Data Network Dashboards

This document is currently under construction

2021-08-05

Contents

1 Preface **5**

2 Introduction **7**

3 Installation **9**

4 Processes **13**

5 Backups **23**
5.1 Restore 24

6 Customizations **27**

7 Development Instructions **31**

8 Code Documentation **39**

9 Dashboards **47**
9.1 Per Database 47
9.2 Database Level Dashboard 57
9.3 General [Deprecated] 61
9.4 Person [Deprecated] 69
9.5 Observation Period [Deprecated] 75
9.6 Visit [Deprecated] 80
9.7 Death [Deprecated] 83
9.8 Concepts Browser [Deprecated] 87
9.9 Provenance [Deprecated] 91
9.10 Data Domains [Deprecated] 94

Chapter 1

Preface

Automated Characterization of Health Information at Large-scale Longitudinal Evidence Systems (ACHILLES) is a profiling tool developed by the OHDSI community to provide descriptive statistics of databases standardized to the OMOP Common Data Model. These characteristics are presented graphically in the ATLAS tool. However, this solution does not allow for database comparison across the data network. The Data Network Dashboards aggregates ACHILLES results files from databases in the network and displays the descriptive statistics through graphical dashboards. This tool is helpful to gain insight in the growth of the data network and is useful for the selection of databases for specific research questions. In the software demonstration we show a first version of this tool that will be further developed in EHDEN in close collaboration with all our stakeholders, including OHDSI.

Contributors

To develop this tool, EHDEN organized a hack-a-thon (Aveiro, December 2-3, 2019), where we defined and implemented a series of charts and dashboards containing the most relevant information about the OMOP CDM databases. The team involved in this task were composed by the following members:

- João Rafael Almeida[1]
- André Pedrosa[1]
- Peter R. Rijnbeek[2]
- Marcel de Wilde[2]
- Michel Van Speybroeck[3]
- Maxim Moinat[4]
- Pedro Freire[1]
- Alina Trifan[1]
- Sérgio Matos[1]
- José Luís Oliveira[1]

1 - Institute of Electronics and Informatics Engineering of Aveiro, Department of Electronics and Telecommunication, University of Aveiro, Aveiro, Portugal

2 - Erasmus MC, Rotterdam, Netherlands

3 - Janssen Pharmaceutica NV, Beerse, Belgium

4 - The Hyve, Utrecht, Netherlands

Considerations

This manual was written to be a guide for a clean installation of this system with all the dashboards that we defined during the project. The first chapter describes the goal of the system and the second how to install the system. The remaining chapters are dedicated to the dashboards, in which chapters describes one dashboard and all its charts. To simplify the representation of the dashboard's layout, we used similar schemas as it is presented in Figure 1.1. The white box is the dashboard and the inside boxes are charts. The colour changes in relation to the type of chart.

Figure 1.1: Example of a dashboards tool presenting the databases available in the network (simulated data)

License

Acknowledges

This work has been conducted in the context of EHDEN, a project that receives funding from the European Union's Horizon 2020 and EFPIA through IMI2 Joint Undertaking initiative, under grant agreement No 806968.

Chapter 2

Introduction

The OHDSI research network has been growing steadily which results in an increasing number of healthcare databases standardized to the OMOP CDM format. The OHDSI community created the ACHILLES tool (Automated Characterization of Health Information at Large-scale Longitudinal Exploration System) to characterize those databases. The results are available to the data custodian in their local ATLAS tool and helps them to gain insights in their data and helps in assessing the feasibility of a particular research questions.

ACHILLES was designed to extract the metadata from a single database, which by itself does not allow the comparison with the remaining databases in the network. However, we believe there is even more value in sharing this information with others to enable network research in a Data Network Dashboard.

Data Network Dashboard

The European Health Data and Evidence Network (EHDEN) project therefore designed a Data Network Dashboard tool, a web application to aggregate information from distributed OMOP CDM databases. It uses the ACHILLES results files to construct graphical dashboards and enables database comparison (Figure 2.1). The tool is built on Apache Superset, which is an open-source enterprise-ready business intelligence web application that can provide powerful and fully customizable graphical representations of data. Achilles results can be uploaded through the EHDEN Database Catalogue using the dashboards plugin but can also be directly uploaded in the tool. Figure 1. Example of a dashboards tool presenting age and gender distributions (simulated data).

In this tools, we defined and implemented a series of charts and dashboards containing the most relevant information about the databases, such as:

- **General**: dashboards that shows the databases types per country, the distribution of data source types, the growth of the Network including

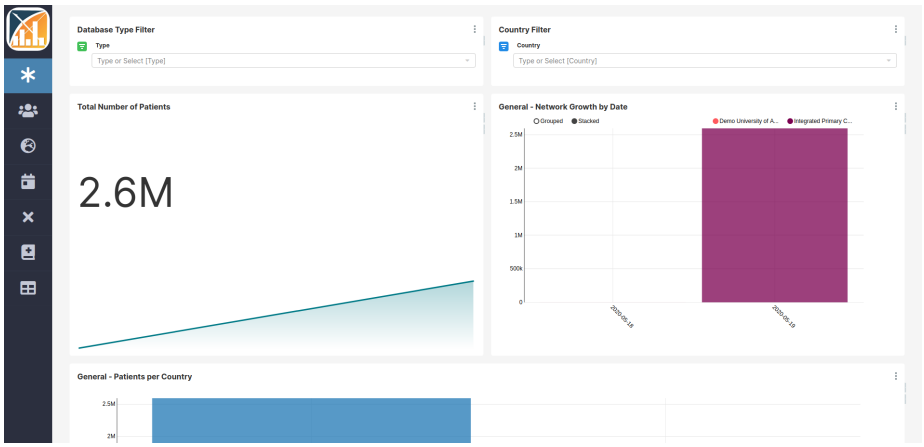


Figure 2.1: Example of a dashboards tool presenting the databases available in the network (simulated data)

the number of database and the number of patients in the databases over time;

- **Person**: representing the number of patients per country, age distribution at first observation, year of birth distribution and normalized gender distribution;
- **Population characteristics**: dashboard with the cumulative patient time, persons with continuous observation per month, and the start and end dates of those periods;
- **Visit**: chart to compare the number and type of visit occurrence records;
- **Death**: information about the number of death records by month, and the patient age at time of death;
- **Concepts**: bubble chart which shows the number of patients and records per concept over the databases;
- **Data domains**: heat map visualization of the major data domains in each database.

Chapter 3

Installation

Currently, we use docker to deploy our environment

First Steps

1. Clone the repository with the command `git clone --recurse-submodules https://github.com/EHDEN/NetworkDashboards`. If you already cloned the repository without the `--recurse-submodules` option, run `git submodule update --init` to fetch the superset submodule.
2. Create a `.env` file on the `docker` directory, using `.env-example` as a reference, setting all necessary environment variables (`SUPERSET\_MAPBOX\_API\_KEY` and `DASHBOARD\_VIEWER\_SECRET\_KEY`).

Dashboard Viewer setup

1. If you wish to expose the dashboard viewer app through a specific domain(s) you must add it/them to the `ALLOWED_HOSTS` list on file `dashboard_viewer/dashboard_viewer/settings.py` and remove the `'*'` entry.
2. Build containers' images: `docker-compose build`. This might take several minutes.
3. Set up the database and create an admin account for the dashboard viewer app: `docker-compose run --rm dashboard ./docker-init.sh`.

Insert Concepts

The concepts table is not in the repository due to its dimension, therefore we use directly the Postgres console to insert this table in the installation.

1. Get your concept csv file from Athena

2. Copy the file into postgres container

```
docker cp concept.csv dashboard_viewer_postgres_1:/tmp/
```

3. Enter in the postgres container:

```
docker exec -it dashboard_viewer_postgres_1 bash
```

4. Enter in the `achilles` database (value of the variable `POSTGRES_ACHILLES_DB` on the .env file) with the `root` user (value of the variable `POSTGRES_ROOT_USER` on the .env file):

```
psql achilles root
```

5. Create the `concept` table

```
CREATE TABLE concept (
  concept_id         INTEGER        NOT NULL,
  concept_name       VARCHAR(255)   NOT NULL,
  domain_id          VARCHAR(20)    NOT NULL,
  vocabulary_id      VARCHAR(20)    NOT NULL,
  concept_class_id   VARCHAR(20)    NOT NULL,
  standard_concept   VARCHAR(1)     NULL,
  concept_code       VARCHAR(50)    NOT NULL,
  valid_start_date   DATE           NOT NULL,
  valid_end_date     DATE           NOT NULL,
  invalid_reason     VARCHAR(1)     NULL
);
```

6. Copy the CSV file content to the table (this could take a while)

   To get both `'` (single quotes) and `"` (double quotes) on the `concept_name` column we use a workaround by setting the quote character to one that should never be in the text. Here we used `\b` (backslash).

```
COPY public.concept FROM '/tmp/concept.csv' WITH CSV HEADER
  DELIMITER E'\t' QUOTE E'\b';
```

7. Create index in table (this could take a while):

```
CREATE INDEX concept_concept_id_index ON concept (concept_id);
CREATE INDEX concept_concept_name_index ON concept (concept_name);
```

8. Set the owner of the `concept` table to the `achilles` user (value of the variable `POSTGRES_ACHILLES_USER` on the .env file):

```
ALTER TABLE concept OWNER TO achiller
```

9. Bring up the containers: `docker-compose up -d`.

10. Run the command `docker-compose run --rm dashboard python manage.py generate_materialized_views` to create the materialized

views on Postgres.

Superset setup

1. Make sure that the container `superset-init` has finished before continuing. It is creating the necessary tables on the database and creating permissions and roles.
2. Execute the script `./superset/one_time_run_scripts/superset-init.sh`. This will create an admin account and associate the `achilles` database to Superset. **Attention:** You must be in the docker directory to execute this script.
3. We have already built some dashboards so if you want to import them run the script `./superset/one_time_run_scripts/load_dashboards.sh`. **Attention:** You must be in the docker directory to execute this script.
4. If you used the default ports:
   - Go to `http://localhost` to access the dashboard viewer app.
   - Go to `http://localhost:8088` to access superset.
5. To any anonymous user view dashboards, add the following:
   - all datasource access on all_datasource_access
   - can csrf token on Superset
   - can dashboard on Superset
   - can explore json on Superset
   - can read on Chart
   - can read on CssTemplate

Dummy data

On a fresh installation, there are no achilles_results data so Superset's dashboards will display "No results". On the root of this repository, you can find the `demo` directory where we have an ACHILLES results file with synthetic data that you can upload to a data source on the uploader app of the dashboard viewer (http://localhost/uploader). If you wish to compare multiple data sources, on the `demo` directory there is also a python script that allows you to generate new ACHILLES results files, where it generates random count values based on the ranges of values for each set of analysis_id and stratums present on a base ACHILLES results file. So, from the one ACHILLES results fill we provided, you can have multiple data sources with different data.

Chapter 4

Processes

Data Sources

Target: platform user

Before uploading any data to this platform, a data source owner has to create a data source instance to then associated the upload data with.

The creation of data source is done through the `[BASE_URL]/uploader/` URL, where 7 fields are expected:

1. name: an extensive name
2. acronym: a short name
3. country: where is the data source localized
4. link (Optional): web page of the data source
5. database type: type of OMOP database
6. coordinates: a more accurate representation of the data source's localization
7. hash (Optional): the internal unique identifier of a data source

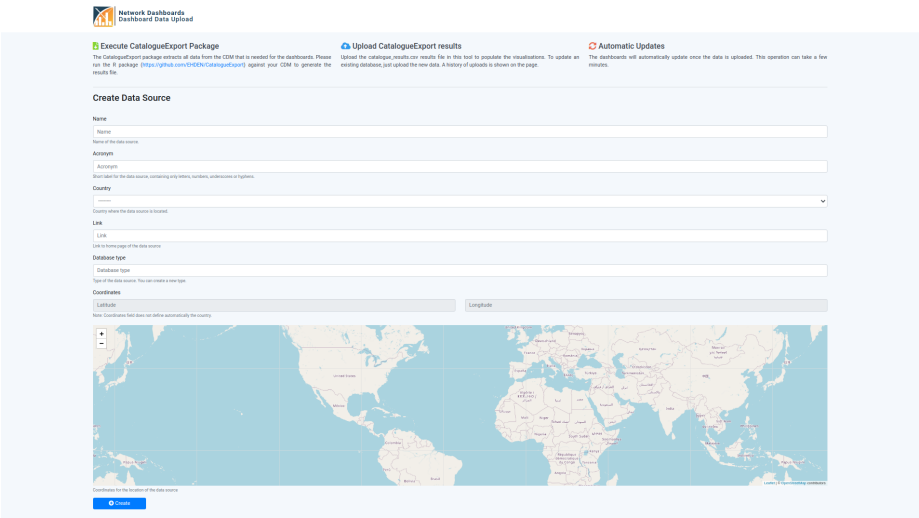
If you access `[BASE_URL]/uploader/` the 7th field (hash) is set automatically for something random, however, if you want to set it use the `[BASE_URL]/uploader/[HASH]/` URL.

To avoid duplication on the database type field, this field is transformed (use title case and trimmed) and then is checked there is already a record (Database Type) with the same value.

There are several ways to create a data source:

1. Create through a web form

By accessing the `[BASE_URL]/uploader/` URL, you will get a form where you can field the fields, where the country field is a dropdown and the coordinates field is set through a map widget.



2. Automatically create when performing a GET to the `[BASE_URL]/uploader/` URL

If the Network Dashboards platform is being used as a third-party application and the main application has all the data for the required fields, the data source can be automatically created and the user is redirected directly to the upload files page.

To perform this, each field should be provided as a URL parameter when accessing the `[BASE_URL]/uploader/` URL. If all required fields are provided and are valid the data source is created and the user is redirected to the upload files page. If a required field is missing or is not valid the webform is presented to the user so it can manually fill those fields.

3. Automatically create by performing a POST to the `[BASE_URL]/uploader/` URL

Since the creation URL does not have csrf cookie protection, you can perform a POST request as you were submitting a form.

Notes For the automatic options:

- Since the coordinates field is composed of two fields (latitude, longitude), it should be submitted as `coordinates_0=[latitude]` and `coordinates_1=[longitude]`
- The country field should match one of the available on the dropdown of the webform.

Draft Status

After a data owner uploads data into his data source, he might not want to make it public right away. To achieve this a data source has a boolean field telling

whether if the data source is in draft mode. This then also allows creating dashboards with data of non-draft data sources only.

There are three ways to change the value of this draft status field:

1. Through the Django admin app (`[BASE_URL]/admin/`)
2. Accessing the respective edit page of the data source. This requires a feature to be enabled, which is more detailed on the Allow Draft Status Updates section of the Customization chapter.
3. Perform a PATCH request to the `[BASE_URL]/uploader/[HASH]/` URL. On this request, other fields, other than the draft status, can be changed. The body of the request must be a JSON object with the fields that will suffer changes and their new values.

Catalogue Results Files

Target: platform user

Once a data source is created you can access its upload page by accessing the `[BASE_URL]/uploader/[HASH]/`. If no data source has the provided hash you will be redirected back to the data source creation form.

On the upload page you can:

1. Go to the edit page of your data source
2. Upload a catalogue results file
3. Check the upload history

A catalogue results file is a CSV file, the result obtained after running the EHDEN/CatalogueExport R package on an OMOP database. It is a variant of the OHDSI/Achilles where it only extracts a subset of analyses of the ACHILLES' original set.

