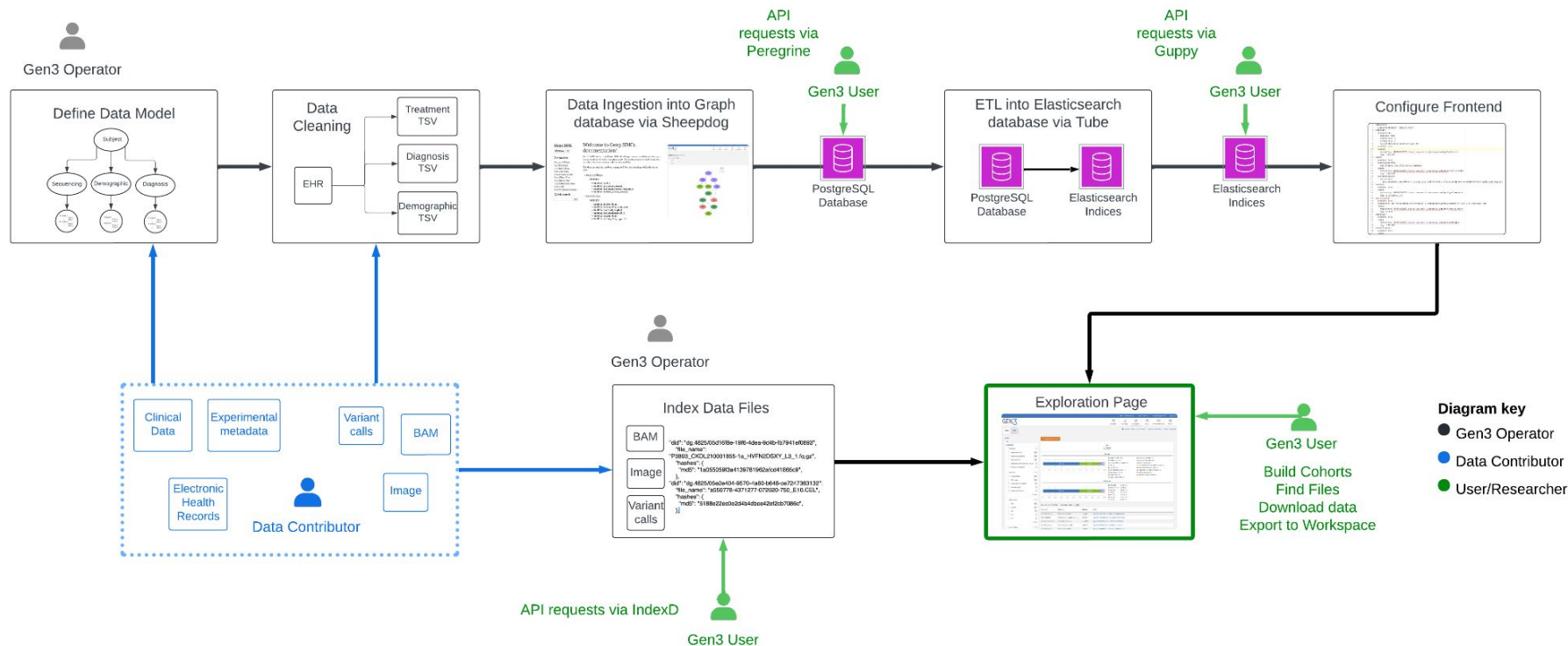


# Understanding ETL and ETL mapping to power the Gen3 Exploration Page

Gen3 Community Forum  
September 3, 2025

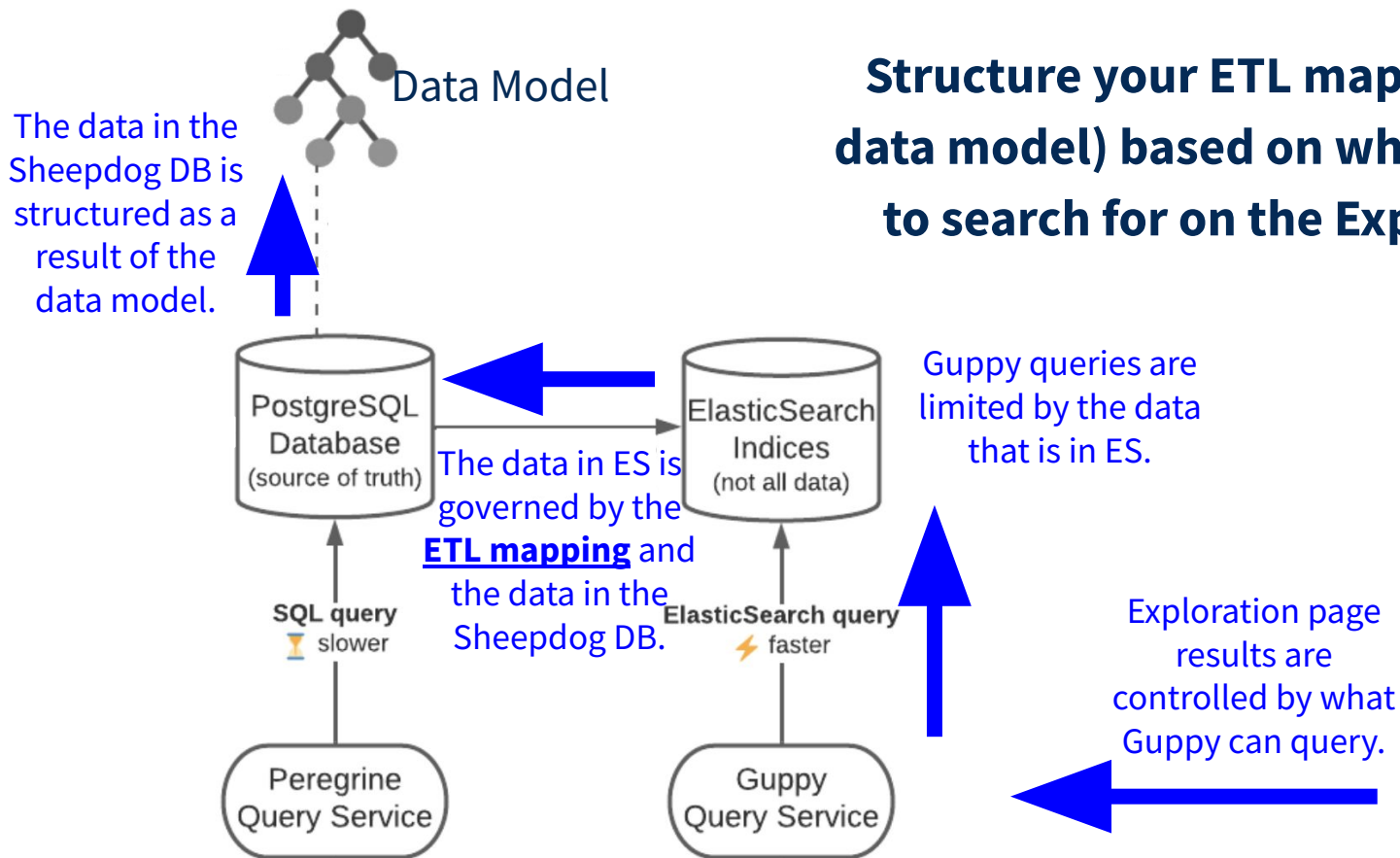
- Data submission overview
- Gen3 ETL (Tube)
  - Data model, ETL mapping, and searching for data – all connected
  - How Tube creates ElasticSearch indices for Guppy to use
  - Review of demonstration data model
  - ETL mapping types and subtypes (with examples)
  - Troubleshooting Gen3 ETL

