and 'Clone or download'. The commit history table is as follows:

Commit	Message	Time
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

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



The screenshot displays the JupyterHub interface. At the top left, the Jupyter logo is visible. Below it, there are three tabs: 'Files', 'Running', and 'Clusters'. The 'Files' tab is active. A message reads 'Select items to perform actions on them.' To the right of this message are buttons for 'Upload', 'New', and a refresh icon. Below the message is a file browser table. The table has a header with columns for 'Name', 'Last Modified', and 'File size'. The table contains two entries: a folder named 'webinar\_demo' and a file named 'dcf-credentials.json'.

	Name ↓	Last Modified	File size
<input type="checkbox"/>	webinar_demo	27 minutes ago	
<input type="checkbox"/>	dcf-credentials.json	31 minutes ago	773 B

# Future of Services



- *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

The screenshot shows the 'Exploration' tab selected in the top navigation bar. Below the navigation bar, there are three red buttons: 'Download All Clinical (1460)', 'Export to PFB', and 'Export to Workspace'. Below these buttons, there are two white boxes: 'Case 1,460' and 'Projects 2'. An arrow points from the 'Export to PFB' button to the text below.

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



- [github.com/uc-cdis](https://github.com/uc-cdis)



- [gen3.org](https://gen3.org)



- Gen3 Community on Slack



- [support@datacommons.io](mailto:support@datacommons.io)



- [ctds.uchicago.edu](https://ctds.uchicago.edu)

# Selected Data Commons Using Gen3



# Data Science with Gen<sup>3</sup> Using Jupyter Notebooks



★ Thursday, June 13, 2019

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



```
CAGGAGGAGTACAGCGCCATGCGGGACCACTACATGCGCACCGI  
GTGTTTGCCATCAACAACACCAAGTCTTTGAGGACATCCACCI  
AAACGGGTGAAGGACTCGGATGACGTGCCCATGGTGTCTGGTGGI  
GCTGCACGCACTGTGGAATCTCGGCAGGCTCAGGACCTCGCCCI
```



# Questions?