The upload form expects a CSV file with the following columns:

Name	Type	Required/Non-Nullable/Non-Empty
analysis_id	int	Yes
stratum_1	string	No
stratum_2	string	No
stratum_3	string	No
stratum_4	string	No
stratum_5	string	No
count_value	int	Yes
min_value	double	No
max_value	double	No
avg_value	double	No
stdev_value	double	No
median_value	double	No
p10_value	double	No

Name	Type	Required/Non-Nullable/Non-Empty
p25_value	double	No
p75_value	double	No
p90_value	double	No

The uploaded file must:

- either contain the first 7 columns OR all 16 columns
- contain the columns in the same order as presented in the table above

While parsing the uploaded file, some data is extracted to then present on the Upload history and to update data source information. This data is extracted from the record with analysis id 0, **which is required to be present on the file**, and 5000, which is optional. Next is presented the data extracted and their description:

- R Package Version: the version of CatalogueExport R package used
- Generation Date: date at which the CatalogueExport was executed on the OMOP database
- Source Release Date: date at which the OMOP database was released
- CDM Release Date: date at which the used CDM version was released
- CDM Version: version of the CDM used
- Vocabulary Version: version of the vocabulary used

The next table is presented where the previous data is stored on the rows with analysis id 0 and 5000:

Analysis Id	Stratum 1	Stratum 2	Stratum 3	Stratum 4	Stratum 5
0		R Package Version	Generation Date		
5000		Source Release Date	CDM Release Date	CDM Version	Vocabulary Ver

Materialized Views

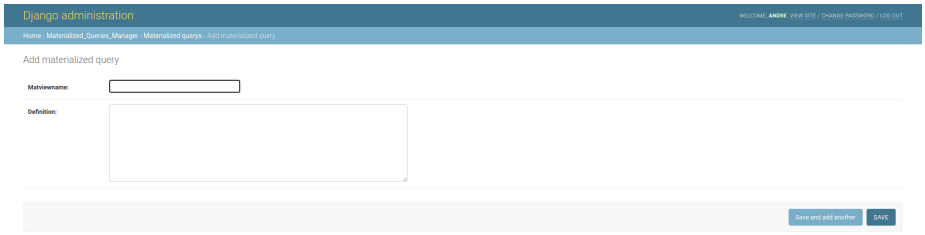
Target: admin user

For each chart, Superset has an underlying SQL query which in our case is run every time a chart is rendered. If one of these queries takes too long to execute the charts will also take too long until they are rendered and eventually users might get timeout messages given a bad user experience.

To avoid this problem, instead of executing the raw SQL query we create a postgres materialized view of the query, which is then used to feed the data to the chart. So only a simple `SELECT x FROM x` query is executed when a chart

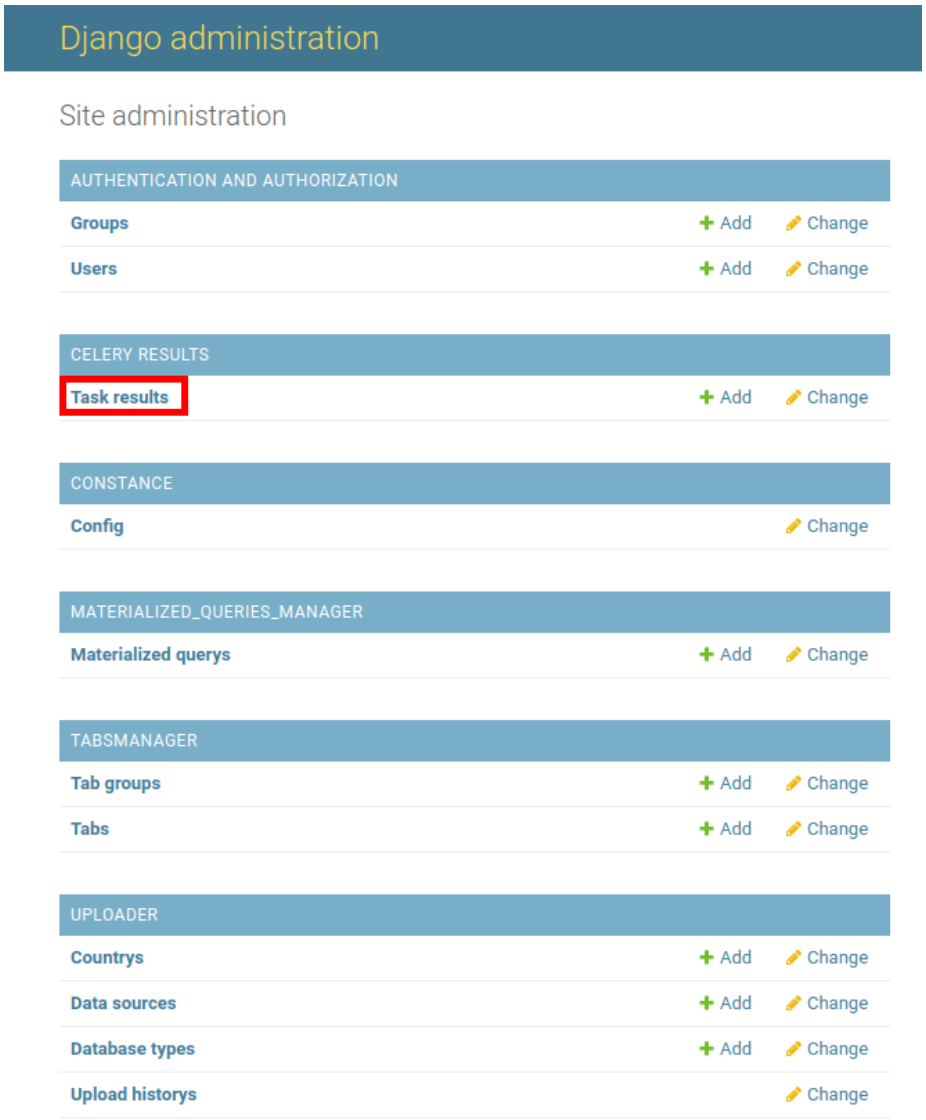
is rendered.

So whenever I create a chart I have to access the Postgres console? No, we created an unmanaged Materialized Queries model that maps to the materialized views on Postgres. With it you can create new materialized views through the Django admin app, by accessing the `[BASE_URL]/admin/` URL.

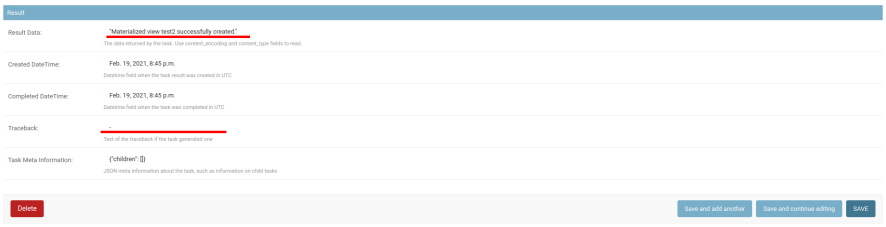


You have to provide the materialized view name and its query, which will then be used to execute the query `CREATE MATERIALIZED VIEW [name] AS [query]`, which will be executed on a background task so the browser doesn't hang and times out, in case of complicated queries. Taking this into account, the record associated will not appear on the Django admin app until the `CREATE MATERIALIZED VIEW` query finishes.

To give feedback on the background task we use celery/django-celery-results, so you can check the status of a task on the Task Results model of the Celery Results app



After the creation of a Materialized Query, the will be a message telling the id of the task which is executing the `CREATE MATERIALIZED VIEW` query. You can then check for the record associated with the task, click on the id to get more details. If something went wrong check the error message either on Result Data or Traceback fields under the Result section



After all this, the final step is to add the materialized view as a Dataset. Login into Superset, then go to Data -> Datasets and create a new one. Select the `Achilles` database, the `public` schema, then the created materialized view and click "ADD". After this, the materialized view can be used as a data source for a new chart.

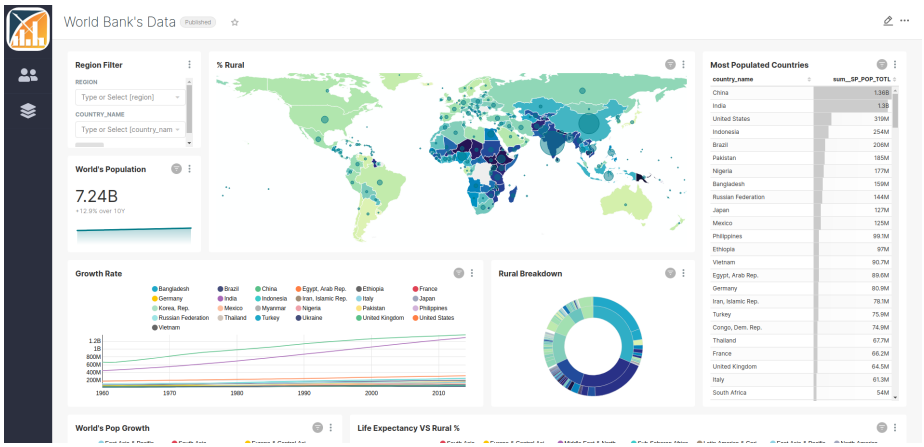
Tabs View [Deprecated]

Note: This app is no longer maintaned and the associated urls were unlinked.

Target: admin user

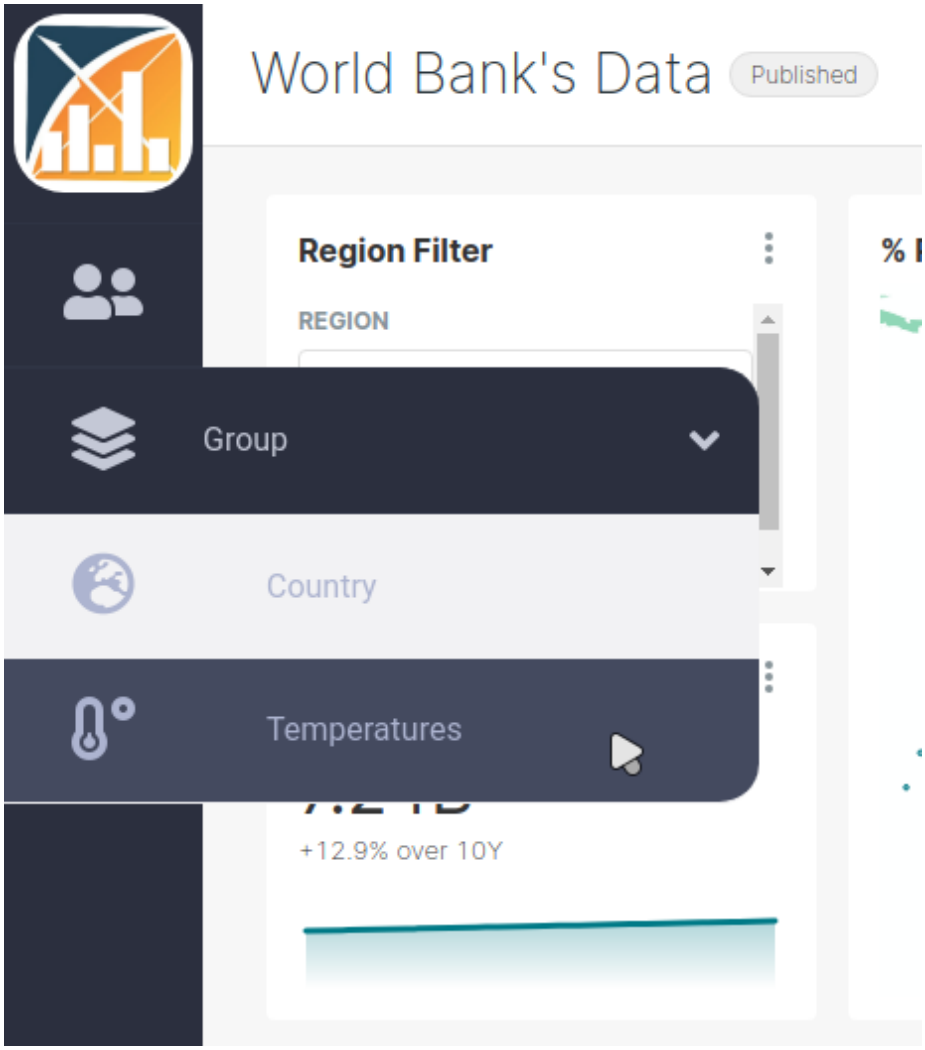
Once there are data sources on the platform, data was uploaded to them and there are dashboards created on Superset, researchers can now browse through the dashboards and analyze and compare the data of the different data sources. One way to allow this would be to let them browse through the dashboard list on Superset. However, if there was some dashboards not ready to show to the public users, they could still access them.

For that, it was created a page, with a sidebar, where public users could browse through the available and ready dashboards. It can be accessed through the URL `[BASE_URL]/tabs/`



The sidebar entries can be configured through the Django admin app, accessing the Tabsmanager app section. Here two models are available to create:

- Tab Groups: They allow to groups several sidebar entries within a collapsable group.



- Tabs: Will create a clickable entry on the sidebar that can be presented within a group. When a tab is clicked the associated dashboard will be displayed on the page.

Each entry, tab, or group of them, expects:

- Title/Name
- Icon: Name of a font awesome version 5 icon
- Position: Allows to order entries along the sidebar. If a Tab has a group, then this field will order the tabs within that group only.
- Visible: If whether or not this tab or group should be visible. The goal

of this field is to avoid having to delete the record from the database just because a certain tab is not ready and later on created it from scratch.

Tabs additionally expect an URL, which will be used to display a Superset dashboard in an iframe. To hide Superset's menu bar, an additional `standalone` URL parameter should be appended to the provided URL of a tab. The value of the `standalone` arguments depends on the expected result:

- 1: menu bar is hidden. the bar where the dashboard title, publish status, and three dots option menu are present will still appear
- 2: both the menu bar and the dashboard title bar are hidden.

By default, the dashboard of the first tab is displayed on the page, however, if one wants a specific tab to be displayed when the page is opened, its title should be present in the hash part of the URL. For example, if there is a tab called People, to make that tab selected at the start the following URL should be used `[BASE_URL]/tabs/#People`.

Chapter 5

Backups

1. Create a credentials file (the structure of the file depends on the target cloud server)
2. Create a `.dashboards_backups.conf` file under your home directory (variable `$HOME`) using `dashboards_backups.conf.example` as base, setting the appropriate value for the several variables.

   For variables associated with files and directories always use *absolute* paths.

   Variables:

   - `RUN`: Set it to `0` if you don't want the next scheduled backup to run.

     This variable allows you to cancel any backup runs while you are doing some maintenance on the application.

   - `CONSTANCE_REDIS_DB`: Number of the Redis database where the django constance config is stored. The default value is 2. This value should be the same as the environment variable `REDIS_CONSTANCE_DB` of the dashboard container.
   - The following variables are associated with the arguemtns of the `backup_uploader` python package. Check its usage for more details:
     - `APP_NAME`: The backup process will generate some directories with this name in places that are shared with other applications.
     - `SERVER`: The name of the target cloud server to where backups should be uploaded (dropbox or mega).
     - `BACKUP_CHAIN_CONFIG`: Allows having different directories with backups of different ages.

   - `CREDENTIALS_FILE_PATH`: File containing the credentials to access the server to upload the backup file.

3. Install the `backup_uploader` python package by following its install instructions.

4. Schedule your backups

```
*    *    *   *    *  Command_to_execute
|    |    |    |   |
|    |    |    |    Day of the Week ( 0 - 6 ) ( Sunday = 0 )
|    |    |    |
|    |    |    Month ( 1 - 12 )
|    |    |
|    |    Day of Month ( 1 - 31 )
|    |
|    Hour ( 0 - 23 )
|
Min ( 0 - 59 )
```

   (Retrived from: Tutorialspoint)

   Ex: To run every day at 3:00 am

   1. `crontab -e`
   2. Add entry `0 3 * * * $HOME/NetworkDashboards/backups/backup.sh` (The path to the backup script might be different)

5.1 Restore

1. Select the compressed backup you want to restore and decompress it:

   `tar -xJf BACKUP_FILE.tar.xz`.