**GEN3**  
DATA COMMONS

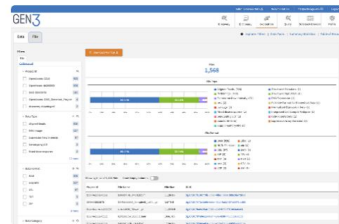


# Data Model, ETL Mapping, and Data Search

**Structure your ETL mapping (and your data model) based on what you will want to search for on the Exploration page**



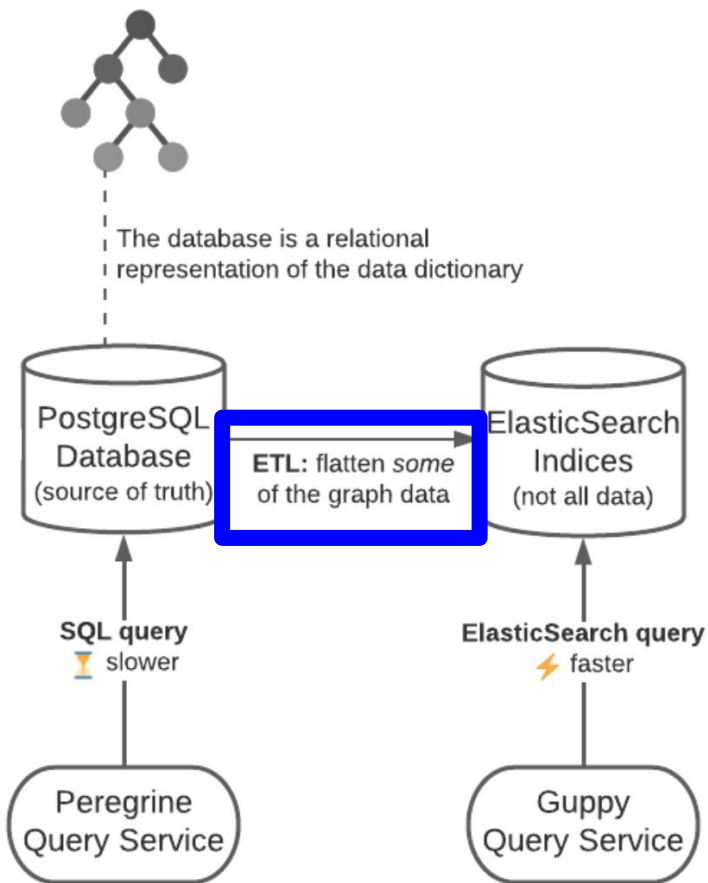
**Exploration Page**



**Each tab requires a separate index created in ETL**

The screenshot displays the MIDRC (Medical Imaging and Data Resource Center) Data Commons interface. At the top, the MIDRC logo is visible. Below it, a horizontal navigation bar contains five tabs: **Cases**, **Annotations**, **Measurements**, **Imaging Studies**, and **Data Files**. Five red arrows point to each of these tabs, emphasizing that each requires a separate ETL index. To the right of these tabs is an **Exploration** button. Below the navigation bar, there are two buttons: **Login to download table** and **Login to download file manifest for cases (1.07m)**. On the left, a **Filters** sidebar lists various categories: Annotations, Procedures, Conditions, COVID Tests, Medications, Demographics, and Imaging Studies. The main content area shows a large number **84,016** under the heading **Cases**. At the bottom, it indicates **Showing 1 - 20 of 84,016 cases** and includes a **Show Empty Columns** toggle.

# What is Tube? Why use it?



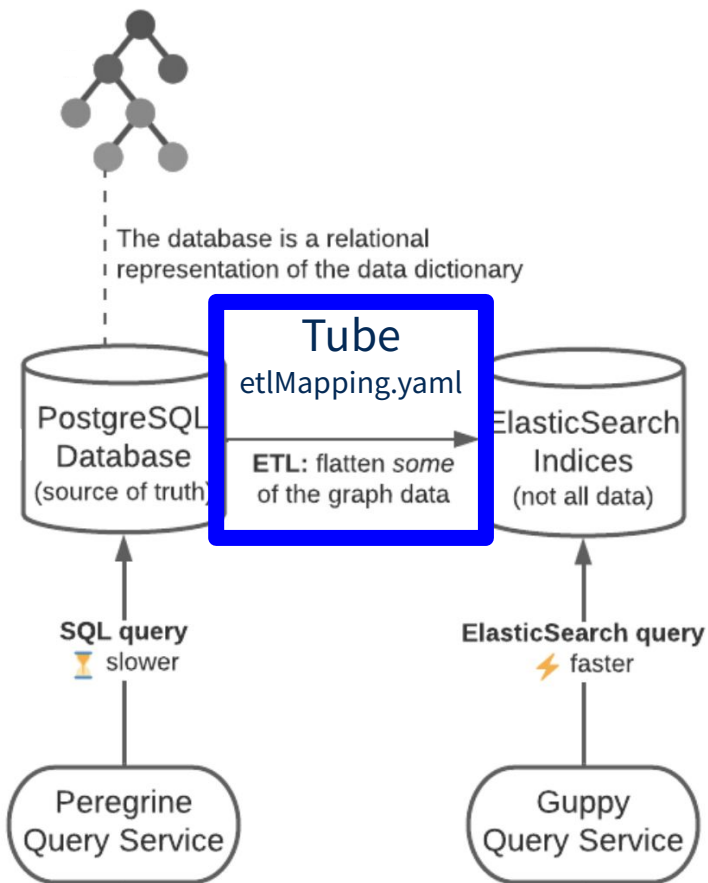
## What is Tube?

- **Tube** is the microservice that travels across the graph to find and grab data in a PostgreSQL DB, then transforms it into Elasticsearch (ES) indexed documents (indices)

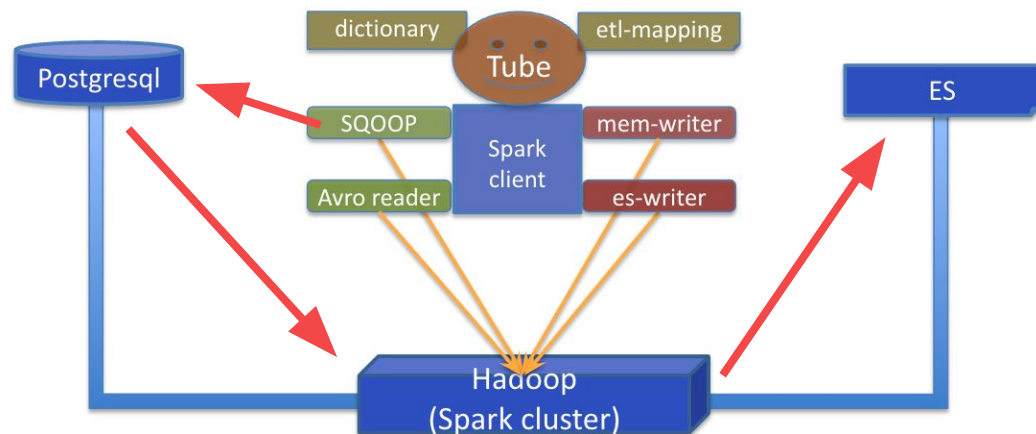
## Why use Tube (or any ETL)?

- The indices in ES allow the microservice Guppy to quickly and efficiently query data.
- Used by the front-end and other Gen3 services.

# ETL - What does Tube do?



Tube controls what data is gathered in the ES indices by configuration through the etlMapping.yaml file, which tells Tube which data tables and fields to ETL from Sheepdog into ElasticSearch indices.



## Tube (simplified)

- uses [SQOOP](#) to **extract data from the Sheepdog PostgreSQL DB**
- SQOOP temporarily dumps it into **Hadoop**
- [Apache Spark](#) then reads data from Hadoop and **transforms it** according to ETL config, **creating indices in ElasticSearch.**



```
1  etlMapping:
2    mappings:
3      - name: my-data-commons_subject
4        doc_type: subject
5        type: aggregator <or> collector
6        root: <node in the graph>
7        props:
8          - name: <property from node>
9            <prop type from root>
10         <additional mapping type>:
11           - <attributes of mapping>
```

- Yaml based schema
- “-” is the start of a new concept and indentation preserves that concept
- Multiple props can be combined for flexibility in index creation

## Mappings

### For every mapping:

- You must specify the name the index will have in ES (**name**)
- You must specify the name that Guppy will use to query the index (**doc\_type**)
- You must specify the type of mapping used for the index (**type**)

### For type: aggregator

Must also indicate **root** node name in the DB

### For type: collector

Indicate node **category** to collect properties from

```
etlMapping:
  mappings:
    - name: this_is_the_study_index
      doc_type: study
      type: aggregator
      root: study
      #<properties to be mapped>
    - name: this_is_a_file_index
      doc_type: file
      type: collector
      category: data_file
      #<properties to be mapped>
```

## Properties (props)

- Props are all the fields that are expected to be in the final ElasticSearch index

### For every property:

- You must specify the name the property will have in the ES index (**name**)
- If the prop name is different in the graph, you must also specify the name of the property in the source DB (**src**)
- You can use **value\_mappings** to map new value names to the existing values in the DB

```
props:  
- name: participant_gender  
  src: gender  
  value_mappings:  
    - f: Female  
    - m: Male
```

ETL mapping supports 6 functions (**fn**):

- **fn: count** - counting how many nodes have values for that prop per case/root
- **fn: max** - reporting the max value among records for that property for a case/root
- **fn: min** - reporting the min value among records for that property for a case/root
- **fn: sum** - reporting the sum of values for records for that property
- **fn: list** - the full set of values (including any duplicates) for records for that property
- **fn: set** - reporting all the unique values for records for that property

(you can use these in either aggregators or collectors)

## Two types of mapping:

### Aggregation (creates an aggregator)

Traveling from a single root node, aggregators gather data from properties on connected nodes into a single index in ES

**Example:** a case aggregator that collects selected clinical data properties from the case node through the clinical nodes on a data model