2. 1. **Redis**

      1. Make sure the redis docker container is down.
      2. (Re)place the file `dump.rdb` on the redis volume by the file `redis.rdb`. By default the redis volume is located where this repository was cloned on the directory `docker/volumes/redis`.
      3. Change its permissions, owner and group:

         ```
         chmod 0644 docker/volumes/redis/dump.rdb
         sudo chown -R 999:999 docker/volumes/redis
         ```

   2. **Postgres**

      1. Make sure all containers that make changes on the database are stopped.

2. Copy the file `postgres_backup.sql` into the postgres container

   `docker cp postgres.sql [CONTAINER_ID]:/tmp`.

3. Execute the backup script:

   `docker exec -u root dashboard_viewer_postgres_1 psql -f /tmp/postgres_backup.sql -U \$POSTGRES_USER -d \$POSTGRES_DB`.

3. **Media Files** If you have a volume pointing to where the media files are stored, replace all files with the ones present on the downloaded backup file. Else:

   1. Bring the dashoard container up `docker-compose up -d dashboard`
   2. Enter in the container `docker exec -it [CONTAINER_ID] bash`
   3. If you don't know where the media files are stored you can check the value of the MEDIA_ROOT variable
      1. `python manage.py shell`
      2. `from django.conf import settings`
      3. `print(settings.MEDIA_ROOT)`
   4. Remove the entire MEDIA_ROOT directory and exit the container
   5. Copy the media directory present on the backup file to the catalogue container `docker cp -a collected-media [CONTAINER_ID]:[MEDIA_ROOT_PARENT_PATH]`

Chapter 6

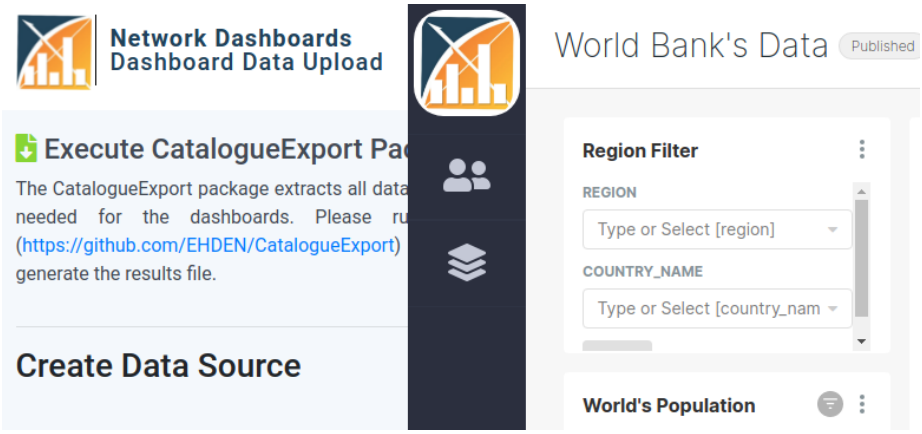
Customizations

This platform is currently being used within the scope of the European Health Data & Evidence Network (EHDEN) project. To allow the dashboard viewer Django application to be easily used by another project or company, several components support customization in runtime, removing the need to change such things directly on the source code.

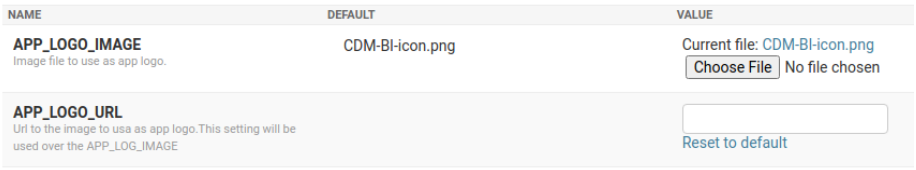
To achieve this we make use of Constance that allows configuring several fields which then can be changed through the Django admin app.

Platform Logo

It is visible both in the Tabs Manager and the Catalogue Results Uploader URLs.

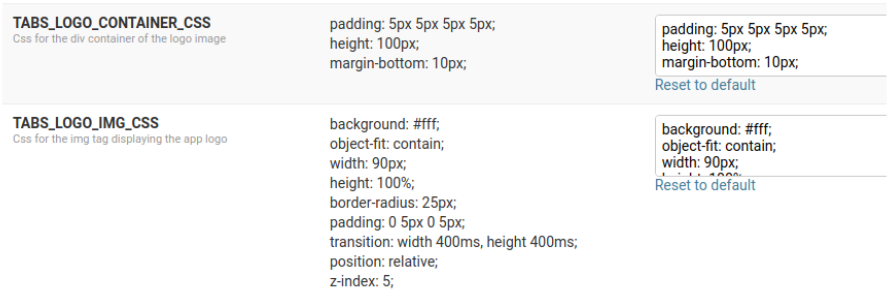


The platform allows two possible ways to choose a logo: upload a file or provide an URL to an image.



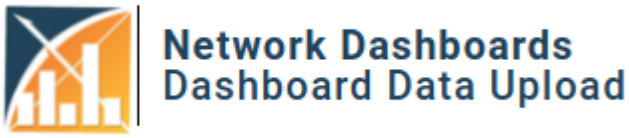
If both fields are provided, the URL one will be used.

On the tabs manager app, we also allow customization of the CSS associated both with the image itself and its container.

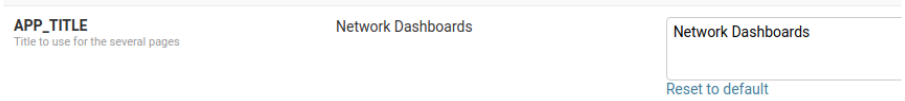


Platform Title

All pages of the uploader app use the same base HTML file which contains a header with the platform logo, page title, and platform title.

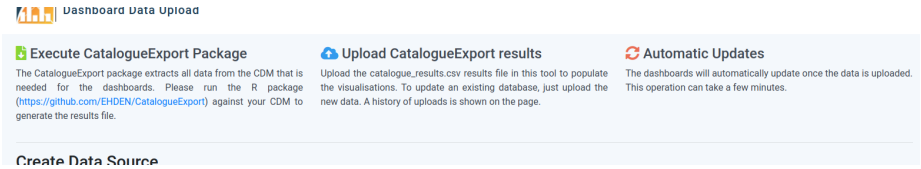


The first was already mentioned before, the second can't be changed. The last can be altered using a Constance field.

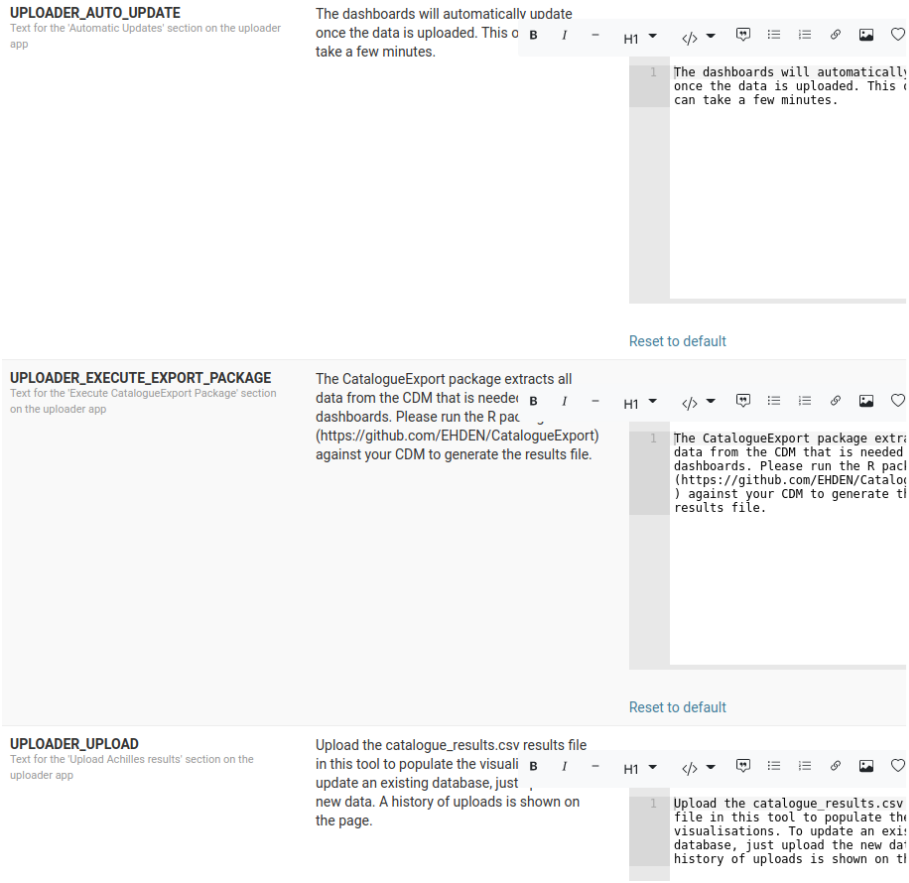


Uploader Page Texts

The data source creation page has three columns with some text providing some instructions for the creation of a data source and the upload of catalogue results.



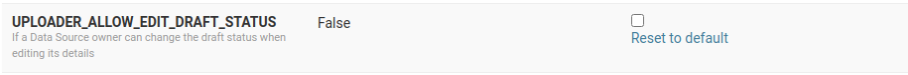
The text of these three columns is customizable, where markdown can be used, which is then transformed into HTML before rending the page.



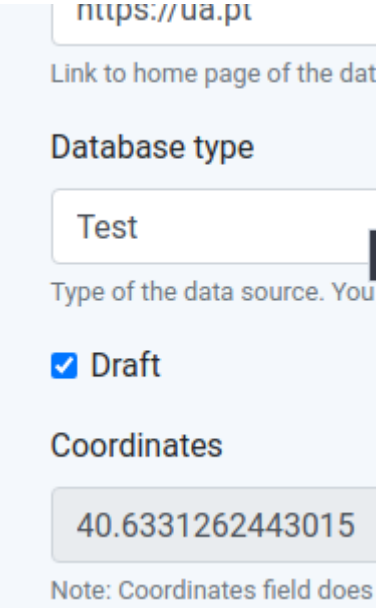
Allow Draft Status Updates

In the section Draft Status of the Processes chapter, it was already explained the concept around draft data sources.

By default, a user can NOT change the data source status on the edit page of a data source, only being allowed to do it through a PATCH request. Changes through the web edit form can be allowed by changing a Constance field.



Them an additional draft field will be available on the edit data source form.



Chapter 7

Development Instructions

Repository Structure Description

- **backups**: Scripts and configuration files to perform backups of all the data involved in the Network Dashboards applications (Dashboard viewer + Superset)
- **dashboard_viewer**: The Dashboard Viewer Django application to manage and upload catalogue results data. More detail can be found in the Code Documentation chapter.
- **demo**: Files that can be used to test some processes of the platform (Upload catalogue results data and import a simple dashboard)
- **docker**: Docker-compose stack-related directories.
  - Environment file
  - Configuration directories (Nginx and Postgres)
  - Custom Superest Dockerfile

  For more information about docker deployment consult the Installation chapter.
- **docs**: Where the files of this gitbook are hosted. Other output formats can also be obtained here. Consult the Documentation section of this chapter for more details.
- **superset**: contains a submodule to the latest supported version of Superset's repository and our custom chart plugins
- **tests**: contains files to launch a docker-compose stack specific to run tests.
- **requirements-dev**: python requirements files to the several tools to either perform code style checks or to run Django tests

- **.pre-commit-config.yaml**: configuration for the pre-commit tool. This is not mandatory to use but is a good tool to automatically fix problems related to code style on staged files
- **setup.cfg**: configurations for the several code style tools
- **tox.ini**: configuration for the tox tool. It helps automate the process to check if the code style is correct and if the Django tests are passing

  It's extremely useful in this context since different code style check tools that we use have some conflicts with python dependencies. It creates a virtual environment for each tox environment, in our case, for each code style check tool plus Django tests

Superset

Currently, we have a custom chart plugin on our superset installation which doesn't allow us to use superset's pre-built images available on their docker hub, since we have to call npm's build procedures on the front-end code. To build our custom docker image we used superset's Dockerfile as a base, where we removed the Dev section and added some code to install our chart plugins before building the front-end code. Also, to make Superset import our custom chart plugins, some changes have to be made to the superset-frontend/src/visualizations/presets/MainPreset.js file.

The changes made to the Dockerfile to install the chart plugins are in this area:

1. L44: First we copy the `superset/plugins` directory into the container, which contains all the extra and custom chart plugins.
2. L48-51: Then we iterate over the chart plugins and execute `npm install ...` on each of them. This will make changes to both the package.json and package-lock.json files and for that, we copy them into a temporary directory `package_json_files`.
3. L54: Then all superset's front-end code is copied into the container, which will override the package*.json files.
4. L56: After this, we copy our custom MainPresets.js file.
5. L60-L63: Finally, we replace the package*.json files with the ones that we saved earlier and then run the npm build command.

Update Superset

1. Update repository's tags `git fetch --tags`.
2. `cd` into superset's submodule directory and checkout to the new desired release tag.
3. Check if there are any changes made to superset's Dockerfile (on the root of the repository for the current latest release), adapt them, and insert them on our custom Dockerfile under the `docker/superset` directory.

4. Check if there are any changes made to superset's `superset-frontend/src/visualizations/presets/MainPre` file. You can use the script `mainpreset_has_changes.py` under the `plugins` directory to check that. Apply the new changes, if any, and remember to keep our chart plugins imported and registered (Currently we only have the *Box plot* plugin).
5. If the version of the frontend package `@superset-ui/plugin-chart-echarts` changed it's necessary to update our box plot plugin. Follow the instructions present here, also take into account the instruction of the next section.

Chart Plugin Development

Instructions on how you can set up your development environment to develop a custom superset chart plugin:

1. Clone the superset repository. **IMPORTANT NOTE**: Since we build the superset's docker image using the existing superset's submodule, it's better not to use it to develop the plugins. If you decide to use it anyways, remember this and this steps. They might override directories (`superset-frontend/node_modules` and `superset/static/assets`) that are generated during the build process, which can cause frontend compilation errors or the app can serve outdated static files.
2. Clone the superset-ui repository into the directory superset-frontend of the superset's repository.
3. Follow the instructions of this tutorial to create the necessary base files of your plugin.
4. Copy the file `MainPreset.js` present on this directory into the superset repository into the `superset-frontend/src/visualizations/presets/` directory.
5. Add the line `npm install -f --no-optional --save ./superset-frontend/superset-ui/plugins/plugin` into the file `docker/docker-frontend.sh` of the superset repository before the existing `npm install ...` commands.
6. When the development is finished, on the root of the superset-ui repository run `yarn install` and then `yarn build [your-chart-name]`.
7. Copy the directory of your plugin (including its sub-directory `esm`), within the superset-ui repository within the directory `plugins`, into the sub-directory `plugins` this directory. Make sure to run the command `yarn build [your-chart-name]` before doing this step.

Important features

1. Standalone Mode: by appending `?standalone=true` to the URL of a dashboard superset's menu bar won't show. New versions support

`?standalone=1` or `?standalone=2` where the first does the same as `?standalone=true` and the second also hides the bar containing the name of the dashboard, leaving just the charts.

2. Filters:

- check this faq entry
- Append `?preselect_filters={"chartId":{"columnToFilterBy":["value1", "value2"]}}` to the dashboard URL to apply a filter once the dashboard is loaded. E.g. `?preselect_filters={"13":{"name":["Demo University of Aveiro"]}}`

3. Custom label colors: check this faq entry

Github Actions

Github has a feature that allows performing automatic actions after a certain event happens on the repository. We use this feature to execute to check if everything is alright with new PR before merging them to dev.

Github calls a job a set of steps that are executed after a certain event. Then several jobs can be groups in workflows. Events are defined at the workflow level, so all the jobs in a workflow will execute at the same time.

We have two workflows:

1. Code analysis checks
2. Django tests

The first has three jobs

1. black: ensures that python's code format is consistent throughout the project
2. isort: sorts and organizes import statements
3. prospector: executes a set of tools that perform some code analysis

The second has just one job that executes the Django tests.

Both workflows execute on commits of pull requests that will be merged into the dev branch.

Regarding the code analysis workflow, the three tools used have requirements that conflict with each other, for that there is a requirements file for each tool on the `requirement-dev` directory of the repository. To avoid having three different virtual environments for each tool, you can use the tox. You just need to install the development requirements (`pip install -r requirements-dev/requirements-dev.txt`) and then just run `tox`. It will manage the necessary virtual environments and install the requirements for each tool. If you, however, want to run a specific tool manually you can check the tox configuration file (tox.ini). For example for the prospector tool the tox configuration is the following:

```
[testenv:prospector]
basepython = python3.8
deps =
    -r{toxinidir}/requirements-dev/requirements-prospector.txt
    -r{toxinidir}/dashboard_viewer/requirements.txt
commands =
    prospector dashboard_viewer
    prospector docker/superset
```

we can see that it installs the requirement for the prospector tool and also the requirements of the Dashboard Viewer Django app and then runs two commands.

For both black and isort tools, when you run tox, it will show the necessary changes that are required to make the code good. You can apply the changes automatically by executing the tools manually without the `--check` and `--check-only` options respectively.

Sometimes prospector can be a pain in the boot, complaining about too much stuff. You can make prospector ignore some bad stuff by adding the comment, `# noqa`, to the end of the specific line where it is complaining.

Tests

Our tests use Django's building testing features, which uses unittest under the hood. Not all featured have tests associated, however, there are already some tests scenarios in mind written as issues on the repository, which have the tag **Test Use Case**.

To run the tests we set up a docker-compose stack, under the test directory which has just the necessary data containers (Redis and Postgres) to avoid having to make changes on the development/production docker-compose stack. Once the stack is up it only necessary to run `SECRET_KEY=secret python manage.py test` to execute the tests. If you are developing any tests that involve celery, there is no need to have a celery process running, since on Django's settings.py we set the test runner to the celery one. This way the `python manage.py test` is enough to test the whole application.

Python Requirements

The python requirements for the Dashboard Viewer Django app are present on the `requirements.txt` file of the `dashboard_viewer` directory. The file is divided into two sections. First are the direct dependencies. Dependencies that are directly used or imported by the Dashboard Viewer Django app. For better maintainability, every direct dependency has a small description in front of it, so any developer knows why it is being mentioned in the requirements file. The second part of the file contains the indirect dependencies. Basically dependencies of our direct dependencies.

After any update is made to the direct dependencies the following procedure should be followed:

1. Create a new virtual environment just for the dependencies of this file
2. Delete the indirect dependencies section of the file
3. Install all the direct dependencies `pip install -r requirements.txt`
4. Append the result of pip's freeze to the requirements file `pip freeze >> requirements.txt`
5. Remove from the second section of the file, duplicated entries of the first section of the file, in other words, remove from the indirect dependencies section the direct dependencies.

With #185 we intend to start using the pip-compile tool. With it, we can have a file with the direct dependencies (requirements.in), and then pip-compile reads that file and automatically creates a requirements.txt file with all the dependencies and which package requires that specific dependency. The update process of dependencies will then just be

1. Install the pip-compile tool `pip install pip-tools`
2. Make the change to the direct dependencies on the requirements.in file (No need for a virtual environment)
3. Call the pip-compile tool on the requirement.in file `pip-compile requirements.in`

Documentation

The plan is to have all the documentation on this git book and any other places that might require some description/information should point to this GitBook so we maintain a commonplace for all the documentation. This way we can make sure that the code and the documentation are in the same place since on a pull request for a specific feature or a bug fix, associated documentation should also be changed with it.

The manual was written in RMarkdown using the bookdown package. All the code is stored in the `docs/src` directory as well as the script to build all the documentation. **Do not change** the files in the root of the `docs` directory, because those files will be removed during the build processed and replaced by the new ones. Therefore, to update this documentation, apply the changes to the files in the directory `docs/src`. To build the documentation, you need to have R installed, and if you are using UNIX-based systems, you only need to run `sh _build.sh` in the `docs/src` directory. The `_build.sh` script executes three commands at the same time to generate different output formats which might conflict with eachother. If some error happends telling that a file was not found and you didn't change nothing related to the specific file, rerun the `_build.sh` script.

In this documentation, we also describe all the settings around the dashboards that are used on the EHDEN project. To avoid an extensive table of contents

and also to avoid having a big chapter page for dashboards, we configured this GitBook to split different sections into different pages. A section on the GitBook is mapped to markdown headings elements of level 2 (H2 or ##). This is, however, inconvenient for small chapters like the preface (index.Rmd). To make it render all the sections on the same page, instead of using headings of level 2 (##) you should use level 3 (###). Although this will make the section numeration start at 0, e.g 1.0.1, 1.0.2, … To avoid this we appended `{-}` to the sections titles so that the numeration does not show.

If a new file is created with more documentation, its name should be placed, including extension, in the desired location in this list of the `docks/src/_bookdown.yml` file.

Chapter 8

Code Documentation

Apps

Materialized Queries Manager

Models

This app has only one model, MaterializedQuery, which maps to a Postgres materialized view. To avoid having to maintain the consistency between both the records of this Django app and the Postgres materialized views:

- the managed Meta flag was set to `False` to avoid Django creating migrations to the model
- the db_table Meta flag was set to the name of the table where Postgres stores the information about the existing materialized views (`pg_matviews`).
- the fields of the model, matviewname and definition, use the same name and type as the ones of the `pg_matviews` Postgres table.

Views

This app has no view exposed since all operations to the MaterializedQuery models are expected to be performed in the Django admin app.

However, we had to change Django's default behaviors for the create, update and delete operations of the model. For the delete operation, we overrode the delete method of the MaterializedQuery Django model to just execute a `DROP MATERIALIZED VIEW` SQL statement. Related to creation and update we had to change some internal methods of Django's admin app ModelAdmin base class.

1. _changeform_view: where Model records were being created. Instead, `CREATE MATERIALIZED VIEW` and `ALTER MATERIALIZED VIEW` SQL statements are executed. However, since some materialized views might take

some time to build, create a record like this could lead to a browser timeout. We then decided to execute these statements in a celery background task. The main changes were made here where we launch the background task.
2. response_add: Since the materialized view might not be created the right way, saying "A record was created successfully" is not adequate. We then changed the message that is presented after the creation to tell in the id of the background task that is creating the materialized query. The result of the query can then be consulted on the associated Task Results record on the Celery Results section app of the Django admin console.
3. response_change: changes here with the same ideas behind as response_add.

If any catalogue results files are being uploaded to the platform, any worker attempting to create or change a materialized view will block until such data is uploaded. Also if any worker is creating materialized views, no other worker can upload catalogue results data.

Through the admin console, there is also the possibility to refresh a specific MaterializedQuery. To do so, on the list view, select the MaterializedQueries to refresh, then on the actions dropdown select "Refresh selected materialized queries". Once again to avoid timeouts, such operations are executed on a background task.

Tabs Manager

Currently, this app is not being used and the URL mapping was delete. To use it again uncomment the tabsManager line on the dashboard_viewer/dashboard_viewer/urls.py file. Then you can access the tabs page through the `[BASE_RUL]/tabs/` URL.

Views

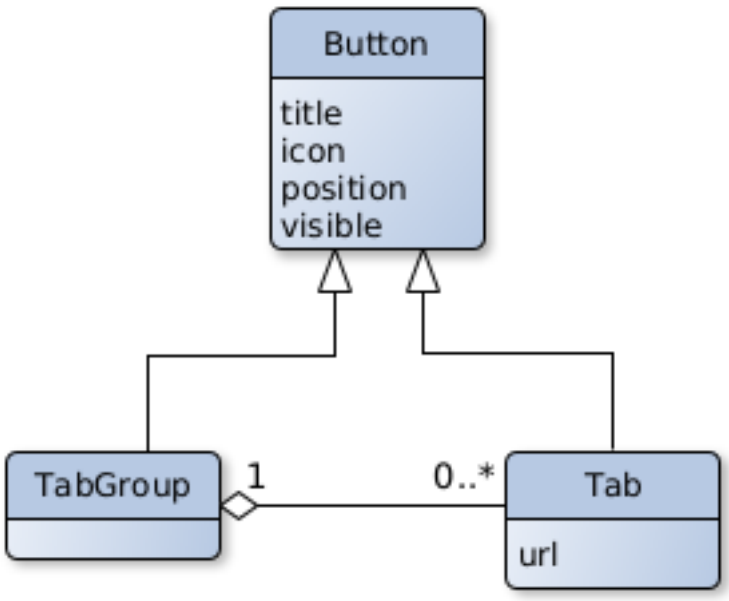
On this app, there is only one view. It is a simple page with just a sidebar to choose which dashboard to displays on an iframe. Besides the simplicity, all the animations around the sidebar buttons are handled by CSS with some JS that adds and removes classes to HTML elements, as events (hover and click) happen. To facilitate the development process of CSS, SCSS was used to build styling of the view. It prevents duplication with the addition of variables and adds the possibility to express parent classes by nesting their declaration.

In cases where there are a lot of buttons on the sidebar, some buttons might get impossible to reach since they are out of the field of view. To avoid this we make use of SimpleBar, which makes the sidebar scrollable, displaying a scroll bar on the right side of the sidebar whenever there are elements outside of the field of view.

API

There is one endpoint, `[BASE_URL]/api/`, where a JSON object of tabs and groups of tabs and their sub-tabs are returned.

Models



Uploader

Views

This app exposes three views: 1. Creation of a data source 2. Edition of a data source 3. To Upload or consult the history of uploads of catalogue results files.

The first one can be accessed through the `[BASE_URL]/uploader/[DATA_SOURCE_HASH]/` URL.

- If no hash is provided on the URL then on the creation of the data source a random one will be assigned.
- If there is already a data source with the provided hash then the user is redirected to the upload page of that data source.

This view also allows creating data sources without displaying the web-form, redirecting directly to the uploader page. This can be achieved by providing the data of several fields of the form as URL arguments. E.g. `[BASE_URL]/uploader/[DATA_SOURCE_HASH]/?acronym=test...`. This is implemented in a way so that whenever a GET is performed, it checks the URL arguments and tries to submit the data source form. If it is valid, all the required fields were provided and are valid, then the user is redirected to the upload page. Else all the valid values are set in the form, the invalid ones are being discarded, and the data source creation page is presented with no error messages. The country field should contain a value

from the ones available on the dropdown presented in the webform and since the coordinates is a two-component value it should be provided as `coordinates_0=[LATITUDE]&coordinates_1=[LONGITUDE]`. It is important to note that this view does not require a CSRF token, so a normal POST form submission can be performed to create a data source.

The second one can be accessed through the `[BASE_URL]/uploader/[DATA_SOURCE_HASH]/edit/` URL or by clicking on the Edit button on the data source upload page.

Finally, on the upload page, a data owner can consult the history of uploads, their state, and eventually error messages if some went wrong. Whenever an upload is made its state will be pending. After the upload, with a 5-second interval, a request is made to the backend to check if the status of the upload changed. If it fails, an error message will be provided in a tooltip above a button with a message icon. Else the state will change to Done and the information about the upload retrieved from the uploaded file, present on the check status request, is filled.

Related to file uploading, after the file form is submitted no validation is made and a message is presented to the user telling that the file is being processed in the background, then the fetch status process mentioned before starts. If validations were performed before returning a message to the user, if the validation took too much time, the browser could timeout. Also if some unexpected error happened on the insertion process performed in the background, the user would get any feedback.

Related to the background task that validates and uploads the data, the validation can fail if:

Error	Message
Invalid Number of Columns	The provided file has an invalid number of columns
Invalid CSV File Format	The provided file has an invalid CSV format. Make sure is a text file separated by commas and you either have 7 (regular results file) or 13 (results file with dist columns) columns.
Missing Required Fields	Some rows have null values either on the column “analysis_id” or “count_value”

Error	Message
Invalid Field Types	The provided file has invalid values on some columns. Remember that only the "stratum_*" columns accept strings, all the other fields expect numeric types.
Duplicated Metadata Rows	Analysis id[output] duplicated on multiple rows. Try (re)running the plugin CatalogueExport on your database.
Missing Required Metadata Row	Analysis id 0 is missing. Try (re)running the plugin CatalogueExport on your database.

Any other error is considered an unexpected error and the following message will be presented "An unexpected error occurred while processing your file. Please contact the system administrator for more details.".

If the file passes all validations, it goes to the upload data phase. Here, if workers are creating or refreshing materialized queries then the worker blocks. If there are other workers inserting data for the same data source it will also block. However, several workers of different data sources can insert data at the same time. All the workers, after inserting the data, check if they are the only worker inserting data. If so they refresh the existing materialized queries. Else the next worker to finish inserting data will do the same check.

Widgets

For the data source form two custom widgets were created for the following fields:

- Database Type: To avoid having duplicate entries with the same meaning (e.g. Hospital, hospital), the input of this field has a autocomplete list where existing values are suggested to the user. Also before saving the field to the database spaces are trimmed and the values are transformed into title case here.
- Coordinates: 1. This is a two-component field; 2. Inserting coordinates by hand is tedious. Considering the previous points, we created a widget with a map built with leaflet where the user just needs to click on the map.

API

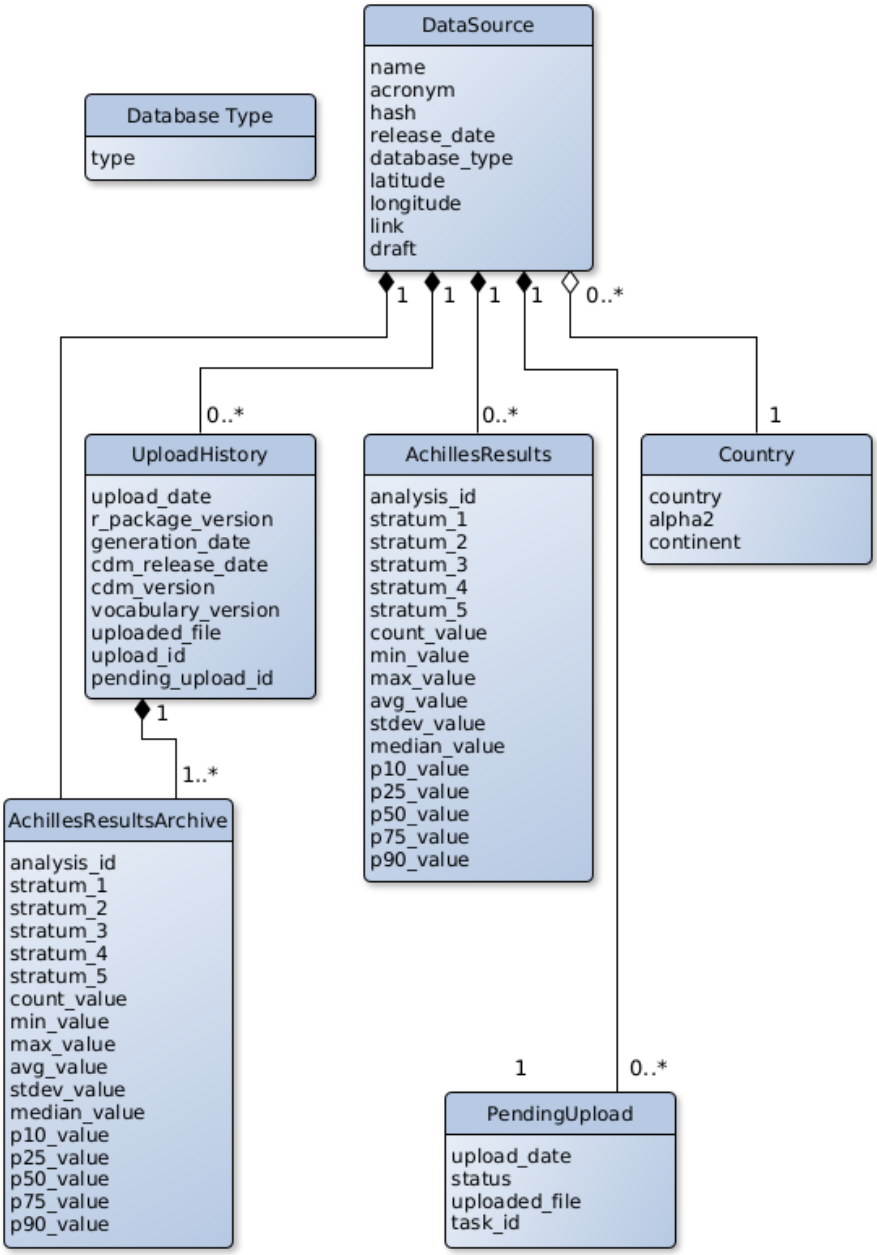
This app provided two API endpoints

1. Update data source information: a PATCH request with a JSON object on the body of the request with the fields and their new values.