### Injection (creates a collector)

Collectors travel across multiple nodes of the same node category to gather data from shared properties on the nodes, and “inject” a parent node's ID into downstream nodes for faster joining

**Example:** a data file collector that collects data from properties on many different file nodes

**Aggregation** - travels from root to multiple connected nodes

## General approaches for creating an aggregator index:

With a “Root” node in mind:

- **flatten\_props:**  
Get props from lower nodes
- **parent\_props:**  
Get props from upper nodes
- **nested\_props:**  
Get props from multiple lower nodes
- **aggregated\_props:**  
Add statistics into indices
- **joining\_props:**  
Join properties between indices

**Injection** - collects props from nodes of the same category, injecting parent node ID into lower nodes

## General approaches for creating a collector index:

With a “category” of node in mind:

- **injecting\_props:**  
Properties from parent nodes are injected into an index made up of lower nodes of a specified category

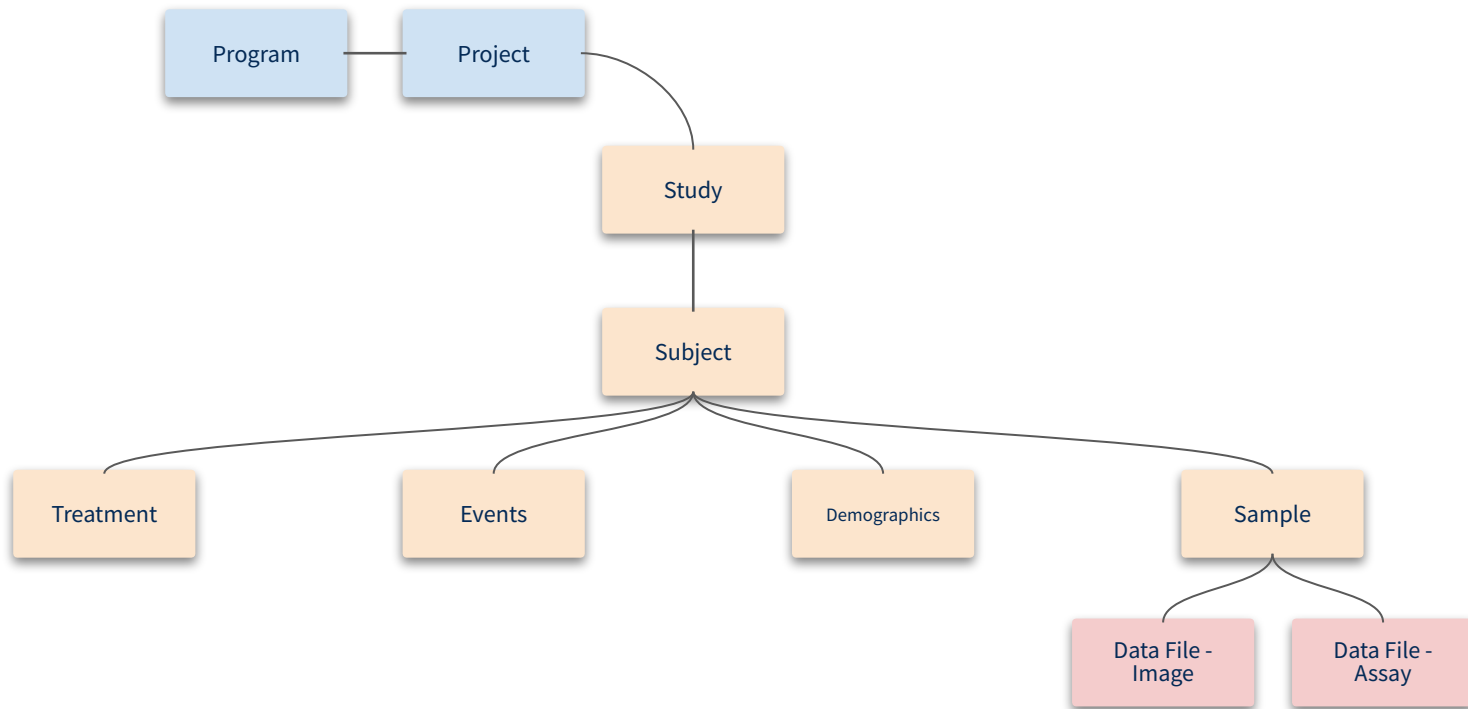
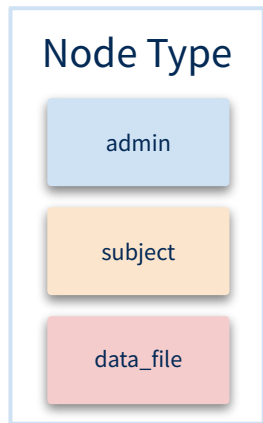
## **The following slides contain examples of each of the mappings subtypes**

- For each mapping, there is a selection of the data model
- Example Mapping
- Example Index

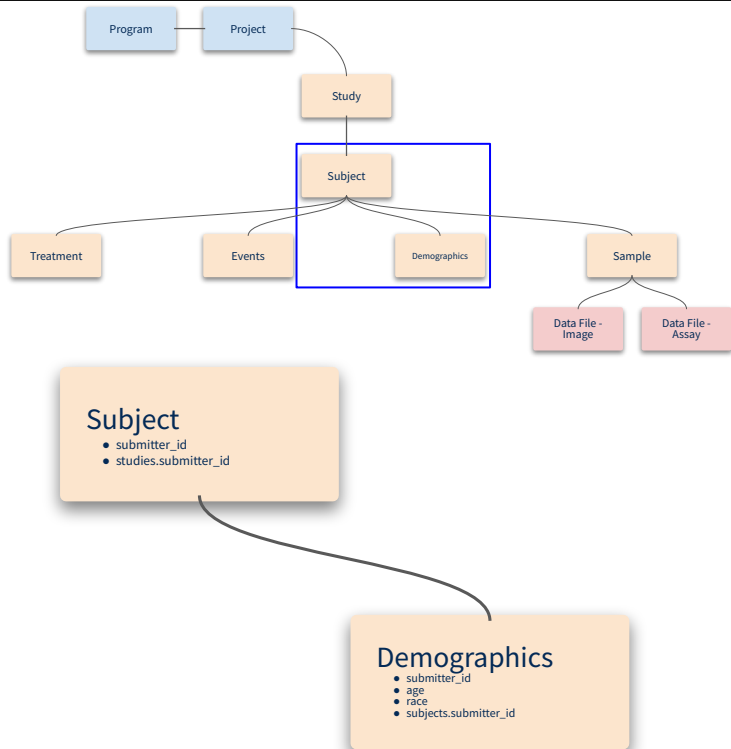
*For our purposes, indices will be displayed as tables where simple enough to do so, and as JSON where more complex.*



# ETL - Example Data Model



# ETL - Creating a Mapping - Flatten Props (one-to-one)



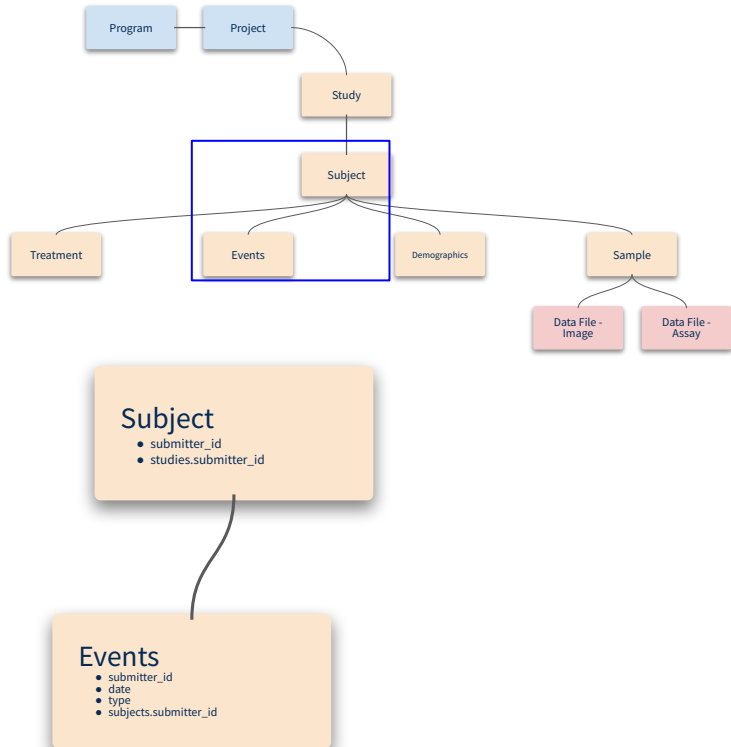
```
1 mappings:
2   - name: simple_flatten # Elasticsearch index name
3     doc_type: subject # document type - used to query the index.
4     type: aggregator
5     root: subject
6     props:
7       - name: submitter_id
8     flatten_props:
9       - name: subject_demographics
10         path: demographics
11         props:
12           - name: age
13           - name: race
```