2. Pending upload status: a GET request that returns JSON data where there is always a `status` field that can have three statuses which then can lead to additional data be also present:

   - Pending: the upload in question hasn't finished
   - Done: the upload finished and there was nothing wrong with the uploaded file. Along with the status, there will be a `data` field with a JSON object with the fields `r_package_version`, `generation_date`, `cdm_version`, and `vocabulary_version` which are data source information that was extracted from the uploaded file.
   - Failed: the upload finished but there was something wrong with the uploaded file. Along with the status, there will be a `failure_msg` field telling the reason for the failure.

Models



- Country data is loaded in a fresh installation through the docker-init.sh script if no records are present on the Country table.
- The DataSource model doesn't have a foreign key to the DatabaseType model to then facilitate the creation of SQL queries to feet Superset's

dashboards. The DatabaseType is used anyway to have a faster way to check if a certain database type already exists on the database, avoiding going through every DataSource record.

- The same situation of the DatabaseType model also happens between the UploadHistory and PendingUpload models. There is no foreign key between a UploadHistory and a PendingUpload. This is because PendingUpload records are deleted once an upload is successful. When the upload view requests the status of a certain upload, it uses the id of the pending upload. If no pending upload is found, it is assumed that the upload was successful and searches for uploads on the UploadHistory model with the pending_upload_id field equal to the certain upload id. Related to where the uploaded files are stored, within the media directory there will be a ACHILLES_RESULTS_STORAGE_PATH directory which will have a directory for each data source. Within this last directory, first, files are uploaded to a `failure` directory. If the upload is successful the file is moved to a success directory. In both cases, the file name will be the date of when the file is being saved into disk plus its original extensions.

JavaScript Packages

While developing the templates for Django views, if a certain javascript library is required, like jquery, one option is to insert `script` tags on the templates and point the src to a CDN. However, this makes the process of maintaining the libraries tedious since a developer has to search and change all the `script` tags if for example wants to update the library's version. To avoid this problem we have a `package.json` file where we define all the libraries that we use and their version. Then we add the `node_modules` directory as a static file directory. With this alternative, updating a library is as simple as changing a number of the `package.json` file, run `npm install` and collect the static files again.

Chapter 9

Dashboards

9.1 Per Database

Label Colors

In order to obtain the colors blue and rose in the chart representing the gender distribution, add the following JSON entry to the JSON object of the `JSON Metadata` field on the edit dashboard page:

```
"label_colors": {
    "Male": "#3366FF",
    "Female": "#FF3399"
}
```

CSS

To hide the dashboard header insert the following css code to the `CSS` field on the edit page:

```
.dashboard > div:not(.dashboard-content) {  /* dashboard header */
  display: none;
}
```

With this every time you want to edit the dashboard layout you have to either comment the CSS inserted or remove it so the "Edit Dashboard" button can show again.

Data Source Filter

For the filter to work the name of the fields to filter should match in all tables used on the charts of this dashboard.

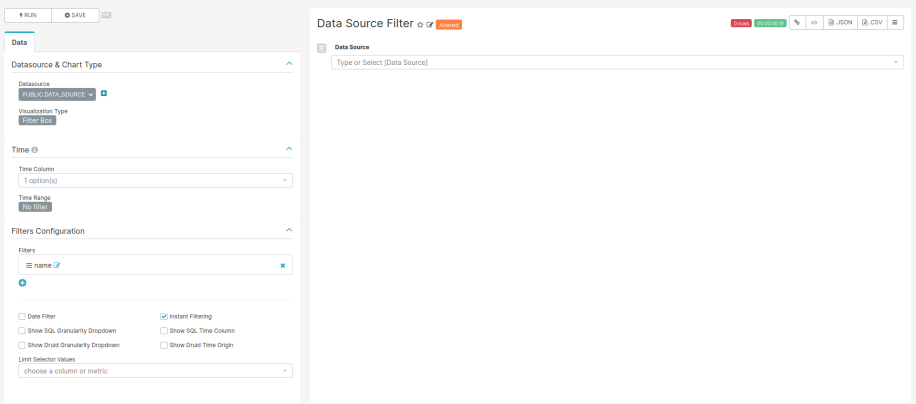


Figure 9.1: Settings for creating the Data Source filter chart

SQL query

No SQL query, use the sql table `data_source` of the `achilles` database.

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Filter Box
  - Time
    - Time range: No filter
  - Filters Configuration
    - Filters:
      - name
    - Date Filter: off
    - Instant Filtering: on

Demographics Tab

Number of Patients

```
SELECT
  achilles_results.count_value,
  data_source.name,
  data_source.acronym
FROM achilles_results
JOIN data_source ON achilles_results.data_source_id=data_source.id
WHERE analysis_id = 1
```

SQL query

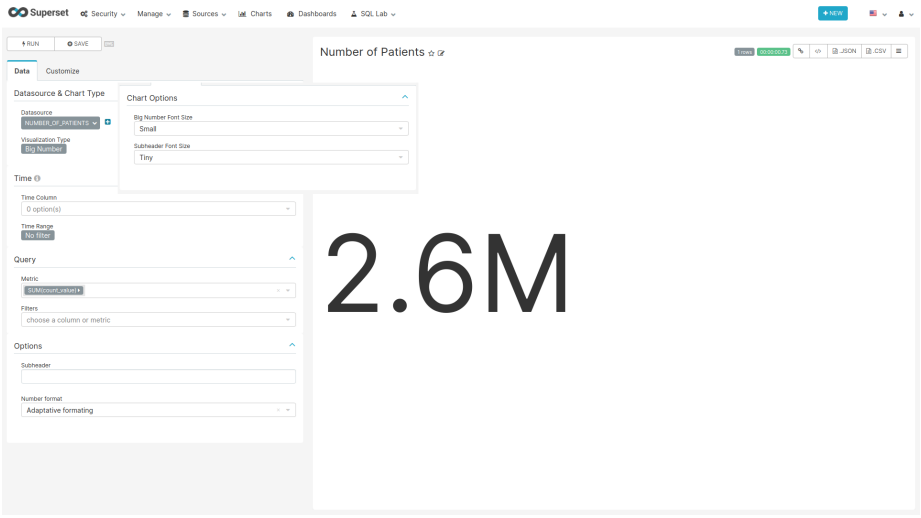


Figure 9.2: Settings for creating the Number of Patients chart

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Big Number
  - Time Time range: No filter
  - Query
    - Metric: sum(count_value)
- Customize Tab
  - Big Number Font Size: Small
  - Subheader Font Size: Tiny

Gender Table

```
SELECT source.name as name,
       source.acronym,
       concept_name as gender,
       count_value
FROM public.achilles_results AS achilles
INNER JOIN public.data_source AS source ON achilles.data_source_id=source.id
INNER JOIN public.concept ON CAST(stratum_1 AS BIGINT) = concept_id
WHERE analysis_id = 2
```

SQL Query {#genderTableQuery}

Chart settings

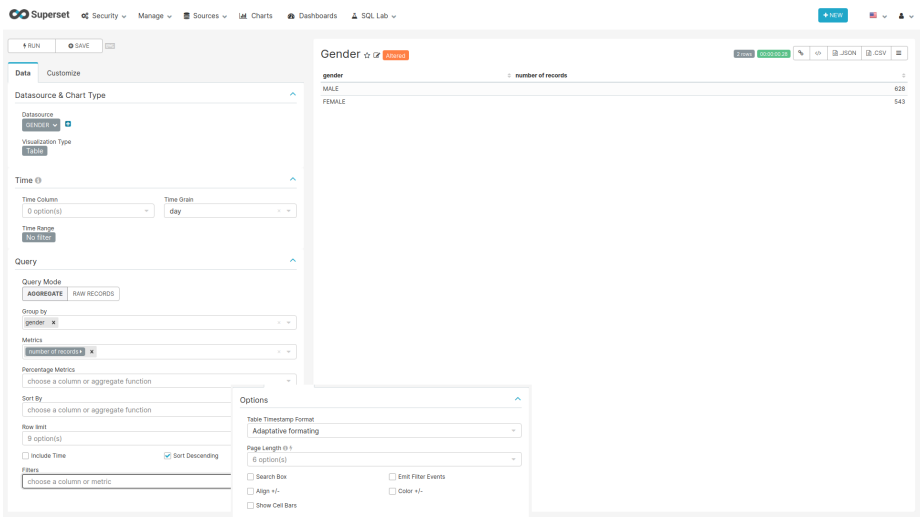


Figure 9.3: Settings for creating the Gender Table chart

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Table
  - Time
    - Time range: No filter
  - Query
    - Query Mode: Aggregate
    - Group by: gender
    - Metrics: SUM(count_value) with label number of records
    - Row lmit: None
- Customize Tab
  - Options
    - Show Cells Bars: off

Gender Pie

SQL query Same as Gender Table query

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Pie Chart
  - Time
    - Time range: No filter
  - Query

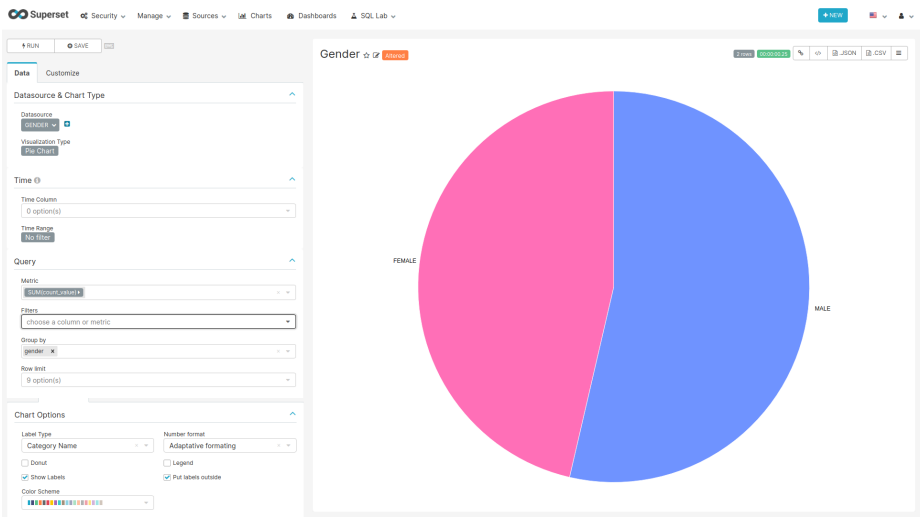


Figure 9.4: Settings for creating the Gender Pie chart

        - Metric: SUM(count_value)
        - Group by: gender
        - Row limit: None
- Customize Tab
    - Chart Options
        - Legend: off

Age at first observation - Table

Same chart as the one used on the Person dashboard.

Age at first observation - Bars

Same chart as the one used on the Person dashboard.

Year of Birth

Same chart as the one used on the Person dashboard.

Data Domains Tab

Average Number of Records per Person

Same chart as the one used on the Data Domains dashboard.

Total Number of Records

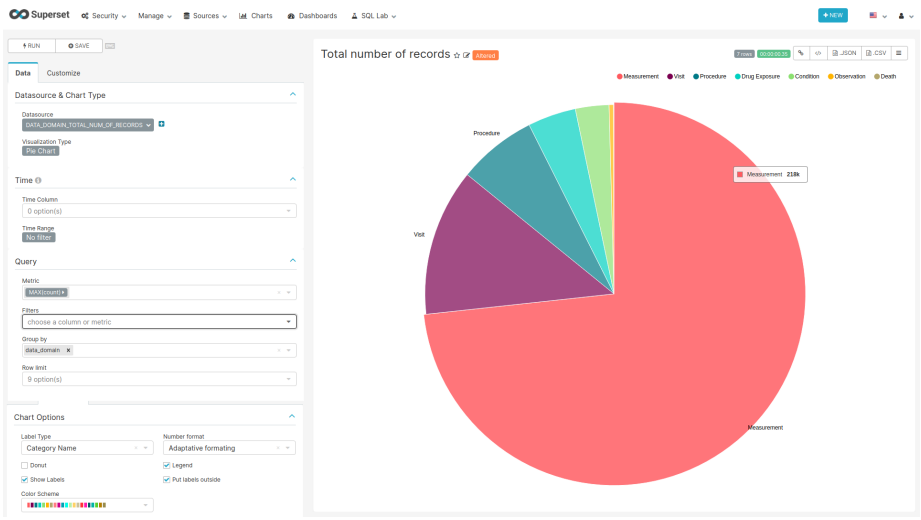


Figure 9.5: Settings for creating the Total Number of Records chart

```
SELECT
data_source.name,
data_source.acronym,
    CASE
    WHEN analysis_id = 201 THEN 'Visit'
    WHEN analysis_id = 401 THEN 'Condition'
    WHEN analysis_id = 501 THEN 'Death'
    WHEN analysis_id = 601 THEN 'Procedure'
    WHEN analysis_id = 701 THEN 'Drug Exposure'
    WHEN analysis_id = 801 THEN 'Observation'
    WHEN analysis_id = 1801 THEN 'Measurement'
    WHEN analysis_id = 2101 THEN 'Device'
    WHEN analysis_id = 2201 THEN 'Note'
    END AS Data_Domain,
    SUM(count_value) AS "count"
FROM achilles_results
JOIN data_source ON achilles_results.data_source_id=data_source.id
GROUP BY name, acronym, analysis_id
HAVING analysis_id IN (201, 401, 501, 601, 701, 801, 1801, 2101, 2201)
```

SQL query

Chart settings

- Data Tab

  - Datasource & Chart Type
    - Visualization Type: Pie Chart
  - Time
    - Time range: No filter
  - Query
    - Metric: MAX(count)
    - Group by: data_domain
    - Row limit: None

Data Provenance Tab

Same six charts used on the Provenance dashboard.

Observation Period Tab

Number of Patitents in Observation Period

Same chart used on the Observation Period dashboard.

Cumulative Observation Period

The cumulative observation time plot shows the percentage of patients that have more that X days of observation time.