## Example Index:

Subject.submitter_id	demographics.age	demographics.race
sub_123	18	Black or African American
sub_456	89	Asian

*Flatten\_props: Get props from lower nodes*

# ETL - Creating a Mapping - Flatten Props (many-to-\*)



```
1 mappings:
2   - name: flatten_props_multi # Elasticsearch index name
3     doc_type: subject
4     type: aggregator
5     root: subject
6     props:
7       - name: submitter_id
8     flatten_props:
9       - name: subject_events
10        path: events
11        props:
12          - name: date
13          - name: type
14        sorted_by: date, desc
```

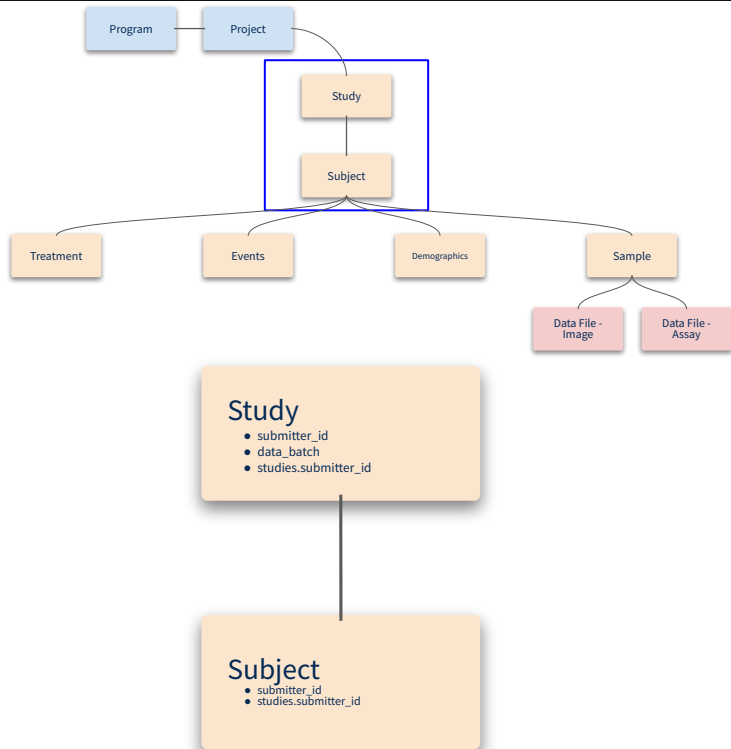
*\*Only returns the most recent date (descending order)*

## Example Index:

Subject.submitter_id	date	type
<del>sub_123</del>	<del>03/20/1996</del>	<del>birth</del>
sub_123	03/20/2009	hospitalization
<del>sub_456</del>	<del>07/26/2025</del>	<del>hospitalization</del>
sub_456	08/26/2025	death

*Flatten\_props: Get props from lower nodes*

# ETL - Creating a Mapping - Parent Props



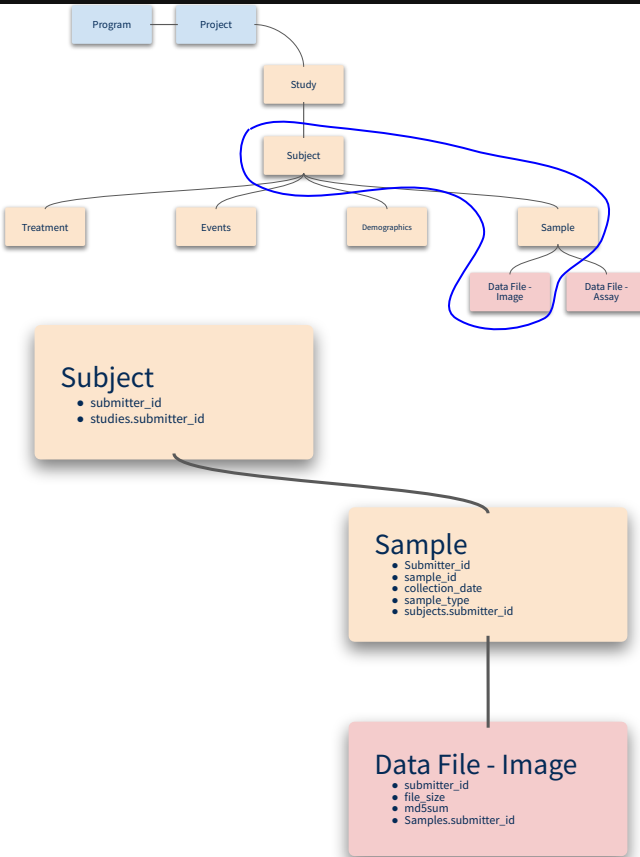
```
1 mappings:
2   - name: my-data-commons_subject # Elasticsearch index name
3     doc_type: subject
4     type: aggregator
5     root: subject
6     props:
7       - name: submitter_id
8     parent_props:
9       - name: subject_study
10        path: study
11        props:
12          - name: data_batch
```

## Example Index:

Subject.submitter_id	data_batch
sub_123	NIH_08262025
sub_456	NSRR_08272025

*Parent\_props: Get props from upper nodes*

# ETL - Creating a Mapping - Nested Props



```
1 mappings:
2 - name: my-data-commons_subject
3   doc_type: subject
4   type: aggregator
5   root: subject
6   props:
7     - name: submitter_id
8   nested_props:
9     - name: subject_samples
10      path: sample
11      props:
12        - name: submitter_id
13        - name: collection_date
14        - name: sample_type
15      nested_props:
16        - name: sample_data-file
17          path: data_file_image
18          props:
19            - name: file_size
20            - name: md5sum
```

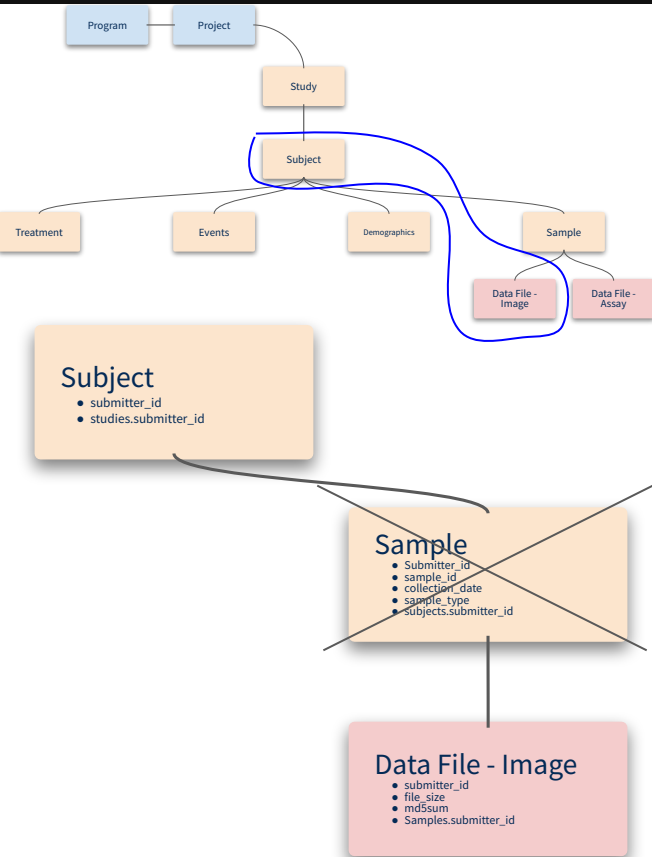
## Example Index:

```
Subject: [
  {
    "submitter id": "sub_123",
    "samples": [
      {
        "submitter id": "sample_id 123",
        "collection date": "08/26/2024",
        "sample type": "assay",
        "data files": [
          {
            "file size": "100 MB",
            "md5sum": "e33ngkdf0a0dm34fg",
            "file_size": "105 MB",
            "md5sum": "9fad9fj3jfa9dfgiafdg9"
          }
        ]
      },
      {
        "submitter id": "sample_id 456",
        "collection date": "04/28/2025",
        "sample type": "imaging",
        "data files": [
          {
            "file size": "1.53 GB",
            "md5sum": "13hgs9fdg4jgoE",
            "file_size": "2.63 MB",
            "md5sum": "31612kjfga90refj"
          }
        ]
      }
    ]
  },
  {
    "submitter id": "sub_456",
    "samples": [
      {
        "submitter id": "sample_id 789",
        "collection date": "05/28/2023",
        "sample type": "assay",
        "data files": [
          {
            "file size": "109 MB",
            "md5sum": "wvtqwsertwsn4w5",
            "file_size": "116 MB",
            "md5sum": "sfghsfvs5wg543r"
          }
        ]
      },
      {
        "submitter id": "sample_id 101",
        "collection date": "04/11/2022",
        "sample type": "imaging",
        "data files": [
          {
            "file size": "1.22 GB",
```