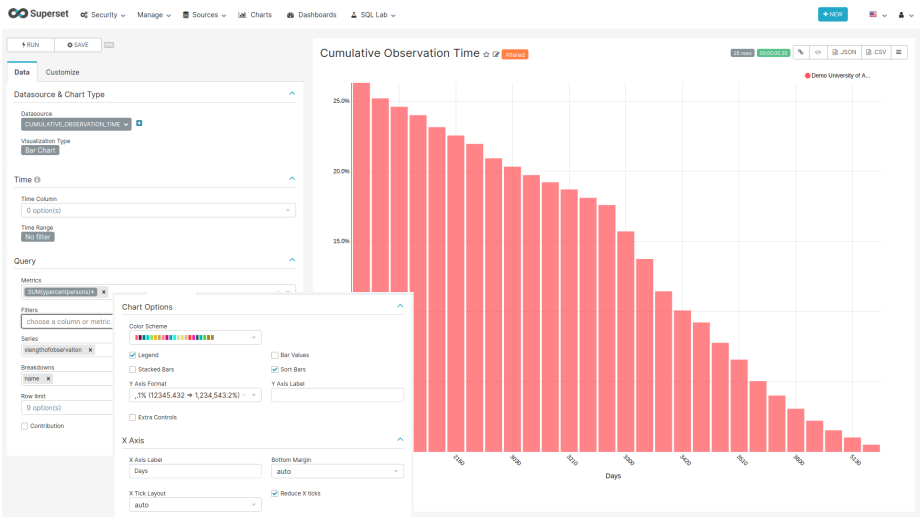


Figure 9.6: Settings for creating the Total Number of Records chart

```
SELECT
  name,
```

```
  acronym,
  xLengthOfObservation,
  round(cumulative_sum / total, 5) as yPercentPersons
FROM (
  SELECT data_source_id, CAST(stratum_1 AS INTEGER) * 30 AS xLengthOfObservation, SUM(c
  FROM achilles_results
  WHERE analysis_id = 108
) AS cumulative_sums
JOIN (
  SELECT data_source_id, count_value as total
  FROM achilles_results
  WHERE analysis_id = 1
) AS totals
ON cumulative_sums.data_source_id = totals.data_source_id
JOIN data_source ON cumulative_sums.data_source_id = data_source.id
ORDER BY name, xLengthOfObservation
```

SQL Query

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Bar Chart
  - Time
    - Time range: No filter
  - Query
    - Metrics: SUM(ypercentpersons)
    - Series: xlengthofobservation
    - Breakdowns: name
    - Row limit: None
- Customize Tab
  - Chart Options
    - Sort Bars: on
    - Y Axis Fomat: ,.1% (12345.432 => 1,234,543.2%)
    - Y Axis Label: Number of Patients
  - X Axis
    - X Axis Label: Days
    - Reduce X ticks: on

Visit Tab

Visit Type Graph

```
SELECT
```

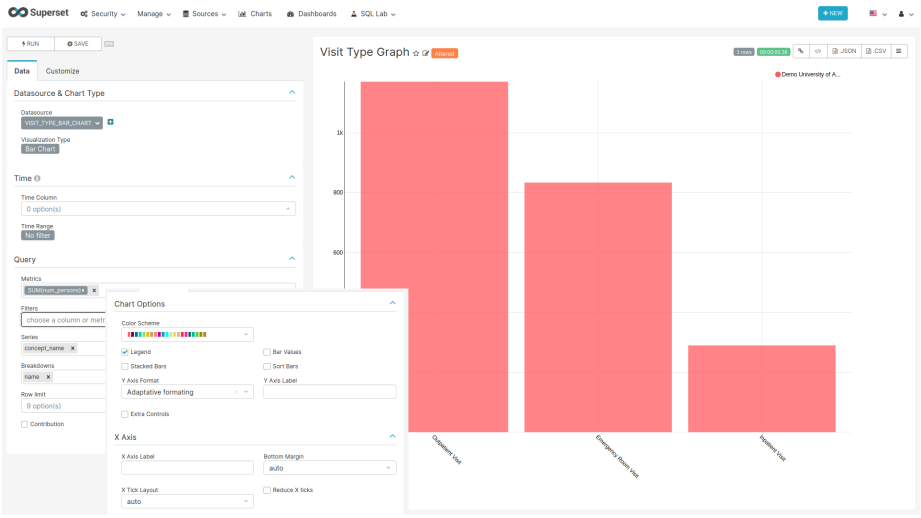


Figure 9.7: Settings for creating the Visit Type Graph chart

```
  data_source.name,
  data_source.acronym,
  concept.concept_name,
  achilles_results.count_value AS num_persons
FROM (SELECT * FROM achilles_results WHERE analysis_id = 200) AS achilles_results
JOIN data_source ON achilles_results.data_source_id = data_source.id
JOIN concept ON CAST(achilles_results.stratum_1 AS BIGINT) = concept.concept_id
```

SQL Query

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Bar Chart
  - Time
    - Time range: No filter
  - Query
    - Metrics: SUM(num_persons)
    - Series: concept_name
    - Breakdowns: name
    - Row limit: None

Visit Type Table

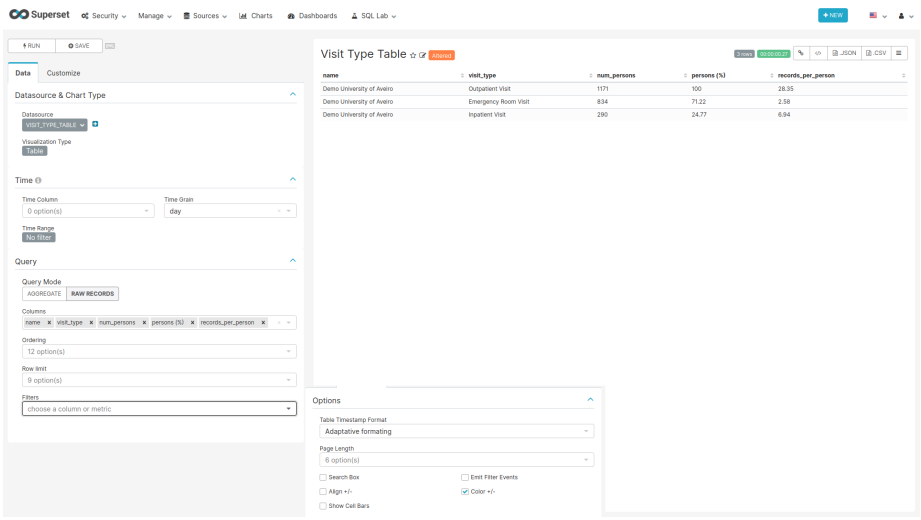


Figure 9.8: Settings for creating the Visit Type Table chart

```
SELECT
  name,
  acronym,
  concept.concept_name,
  ar1.count_value AS num_persons,
  round(100.0 * ar1.count_value / denom.count_value, 2) AS percent_persons,
  round(1.0 * ar2.count_value / ar1.count_value, 2) AS records_per_person
FROM (
  SELECT *
  FROM achilles_results WHERE analysis_id = 200) AS ar1
  JOIN (
    SELECT *
    FROM achilles_results WHERE analysis_id = 201) AS ar2
    ON ar1.stratum_1 = ar2.stratum_1 AND ar1.data_source_id = ar2.data_source_id
  JOIN (
    SELECT *
    FROM achilles_results WHERE analysis_id = 1) AS denom
    ON ar1.data_source_id = denom.data_source_id
  JOIN data_source ON data_source.id = ar1.data_source_id
  JOIN concept ON CAST(ar1.stratum_1 AS INTEGER) = concept_id
ORDER BY ar1.data_source_id, ar1.count_value DESC
```

SQL Query

Chart Settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Table
  - Time
    - Time range: No filter
  - Query
    - Query Mode: Raw Records
    - Columns: name, visit_type, num_persons, percent_persons with label persons (%), records_per_person
    - Row limit: None
- Customize Tab
  - Options
    - Show Cell Bars: off

Concept Browser Tab

Concept Browser Table

Same chart used on the Concept Browser dashboard.

Meta Data Tab

Meta Data Table

Same chart used on the General dashboard.

9.2 Database Level Dashboard

This dashboard is an exact copy of the Per Database dashboard but several legends and fields displayed on the original are hidden either through CSS or by changing some chart settings. On the following sections we will only present the things to change on the original charts.

Label Colors

In order to obtain the colors blue and rose in the chart representing the gender distribution, add the following JSON entry to the JSON object of the `JSON Metadata` field on the edit dashboard page:

```
"label_colors": {
    "Male": "#3366FF",
    "Female": "#FF3399"
}
```

CSS

To hide the dashboard header insert the following css code to the `CSS` field on the edit page:

```
/* hides the filter badges on right side of charts */
.dashboard-filter-indicators-container {
    display: none;
}
/* hides the acronym filter */
.grid-content > .dragdroppable.dragdroppable-row > .with-popover-menu {
    display: none;
}
/*
WARNING panel 1 id hardcoded
Hides the X Axis Label of the heatmap on the Data Domains tab
*/
#TABS-nlIU6H5mcT-pane-1 g.x.axis > g.tick text {
    display: none;
}
/*
WARNING panel 2 id hardcoded
Hides the X Axis Labels of the bar charts on the Data Provenance tab
*/
#TABS-nlIU6H5mcT-pane-2 g.nv-x.nv-axis.nvd3-svg > g.nvd3.nv-wrap.nv-axis > g > g.tick.
    display: none;
}
```

With this every time you want to edit the dashboard layout you have to either comment the CSS inserted or remove it so the "Edit Dashboard" button can show again.

Data Source Filter - hidden

For the filter to work the name of the fields to filter should match in all tables used on the charts of this dashboard.

SQL query

No SQL query, use the sql table `data_source` of the `achilles` database.

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Filter Box
  - Time
    - Time range: No filter

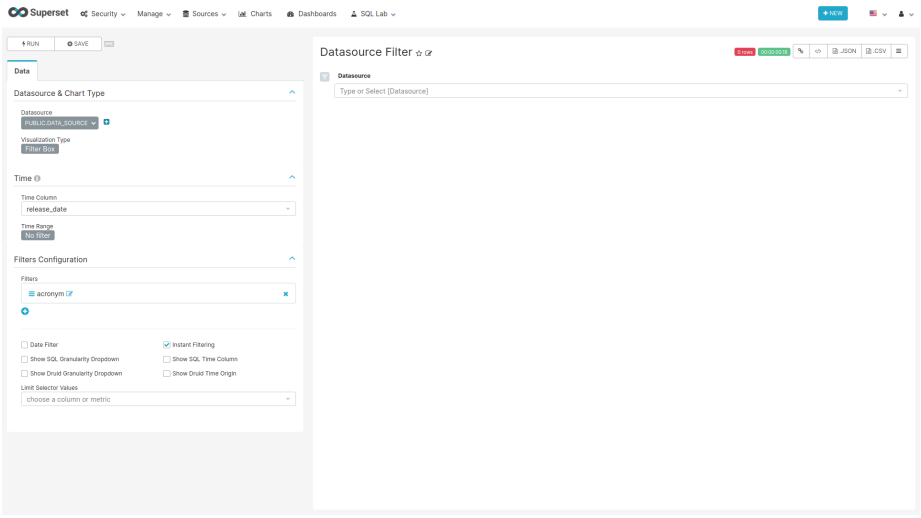


Figure 9.9: Settings for creating the Data Source filter chart

  - Filters Configuration
    - Filters:
      - acronym
    - Date Filter: off
    - Instant Filtering: on

Demographics Tab

Number of Patients

No changes

Gender Table

No changes

Gender Pie

No changes

Age at first observation - Table

Remove the `name` field from the columns to display.

- Data Tab
  - Query
    - Columns: 0-10, 10-20, 20-30, 30-40, 40-50, 50-60, 60-70, 70-80, 80-90, 90+

Age at first observation - Bars

Remove legend.

- Customize Tab
  - Chart Options
    - Legend: off

Year of Birth

Remove legend.

- Customize Tab
  - Chart Options
    - Legend: off

Data Domains Tab

No changes

Data Provenance Tab

No changes

Observation Period Tab

Number of Patitents in Observation Period

Remove legend.

- Customize Tab
  - Chart Options
    - Legend: off

Cumulative Observation Period

Remove legend.

- Customize Tab
  - Chart Options
    - Legend: off

Visit Tab

Visit Type Graph

Remove legend.

- Customize Tab
  - Chart Options
    - Legend: off

Visit Type Table

Remove the `name` field from the columns to display.

- Data Tab
  - Query
    - Columns: visit_type, num_persons, percent_persons with label persons (%), records_per_person

Concept Browser Tab

Concept Browser Table

Remove the `source_name` field from the columns to display.

- Data Tab
  - Query
    - Columns: concept_id, concept_name, domain_id, magnitude_persons, magnitude_occurrences

Meta Data Tab

Meta Data Table

Remove the `name` field from the columns to display.

- Data Tab
  - Query
    - Columns: source_release_date, cdm_release_date, cdm_version, vocabulary_version

9.3 General [Deprecated]

CSS

To hide the dashboard header insert the following css code to the `CSS` field on the edit page:

```
.dashboard > div:not(.dashboard-content) {  /* dashboard header */
  display: none;
}
```

With this every time you want to edit the dashboard layout you have to either comment the CSS inserted or remove it so the “Edit Dashboard” button can show again.

Database Type and Country Filter

Theses filter were designed to be used in the dashboard aiming the filtering of the data based on the field ‘’database_type” and “country” from the table

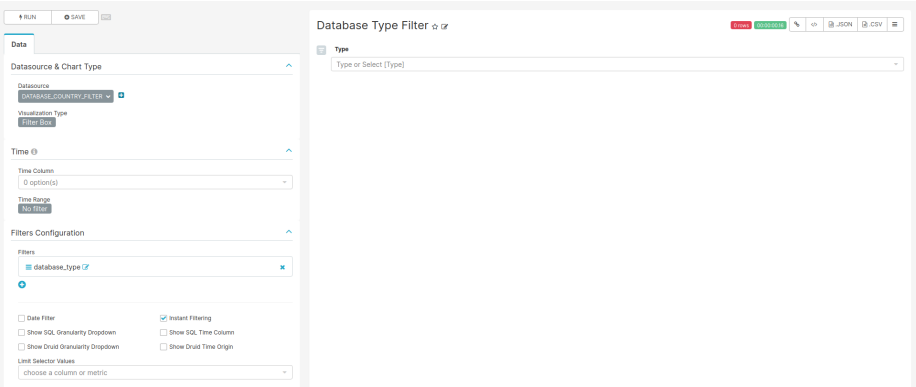


Figure 9.10: Settings for creating filters charts

‘’data_source”.

For the filters to work the name of the fields to filter should match in all tables used on the charts of this dashboard.