## Nested Indices

*Nested\_props: Get props from multiple lower nodes*

# ETL - Mapping - Nested Props (skipping nodes)



```
1 mappings:
2 - name: my-data-commons_subject
3   doc_type: subject
4   type: aggregator
5   root: subject
6   props:
7     - name: submitter_id
8   nested_props:
9     - name: subject_samples
10       path: sample.data_file_image
11       props:
12         - name: file_size
13         - name: md5sum
```

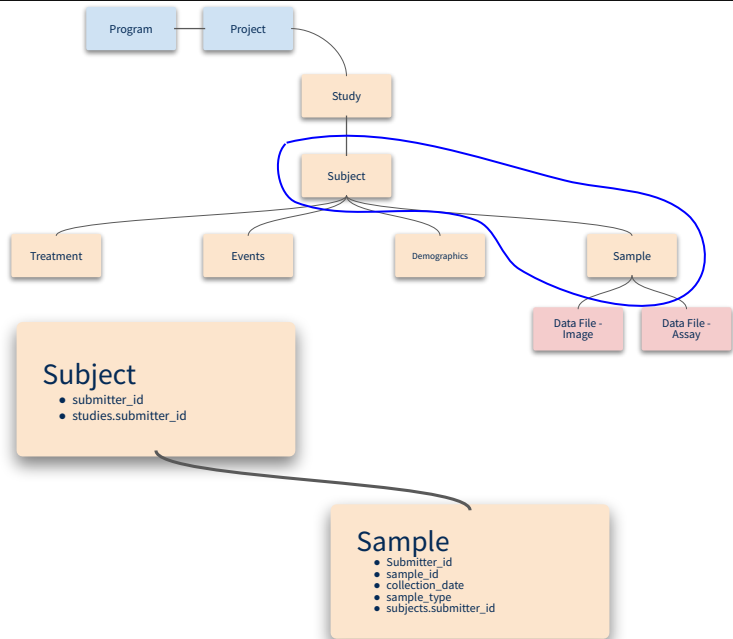
## Example Index:

```
Subject: [
  {
    "submitter_id": "sub_123",
    ("subject samples": [
      {
        "file size": "100 MB",
        "md5sum": "e33ngkdf0a0dm34fg",
        ("file size": "105 MB",
        "md5sum": "9fad9fj3jfajdfgiafdg9")
      }
    ]),
    ("subject samples": [
      {
        "file size": "1.53 GB",
        "md5sum": "13hgs9fdg4jqof",
        ("file size": "2.63 MB",
        "md5sum": "31612kjfga90refj")
      }
    ]),
  },
  {
    "submitter_id": "sub_456",
    ("subject samples": [
      {
        "file size": "109 MB",
        "md5sum": "wvtqwsertwn4w5",
        ("file size": "116 MB",
        "md5sum": "sfghsfvs5wg543r")
      }
    ]),
    ("subject samples": [
      {
        "file size": "1.22 GB",
        "md5sum": "afshgw4ttrw45q",
        ("file size": "3.11 MB",
        "md5sum": "arfg45gw5ww4rfrw4")
      }
    ]),
  },
]
```

## Nested Indices

*Nested\_props: Get props from multiple lower nodes*

# ETL - Mapping - Aggregated Props



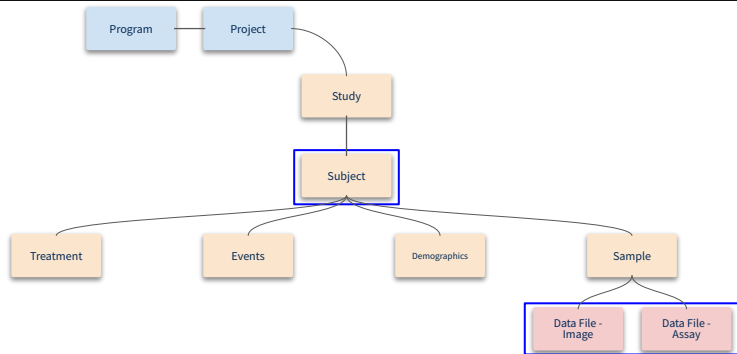
```
1 mappings:
2   - name: my-data-commons_subject
3     doc_type: subject
4     type: aggregator
5     root: subject
6     props:
7       - name: submitter_id
8     nested_props:
9       - name: subject_samples
10        path: sample
11        props:
12          - name: collection_date
13          - name: sample_type
14        aggregated_props: # used to get aggregate statistics of
15          - name: sample_count
16            path: samples # path to node from root
17            fn: count
```

## Example Index:

Subject.submitter_id	collection_date	sample_type	sample_count
sub_123	08/26/2025	assay	3
	08/26/2025	imaging	3
	09/01/2025	assay	3
sub_456	07/26/2024	imaging	3
	08/23/2025	imaging	3
	08/23/2025	assay	3

*Aggregated\_props: Add statistics into indices*

# ETL - Mapping - Injection



## Subject

- submitter\_id
- studies.submitter\_id

## Data File - Image

- submitter\_id
- project\_id

## Data File - Assay

- submitter\_id
- project\_id

```
1 mappings:
2   - name: data-file_subject
3     doc_type: data_file
4     type: collector
5     category: data_file
6     props:
7       - name: submitter_id
8     injecting_props:
9       subject:
10        - fn: set
11          name: _subject_id #note the preceding "_"
12          src: subject_id
```

## Example Index:

Data_files_image.submitter_id	_subject_id
image_123	subject_123
image_234	subject_123
image_345	subject_123
image_456	subject_345
image_567	subject_345
image_678	subject_345

Data_files_assay.submitter_id	_subject_id
assay_123	subject_567
assay_234	subject_567
assay_345	subject_567
assay_456	subject_789
assay_567	subject_789
assay_678	subject_789

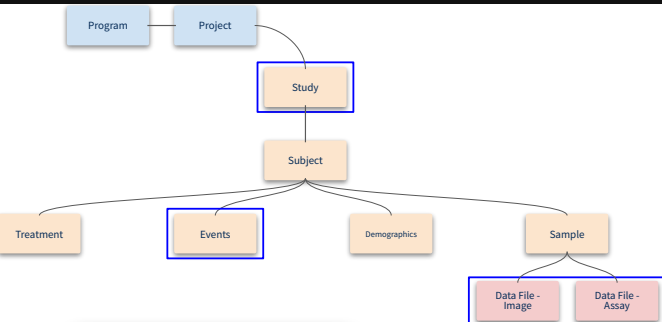
*Injecting\_props: Objects from nodes are injected into an index made up of nodes from that category*



In collectors, the default function is **set**

- **fn: set** - reporting all the unique values for records for that property

# ETL - Mapping - Joining Props (Combination)



## Study

- Submitter\_id
- Data\_batch

## Events

- submitter\_id
- date
- type
- subjects.submitter\_id

## Data File - Image

- submitter\_id
- project\_id

## Data File - Assay

- submitter\_id
- project\_id

## Injected index

```
1 mappings:
2   - name: data-file_study
3     doc_type: data_file
4     type: collector
5     category: data_file
6     props:
7       - name: submitter_id
8     injecting_props:
9       study:
10         - props:
11           - fn: set
12             name: study_id
13             src: study_id
```

## Joining indices

```
15 - name: sample_for_join
16   doc_type: subject
17   type: aggregator
18   root: events
19   props:
20     - name: event_id
21   joining_props:
22     - index: data-file_study
23       join_on: study_id
24       props:
25         - fn: set
26           name: study_id
27           src: study_id
```

## Example Index:

events.submitter_id	study_id	data_file.submitter_id
event_1	study_1	wb54yt4byw5b3
event_2	study_1	sasdffcvsfsgsrg
event_3	study_1	gd4qbg5gw545s
event_1	study_2	ds43w5bgb45try
event_2	study_2	aq4bqbtqnbq4wt4
event_3	study_2	45q4g5w5rgw545

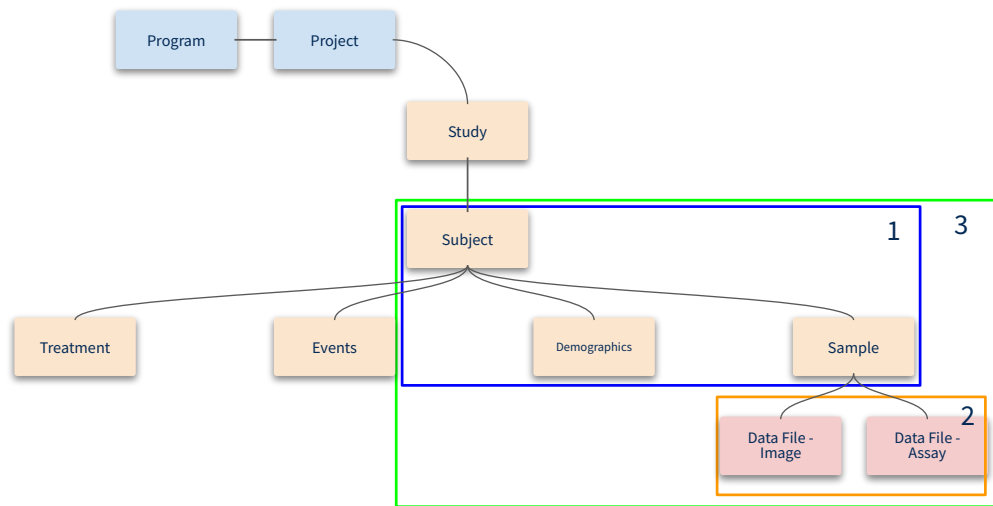
*Joining\_props: Join properties between indices*

## Goal: Create a File Manifest

We want a search to be able to capture what data files are present for studies, subjects and demographics

Steps:

1. Create an Aggregator index connecting Subject with Demographics and Sample (nested)
2. Create an collector (injected) index with the Data File nodes
3. Join the file index to the aggregator to build a file manifest\*



\* File manifests must have (at least) the `object_id` of the files to be useful. So, remember to add `object_id` to the `joining_props` of the aggregator indices

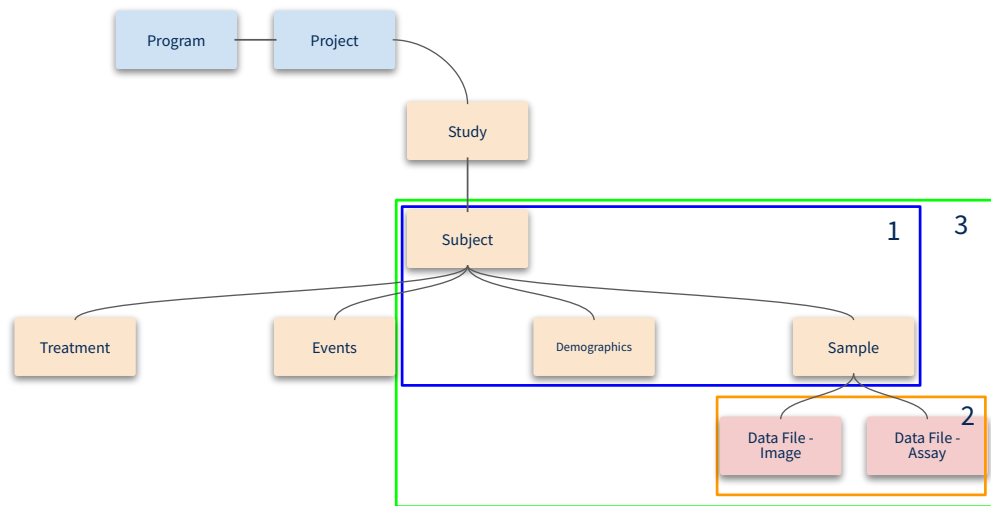
# ETL - Mapping - Putting it all together - Subject File Manifest

1.

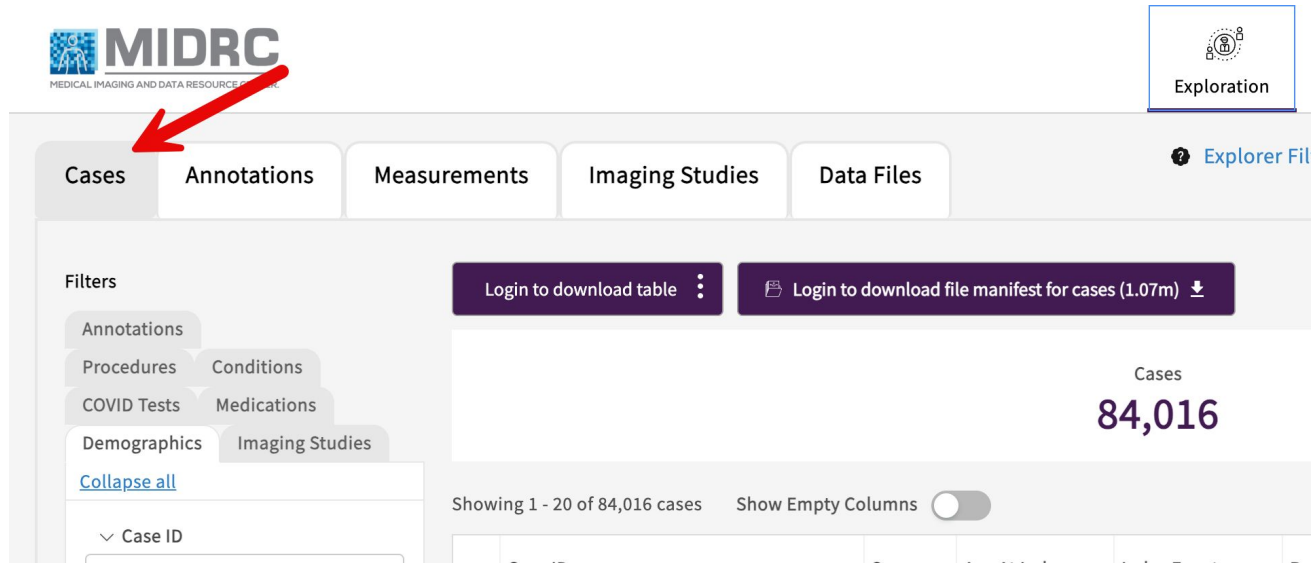
3.

2.

```
1 mappings:
2   - name: file_manifest
3     doc_type: subject
4     type: aggregator
5     root: subject
6     props:
7       - name: submitter_id
8     nested_props:
9       - name: demographic
10         path: demographic
11         props:
12           - name: race
13           - name: age
14       - name: sample
15         path: sample
16         props:
17           - name: collection_id
18           - name: sample_type
19     joining_props:
20       - index: file
21         join_on: _subject_id
22         props:
23           - name: object_id
24             src: object_id
25             fn: set
26   - name: file_collector
27     doc_type: file
28     type: collector
29     root: None
30     category: data_file
31     props:
32       - name: submitter_id
33       - name: object_id
34       - name: md5sum
35       - name: file_size
36     injecting_props:
37       subject:
38         props:
39           - name: _subject_id
40             src: id
```



**We showed you how the MIDRC exploration page has tabs created from indices generated by ETL**



The screenshot displays the MIDRC Exploration interface. At the top left is the MIDRC logo (Medical Imaging and Data Resource Center). In the top right corner, there is an 'Exploration' button with a magnifying glass icon. Below the logo, a horizontal row of tabs is visible: 'Cases', 'Annotations', 'Measurements', 'Imaging Studies', and 'Data Files'. A red arrow points to the 'Cases' tab. To the right of these tabs is a link labeled 'Explorer Filter'. Below the tabs, there are two buttons: 'Login to download table' and 'Login to download file manifest for cases (1.07m)'. On the left side, there is a 'Filters' section with a list of filter categories: Annotations, Procedures, Conditions, COVID Tests, Medications, Demographics, and Imaging Studies. A 'Collapse all' link is also present. In the center-right area, the text 'Cases 84,016' is displayed. At the bottom, it says 'Showing 1 - 20 of 84,016 cases' and 'Show Empty Columns' with a toggle switch.

# Putting it all together (Part 2)

Here is an abridged version of the midrc\_case index ETL mapping.

- Called **midrc\_case** in ES
- Called **case** in Guppy
- This is an aggregator with the case node as a root.