SQL query

```
SELECT source.name,
       country.country,
       source.database_type,
       source.acronym
FROM public.data_source AS source
INNER JOIN public.country AS country ON source.country_id=country.id
```

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Filter Box
  - Time
    - Time range: No filter
  - Filters Configuration
    - Filters:
      - database_type or country
    - Date Filter: off
    - Instant Filtering: on

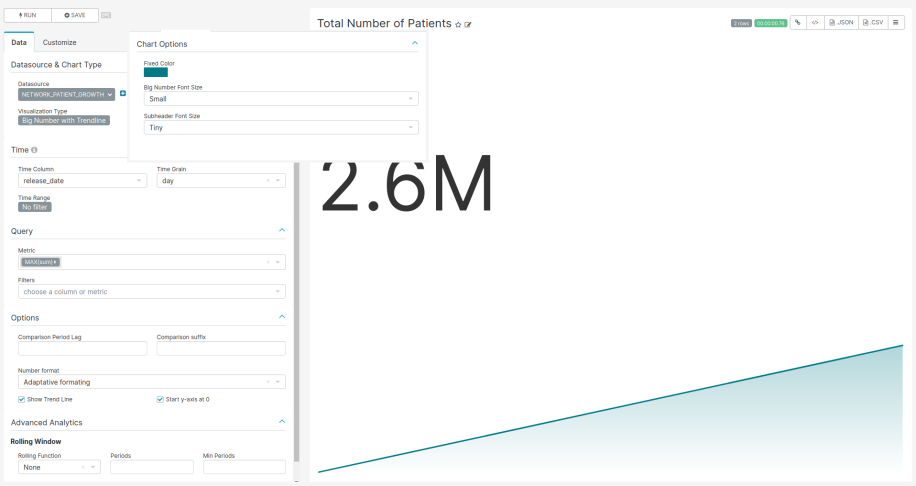


Figure 9.11: Settings for creating the Total Number of Patients chart

Total Number of Patients

SQL query

```
SELECT
 country,
 database_type,
 release_date,
 SUM(count_value) OVER (ORDER BY release_date ASC)
FROM achilles_results
JOIN data_source ON data_source_id = data_source.id
JOIN country ON data_source.country_id = country.id
WHERE analysis_id = 1
```

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Big Number with Trendline
  - Time
    - Time range: No filter
  - Query
    - Metrics: MAX(sum)
    - Series: release_date
    - Breakdowns: source
- Customize Tab
  - Chart Options

* Big Number Font Size: Small
* Subheader Font Size: Tiny

Network Growth by Date

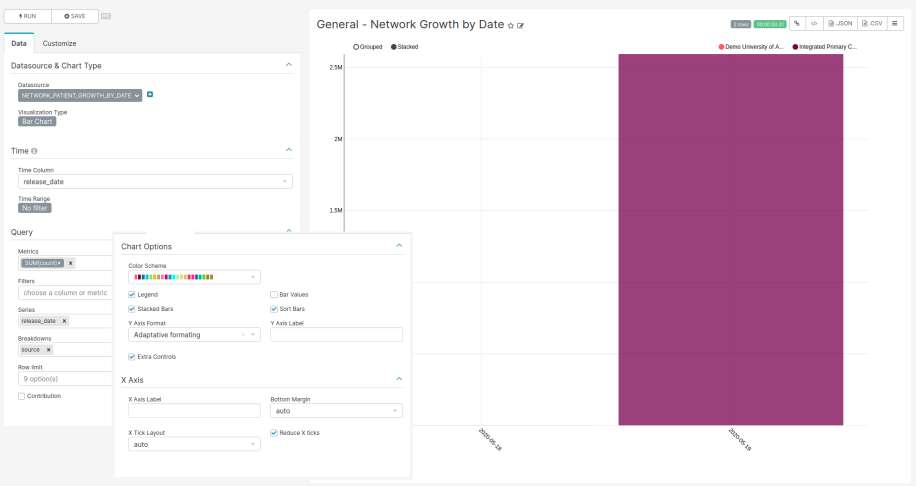


Figure 9.12: Settings for creating the Network Growth by Date chart

SQL query

```
SELECT  source.name AS source,
        country.country,
        source.database_type,
        source.release_date,
        concepts.concept_name AS gender,
        achilles.count_value as count
FROM public.achilles_results AS achilles
INNER JOIN public.data_source AS source ON achilles.data_source_id=source.id
INNER JOIN public.country AS country ON source.country_id=country.id
JOIN (
  SELECT '8507' AS concept_id, 'Male' AS concept_name
  UNION
  SELECT '8532', 'Female'
) AS concepts ON achilles.stratum_1 = concept_id
WHERE analysis_id = 2
```

Chart settings

- Data Tab
  - Datasource & Chart Type

        * Visualization Type: Bar Chart
    - Time
        * Time range: No filter
    - Query
        * Metrics: SUM(count_value)
        * Series: release_date
        * Breakdowns: source
- Customize Tab
    - Chart Options
        * Stacked Bars: on
        * Sort Bars: on
        * Extra Controls: on
    - X Axis
        * Reduce X ticks: on

Patients per Country

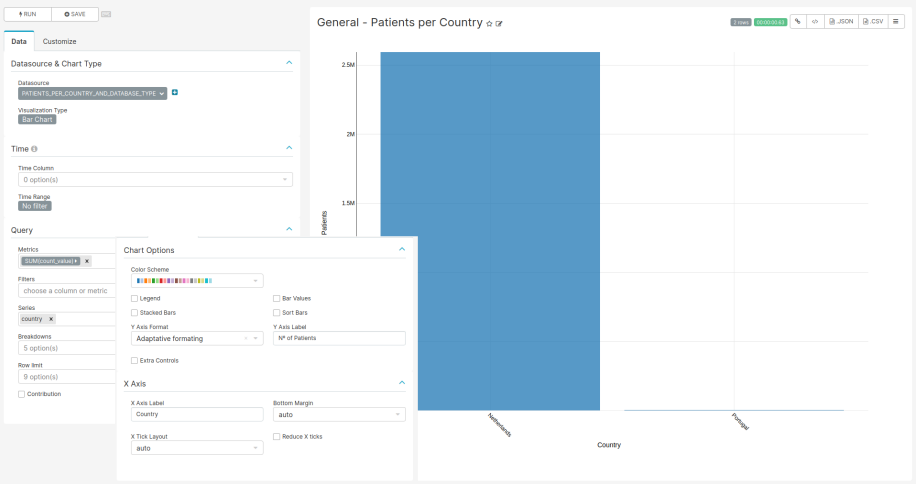


Figure 9.13: Settings for creating the Patients per Country chart

SQL query {#patientsPerCountryQuery}

```
SELECT country.country,
       source.database_type,
       count_value
FROM public.achilles_results AS achilles
INNER JOIN public.data_source AS source ON achilles.data_source_id=source.id
INNER JOIN public.country AS country ON source.country_id=country.id
WHERE analysis_id = 1
```

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Bar Chart
  - Time
    - Time range: No filter
  - Query
    - Metrics: SUM(count_value)
    - Series: country
- Customize Tab
  - Chart Options
    - Legend: off
    - Y Axis Label: Nº of Patients
  - X Axis
    - X Axis Label: Country

Database Types per Country

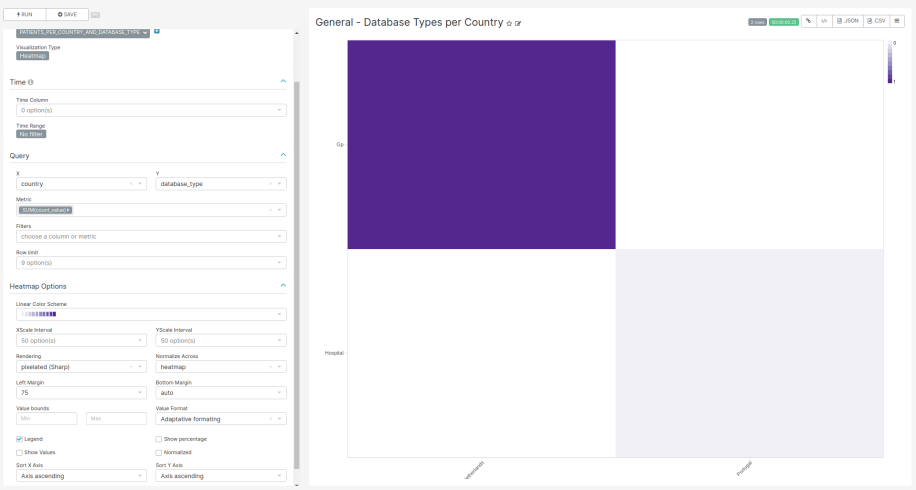


Figure 9.14: Settings for creating the Database Type per Country chart

SQL query

Same as Patients per Country query

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Heatmap

- Time
  - Time range: No filter
- Query
  - X: country
  - Y: database_type
  - Metric: SUM(countr_value)
- Heatmap Options
  - Left Margin: 75
  - Show Percentage: off

World Map

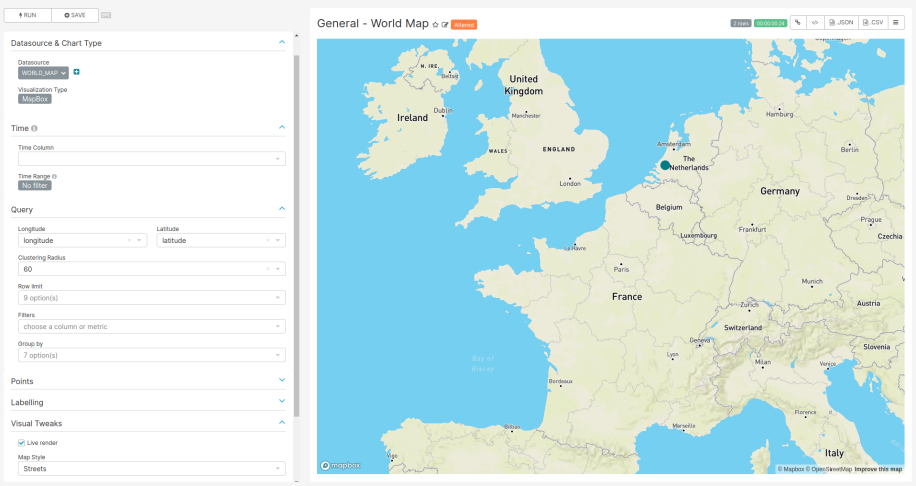


Figure 9.15: Settings for creating the World Map chart

SQL query

```
SELECT  name,
        acronym,
        database_type,
        latitude,
        longitude,
        country
FROM public.data_source AS source
INNER JOIN public.country AS country ON source.country_id=country.id
```

Chart settings

- Data Tab
  - Datasource & Chart Type

    - Visualization Type: MapBox
  - Time
    - Time range: No filter
  - Query
    - Longitude: longitude
    - Latitude: latitude
  - Visual Tweaks
    - Map Style: Streets or Light or Outdoors

Meta Data

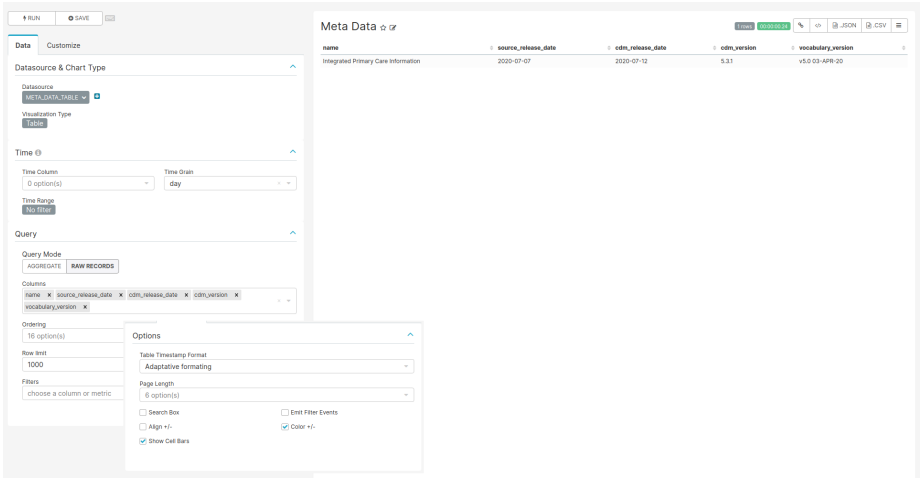


Figure 9.16: Settings for creating the Meta Data chart

SQL query

```
SELECT
  acronym,
  stratum_1 as "name",
  database_type,
  country,
  stratum_2 as "source_release_date",
  stratum_3 as "cdm_release_date",
  stratum_4 as "cdm_version",
  stratum_5 as "vocabulary_version"
FROM achilles_results
JOIN data_source ON achilles_results.data_source_id = data_source.id
JOIN country ON data_source.country_id = country.id
WHERE analysis_id=5000
```

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Table
  - Time
    - Time range: No filter
  - Query
    - Query Mode: Raw Records
    - Columns: name, source_release_date, cdm_release_date, cdm_version, vocabulary_version

9.4 Person [Deprecated]

Label Colors

In order to obtain the colors blue and rose in the chart representing the gender distribution, add the following JSON entry to the JSON object of the `JSON Metadata` field on the edit dashboard page:

```
"label_colors": {
    "Male": "#3366FF",
    "Female": "#FF3399"
}
```

CSS

To hide the dashboard header insert the following css code to the `CSS` field on the edit page:

```
.dashboard > div:not(.dashboard-content) {  /* dashboard header */
  display: none;
}
```

With this every time you want to edit the dashboard layout you have to either comment the CSS inserted or remove it so the "Edit Dashboard" button can show again.

Data Source Filter

For the filter to work the name of the fields to filter should match in all tables used on the charts of this dashboard.

SQL query

No SQL query, use the sql table `data_source` of the `achilles` database.

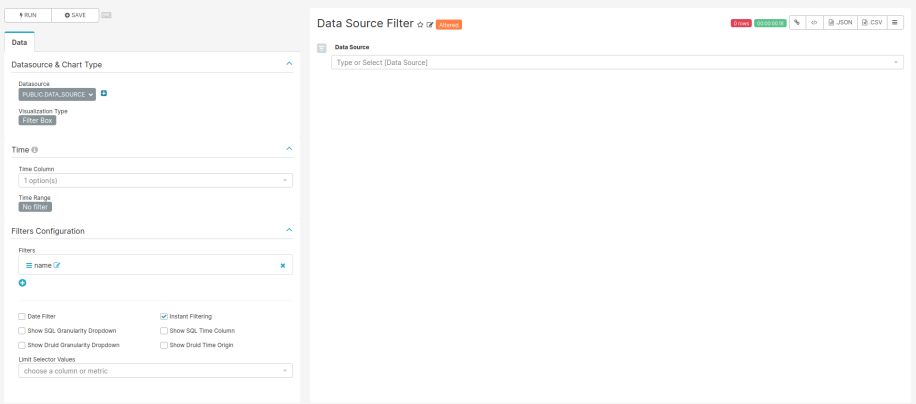


Figure 9.17: Settings for creating the Data Source filter chart

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Filter Box
  - Time
    - Time range: No filter
  - Filters Configuration
    - Filters:
      - name
    - Date Filter: off
    - Instant Filtering: on

Age at first observation - Table {#age1ObservationTable}

SQL query

```
SELECT source.name,
       source.acronym,
       SUM(CASE WHEN CAST(stratum_2 AS INTEGER) < 10 THEN count_value END) AS "0-10",
       SUM(CASE WHEN CAST(stratum_2 AS INTEGER) >= 10 AND CAST(stratum_2 AS INTEGER) <
       SUM(CASE WHEN CAST(stratum_2 AS INTEGER) >= 20 AND CAST(stratum_2 AS INTEGER) <
       SUM(CASE WHEN CAST(stratum_2 AS INTEGER) >= 30 AND CAST(stratum_2 AS INTEGER) <
       SUM(CASE WHEN CAST(stratum_2 AS INTEGER) >= 40 AND CAST(stratum_2 AS INTEGER) <
       SUM(CASE WHEN CAST(stratum_2 AS INTEGER) >= 50 AND CAST(stratum_2 AS INTEGER) <
       SUM(CASE WHEN CAST(stratum_2 AS INTEGER) >= 60 AND CAST(stratum_2 AS INTEGER) <
       SUM(CASE WHEN CAST(stratum_2 AS INTEGER) >= 70 AND CAST(stratum_2 AS INTEGER) <
       SUM(CASE WHEN CAST(stratum_2 AS INTEGER) >= 80 AND CAST(stratum_2 AS INTEGER) <
       SUM(CASE WHEN CAST(stratum_2 AS INTEGER) >= 90 THEN count_value END) AS "90+"
FROM public.achilles_results AS achilles
```