This index includes:

- Props
- Aggregated\_props
- Parent\_props
- Nested\_props
- Joining\_props

aggregated\_props

parent\_props  
nested\_props

joining\_props

```
etl:
  esEndpoint: elasticsearch
  etlMapping:
    mappings:
      - name: midrc_case
        doc_type: case
        type: aggregator
        root: case
        props:
          - name: project_id
          - name: submitter_id
          - name: sex
          - name: race
          - name: age_at_index
          - name: index_event
          - ... (etc)
        aggregated_props:
          - name: _imaging_studies_count
            path: imaging_studies
            fn: count
          - name: _ct_series_file_count
            path: imaging_studies.ct_series_files
            fn: count
          - ... (etc)
        parent_props:
          - path: datasets[dataset_submitter_id:submitter_id,license,data_url_doi,data_contributor]
        nested_props:
          - name: case_annotations
            path: annotations
            props:
              - name: annotation_method
              - name: annotator_id
          - name: imaging_study_annotations
            path: imaging_studies.annotations
            props:
              - name: airspace_disease_grading
              - name: class_covid19_pneumonia
              - name: annotation_name
              - name: midrc_mRALE_score
              - name: annotation_method
              - name: annotator_id
              - name: instance_uids
          - ... (etc)
        joining_props:
          - index: data_file
            join_on: _case_id
            props:
              - name: object_id
                src: object_id
                fn: set
              - name: data_format
                src: data_format
                fn: set
              - name: data_type
                src: data_type
                fn: set
              - name: data_category
```

```
etl:
  esEndpoint: elasticsearch
  etlMapping:
    mappings:
      - name: midrc_case
        doc_type: case
        type: aggregator
        root: case
        props:
          - name: project_id
          - name: submitter_id
          - name: sex
          - name: race
          - name: age_at_index
          - name: index_event
          - ... (etc)
        aggregated_props:
          - name: _imaging_studies_count
            path: imaging_studies
            fn: count
          - name: _ct_series_file_count
            path: imaging_studies.ct_series_files
            fn: count
          - ... (etc)
        parent_props:
          - path: datasets[dataset_submitter_id:submitter_id,license,data_url_doi,data_contributor]
        nested_props:
          - name: case_annotations
            path: annotations
            props:
              - name: annotation_method
              - name: annotator_id
          - name: imaging_study_annotations
            path: imaging_studies.annotations
            props:
              - name: airspace_disease_grading
              - name: class_covid19_pneumonia
              - name: annotation_name
              - name: midrc_mRALE_score
              - name: annotation_method
              - name: annotator_id
              - name: instance_uids
          - ... (etc)
        joining_props:
          - index: data_file
            join_on: _case_id
            props:
              - name: object_id
                src: object_id
                fn: set
              - name: data_format
                src: data_format
                fn: set
              - name: data_type
                src: data_type
                fn: set
              - name: data_category
```

# ETL - Mapping - Putting it all together (Part 2)

Sneak peek at how ETL mapping connects to Exploration page

```
guppy:
  ... (etc)
  indices:
    - index: midrc_case
      type: case
    - index: midrc_measurement
      type: measurement
    - index: midrc_annotation
      type: annotation
    - index: midrc_data_file
      type: data_file
    - index: midrc_imaging_study
      type: imaging_study
```

```
portal:
  gitops:
    json: |
      "explorerConfig": [
        {
          "tabTitle": "Cases",
          "charts": {},
          "filters": {
            "tabs": [
              {
                "title": "Demographics",
                "searchFields": [
                  "submitter_id"
                ],
                "fields": [
                  "sex",
                  "race",
                  "ethnicity",
                  "age_at_index",
                  "index_event",
                  "zip",
                  "covid19_positive",
                  "project_id"
                ]
              },
              {
                "title": "Imaging Studies".
```

```
"guppyConfig": {
  "dataType": "case",
  "nodeCountTitle": "Cases",
  "fileCountField": "data_file_count",
  "fieldMapping": [
    {
      "field": "project_id",
      "name": "Project ID"
    },
    {
      "field": "submitter_id",
      "name": "Case ID"
    },
    {
      "field": "imaging_studies.age_at_imaging",
      "name": "Age at Imaging"
    },
    ... (etc)
```

## Questions? (before our ETL Troubleshooting part of the presentation)

### Reminders:

- Slides and the recording will be available as a resource
- Code snippets of examples shown here are available on [GitHub](#)
- [Gen3-gitops](#) has the ETL mappings for all Gen3 commons supported by CTDS
  - At the end of this presentation, there is a slide that specifically links to ETL mapping, Guppy values.yaml, explorerConfig, guppyConfig, and exploration page for 2 different open-access Gen3 data commons
- Additional documentation is coming soon



# Troubleshooting your ETL\*

*\* And processes happening between your ETL and the data showing up on your Exploration page*

- Did you deploy the new ETL mapping?
- Did you make sure to run the ETL after deploying the new ETL mapping?
- Did you re-roll Guppy after your ETL run?
- If so – were there any Guppy errors at startup? Check both your Guppy and Portal/FEF logs to see if there are any errors (eg, new portal pod may not be deployed because of a simple JSON parsing error).
- Did you update the Portal/FEF explorer config to make sure it is aligned with any new data in ES and accessed by Guppy?

*Special thanks to Joshua Harris from Australian BioCommons for developing a substantial part of this troubleshooting guide*

## Check the ETL log *(assuming you are using Tube for ETL)*

- Will explicitly say that the ETL succeeded, even if there are lots of other warnings
- To check the ETL log:

```
kubectl logs <etl pod name> -c tube -f
```

*Special thanks to Joshua Harris from Australian BioCommons for developing a substantial part of this troubleshooting guide*

### Log on to the ES pod and check the data there after running ETL

- To log onto your ES pod:

```
kubectl exec -it <es proxy pod> -- sh
```

- To check what indices are present in your ES:

- In two terminals, do the following:

- port forward ES service:

```
kubectl port-forward svc/elasticsearch 9200:9200
```

- See data in all indices in ES:

```
curl -X GET http://localhost:9200/_cat/indices
```

- See data in a specific index:

```
curl -X GET http://localhost:9200/<index name>
```

*Special thanks to Joshua Harris from Australian BioCommons for developing a substantial part of this troubleshooting guide*

## **If your data is in ES:**

If your data is in ES - you know ETL is working and data is being created in ES

## **If your data is NOT in ES:**

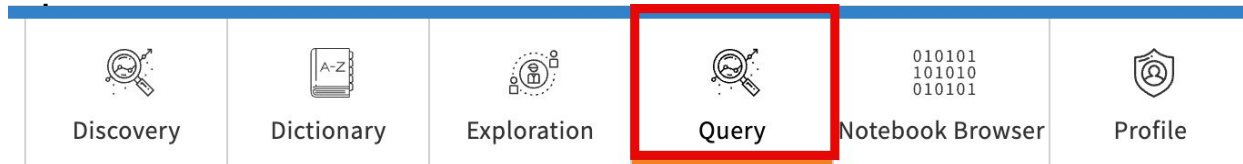
Your ETL has a problem. The ETL log should provide a clue about what's wrong - you can tell from the logging before it failed where in the ETL mapping it had the failure.

- Examine your ETL mapping; check if there's anything that could be a typo or an incorrect prop name or path.
- Look at your data in Sheepdog. Particularly look for “special characters” (backslash and non-ASCII characters), especially if it's a particular data column missing.
- Confirm with Peregrine queries that the path to the prop you included in the mapping is a valid path.

## Check the data can be queried by Guppy - Frontend

Once you know the data is in ES - restart Guppy, and check that Guppy can pull data from ES. This will help you determine whether the data in ES is queryable by Guppy from the /graphql API.

- You can work from the query page on the front end and submit your queries

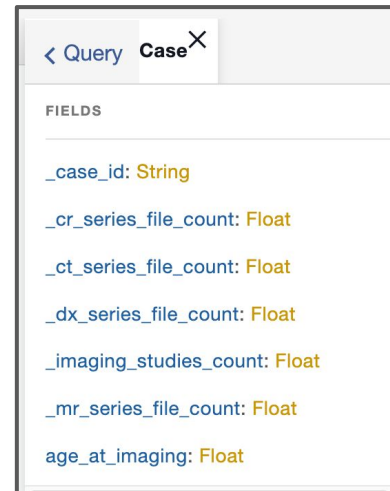
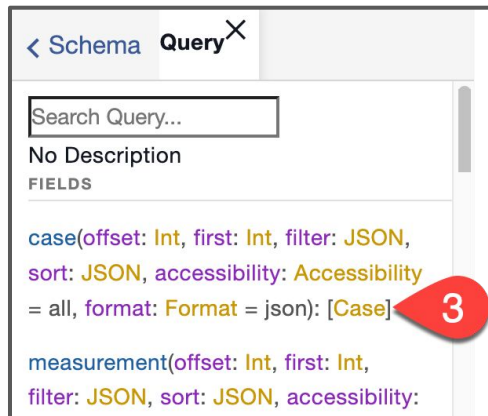
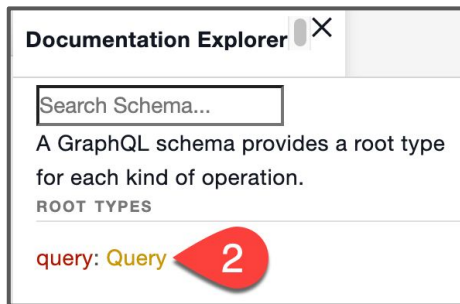
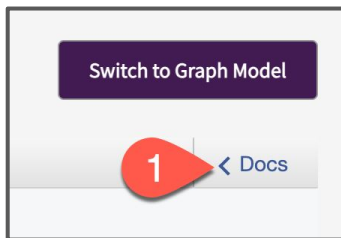


*Special thanks to Joshua Harris from Australian BioCommons for developing a substantial part of this troubleshooting guide*

## View the Guppy Schema - Frontend

Looking at the Guppy schema could help you see misalignments between ETL/ES and Guppy configuration. There are 2 ways you can view the Guppy schema from the frontend.

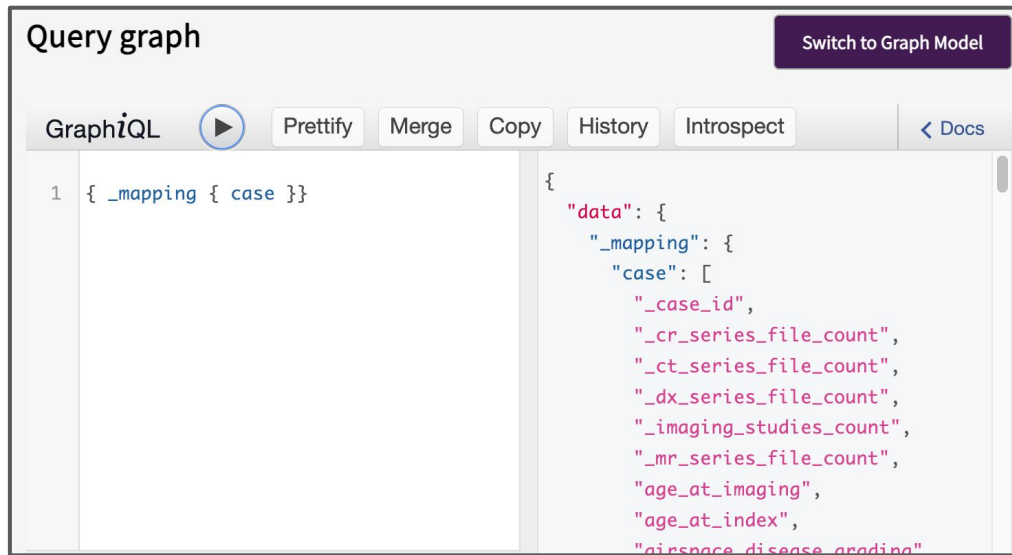
- **Through Docs:** On the Query page, click "Docs" on the upper right side of the graphiQL interface, then click "query" to view the indices available. Click on an index to view the fields available for each index.



## View the Guppy Schema - Frontend

- **Through Query page:** You can also get the guppy schema [using a "mapping" guppy query](#) on the Query page (flat model) with this query:  

```
{ _mapping { <guppy_index> }}
```



The screenshot shows the 'Query graph' interface. At the top right is a button 'Switch to Graph Model'. Below it is a toolbar with buttons: 'GraphiQL' (with a play icon), 'Prettify', 'Merge', 'Copy', 'History', 'Introspect', and '< Docs'. The main area is split into two panes. The left pane shows a query: 

```
1 { _mapping { case } }
```

. The right pane shows the JSON response: 

```
{  "data": {    "_mapping": {      "case": [        "_case_id",        "_cr_series_file_count",        "_ct_series_file_count",        "_dx_series_file_count",        "_imaging_studies_count",        "_mr_series_file_count",        "age_at_imaging",        "age_at_index",        "airspace_disease_grading"      ]    }  } }
```



## Check the data can be queried by Guppy (Backend)

- To log onto your Guppy pod:  
**kubectl exec -it <guppy pod> -- sh**
- Use curl to create a graphql query. For example, here is a sample Guppy query (*h/t Joshua Harris*)

```
curl -X POST https://<commons api>/guppy/graphql \
-H "Content-Type: application/json" \
-H "Authorization: Bearer $TOKEN" \
-d '{
  "query": "query { subject(first: 10, accessibility:
accessible) { project_id submitter_id timepoints
{ baseline_timepoint } } }"
}'
```

*Special thanks to Joshua Harris from Australian BioCommons for developing a substantial part of this troubleshooting guide*

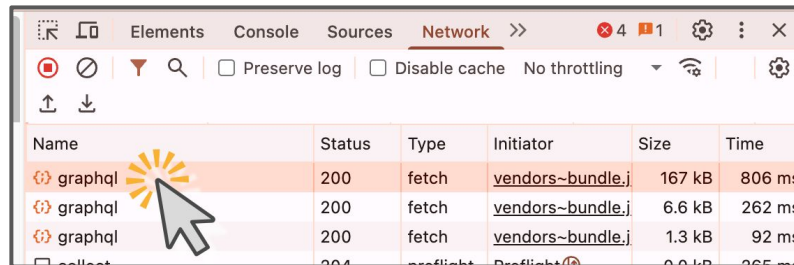
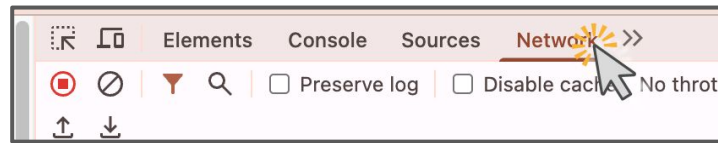
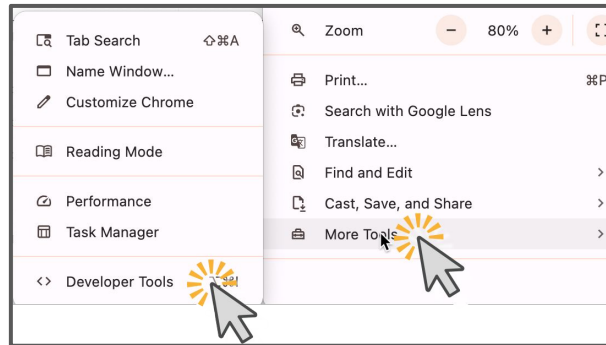
### If the data cannot be queried by Guppy:

- Guppy logs should be helpful here.
- Check your Guppy config. Make sure the index names (**type**) in the Guppy config match up, with no typos, to the **doc\_type** fields in the ETL mapping config. Check the JSON structure for problems. (Remember to re-roll Guppy after any config changes before testing if it fixed the problem).
- Check your Guppy config for appropriate **tier\_access\_level**

*Special thanks to Joshua Harris from Australian BioCommons for developing a substantial part of this troubleshooting guide*

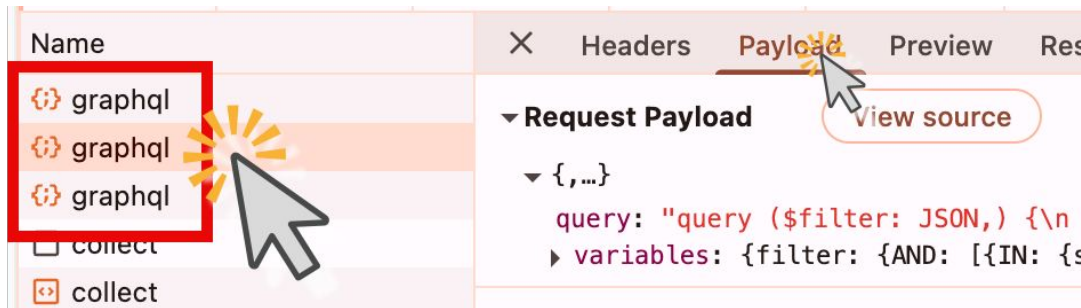
## Check the front end

1. Go to your Exploration page and then open Developer Tools
2. Click on the Network tab
3. On the Exploration page, select the filter of interest.
4. In the dev tools window, you should see /graphql API calls appear in the list. Look at the status - 200 indicates successful call, anything else indicates trouble.
5. Click on one of the graphql calls



## Look at the index the query is calling

5. Click on the Payload tab (can also click “View Source” - see box below)
6. Now, you can see the query that is being used. Check the index and make sure it's what you expect it to be.
7. Check the other graphql queries, as well.



“View source” will let you see/copy the exact query text being sent to Guppy. You use this to troubleshoot by using the Gen3 SDK to send that in a Gen3Query function (**graphql\_query()**).

## Look at CTDS open-source examples

[Gen3-gitops](#) is open access now. When in doubt, look through the etlConfig sections in the values.yaml's for any commons run by CTDS.

### MIDRC

[MIDRC ETL mapping](#)

[MIDRC Guppy indices](#)

[MIDRC Explorer config](#)

[MIDRC Guppy config](#)

[MIDRC Exploration page](#)

### Gen3 Data Hub

[G3DH ETL mapping](#)

[G3DH Guppy indices](#)

[G3DH Explorer config](#)

[G3DH Guppy config](#)

[G3DH Exploration page](#)



- Speakers

- Michael Fitzsimons - Center for Translational Data Science, University of Chicago
- Dan Biber - Center for Translational Data Science, University of Chicago
- Sara Volk de Garcia - Center for Translational Data Science, University of Chicago

- Gen3 Forum Steering Committee

- Robert Grossman - Center for Translational Data Science, University of Chicago
- Steven Manos - Australian BioCommons
- Claire Rye - New Zealand eScience Infrastructure
- Plamen Martinov - Open Commons Consortium
- Michael Fitzsimons - Center for Translational Data Science, University of Chicago

# Architecture of ETL