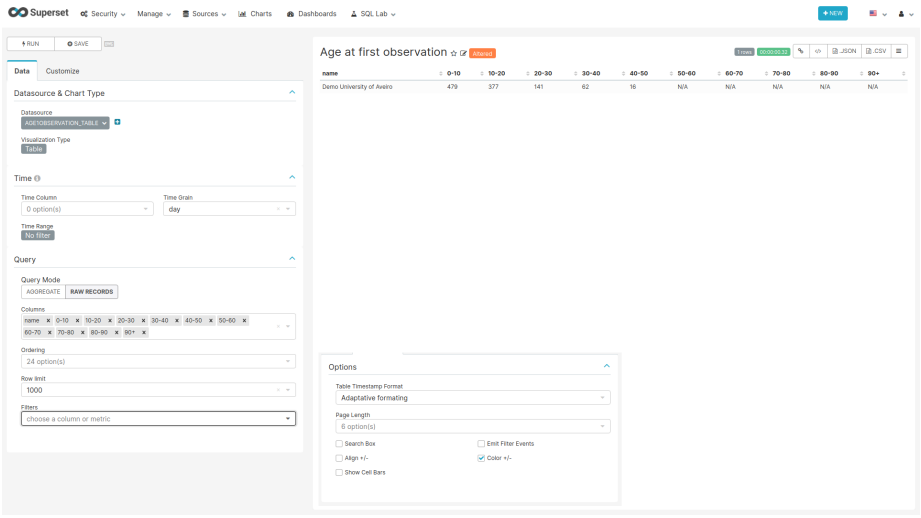


Figure 9.18: Settings for creating the Age at First Observation Table chart

```
INNER JOIN public.data_source AS source ON achilles.data_source_id=source.id
INNER JOIN public.concept ON CAST(stratum_1 AS BIGINT) = concept_id
WHERE analysis_id = 102
GROUP BY name, acronym
```

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Table
  - Time
    - Time range: No filter
  - Query
    - Query Mode: Raw Records
    - Columns: name, 0-10, 10-20, 20-30, 30-40, 40-50, 50-60, 60-70, 70-80, 80-90, 90+
- Customize Tab
  - Options
    - Show Cell Bars: off

Age at first observation - Bars {#age1ObservationBars}

SQL query

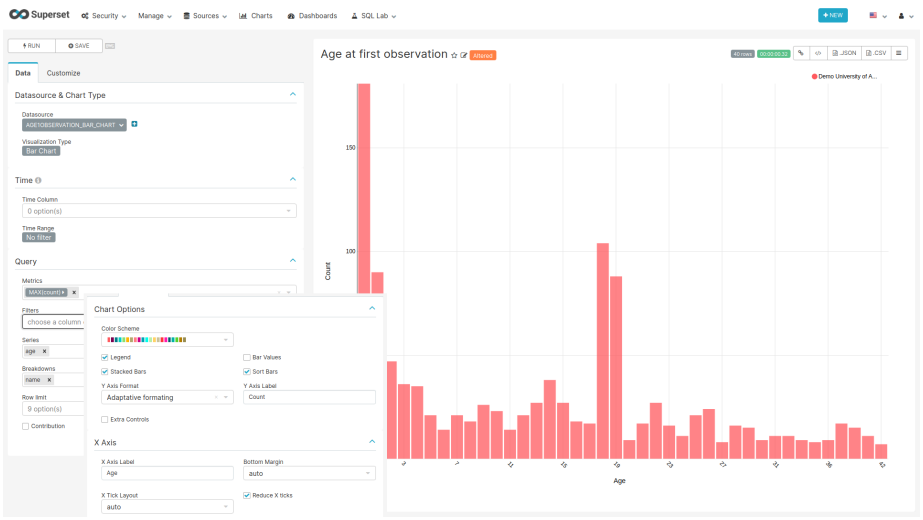


Figure 9.19: Settings for creating the Age at First Observation Bar chart

```
SELECT source.name,
    cast(stratum_1 AS int) AS Age,
    count_value AS count,
    source.acronym
FROM public.achilles_results AS achilles
INNER JOIN public.data_source AS source ON achilles.data_source_id=source.id
WHERE analysis_id = 101
```

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Bar Chart
  - Time
    - Time range: No filter
  - Query
    - Metrics: MAX(count)
    - Series: age
    - Breakdowns: name
- Customize Tab
  - Chart Options
    - Stacked Bars: on
    - Sort Bars: on
    - Y Axis Label: Count
  - X Axis

    * X Axis Label: Age
    * Reduce X ticks: on

Year of Birth {#yearOfBirth}

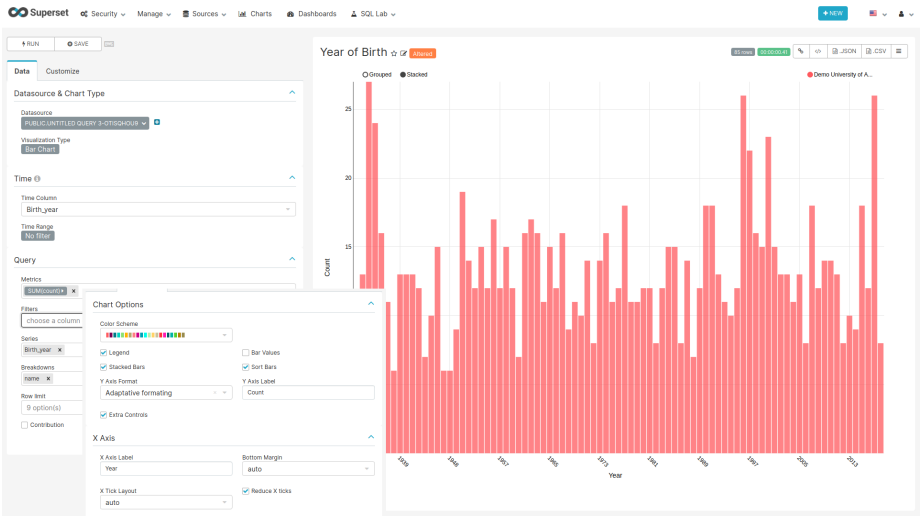


Figure 9.20: Settings for creating the Year of Birth chart

SQL query

```
SELECT source.name,
       source.acronym,
       stratum_1 AS "Birth_year",
       count_value AS count
FROM public.achilles_results AS achilles
INNER JOIN public.data_source AS source ON achilles.data_source_id=source.id
WHERE analysis_id = 3
```

Chart settings

- Data Tab
    - Datasource & Chart Type
        * Visualization Type: Bar Chart
    - Time
        * Time range: No filter
    - Query
        * Metrics: SUM(count)
        * Series: Birth_year
        * Breakdowns: name

- Customize Tab
  - Chart Options
    - Stacked Bars: on
    - Sort Bars: on
    - Y Axis Label: Count
    - Extra Controls: on
  - X Axis
    - X Axis Label: Year
    - Reduce X ticks: on

Gender

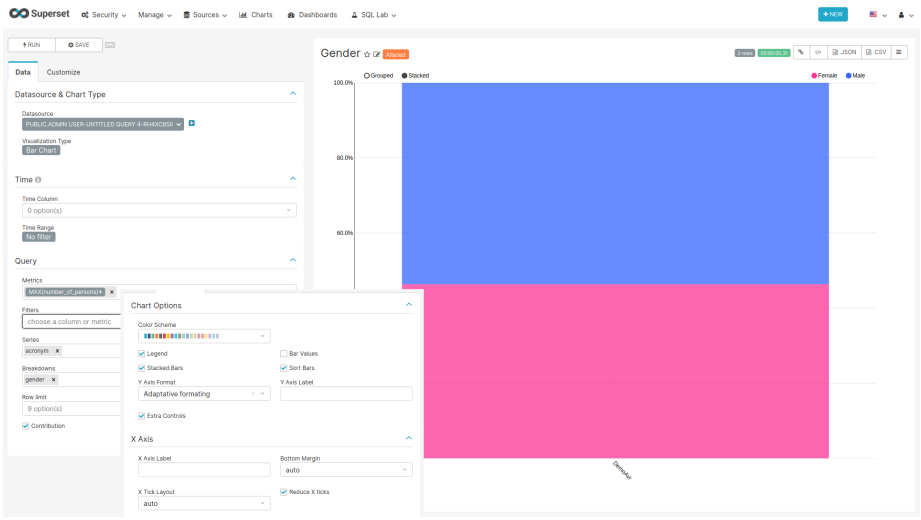


Figure 9.21: Settings for creating the Gender chart

SQL query

```
SELECT source.name,
       concept_name AS Gender,
       count_value AS Number_of_persons,
       source.acronym
FROM public.achilles_results AS achilles
INNER JOIN public.data_source AS source ON achilles.data_source_id=source.id
JOIN (
  SELECT '8507' AS concept_id, 'Male' AS concept_name
  UNION
  SELECT '8532' AS concept_id, 'Female' AS concept_name
) AS concepts ON achilles.stratum_1 = concept_id
WHERE analysis_id = 2
```

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Bar Chart
  - Time
    - Time range: No filter
  - Query
    - Metrics: MAX(Number_of_persons)
    - Series: acronym
    - Breakdowns: gender
    - Contribution: on
- Customize Tab
  - Chart Options
    - Stacked Bars: on
    - Sort Bars: on
    - Extra Controls: on
  - X Axis
    - Reduce X ticks: on

9.5 Observation Period [Deprecated]

CSS

To hide the dashboard header insert the following css code to the `CSS` field on the edit page:

```
.dashboard > div:not(.dashboard-content) {  /* dashboard header */
  display: none;
}
```

With this every time you want to edit the dashboard layout you have to either comment the CSS inserted or remove it so the "Edit Dashboard" button can show again.

Data Source Filter

For the filter to work the name of the fields to filter should match in all tables used on the charts of this dashboard.

SQL query

No SQL query, use the sql table `data_source` of the `achilles` database.

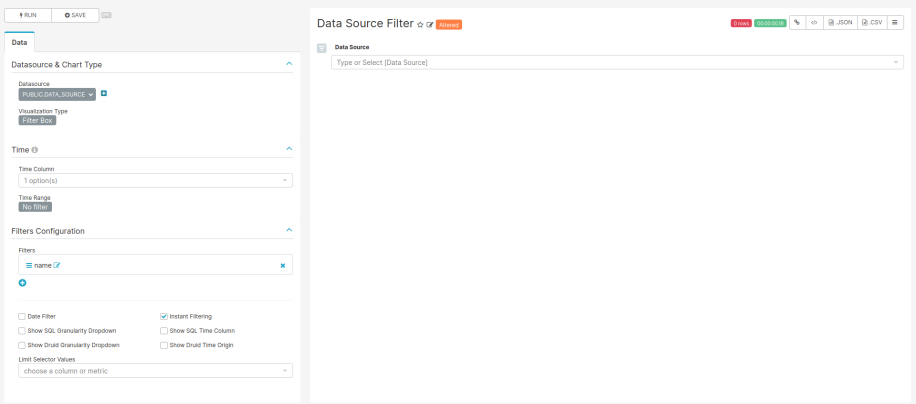


Figure 9.22: Settings for creating the Data Source filter chart

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Filter Box
  - Time
    - Time range: No filter
  - Filters Configuration
    - Filters:
      - name
    - Date Filter: off
    - Instant Filtering: on

Number of Patients in Observation Period {#numInObservationPeriod}

The Number of Patients in Observation Period plot shows the number of patients that contribute at least one day in a specific month.

SQL query

```
SELECT source.name,
       source.acronym,
       to_date(stratum_1, 'YYYYMM') as Date,
       count_value
FROM public.achilles_results AS achilles
INNER JOIN public.data_source AS source ON achilles.data_source_id=source.id
WHERE analysis_id = 110
```

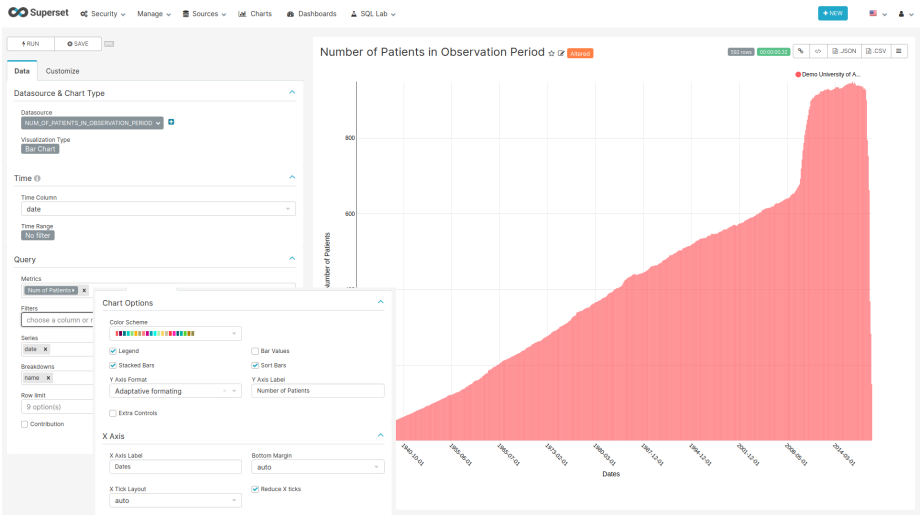


Figure 9.23: Settings for creating the Number of Patients in Observation Period chart

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Bar Chart
  - Time
    - Time range: No filter
  - Query
    - Metrics: MAX(count_value) with label “Num of Patients”
    - Series: date
    - Breakdowns: name
- Customize Tab
  - Chart Options
    - Stacked Bars: on
    - Sort Bars: on
    - Y Axis Label: Number of Patients
  - X Axis
    - X Axis Label: Dates
    - Reduce X ticks: on

Observation Period Start Dates

SQL query

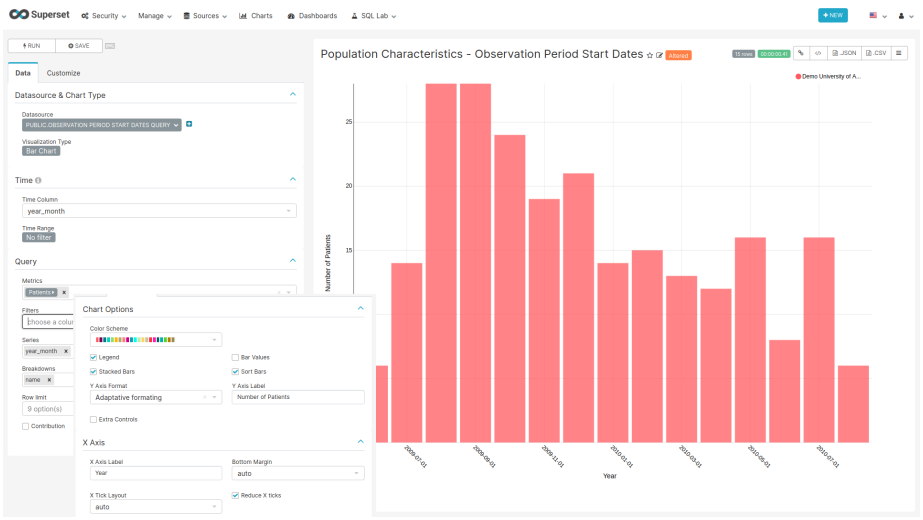


Figure 9.24: Settings for creating the Observation Period Start Dates chart

```
SELECT source.name,
       source.acronym,
       to_date(stratum_1, 'YYYYMM') AS year_month,
       count_value
FROM public.achilles_results AS achilles
INNER JOIN public.data_source AS source ON achilles.data_source_id=source.id
WHERE analysis_id = 111
```

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Bar Chart
  - Time
    - Time range: No filter
  - Query
    - Metrics: SUM(count_value) with label “Patients”
    - Series: year_month
    - Breakdowns: name
- Customize Tab
  - Chart Options
    - Stacked Bars: on
    - Sort Bars: on
    - Y Axis Label: Number of Patients
  - X Axis

* X Axis Label: Year
* Reduce X ticks: on

Observation Period End Dates

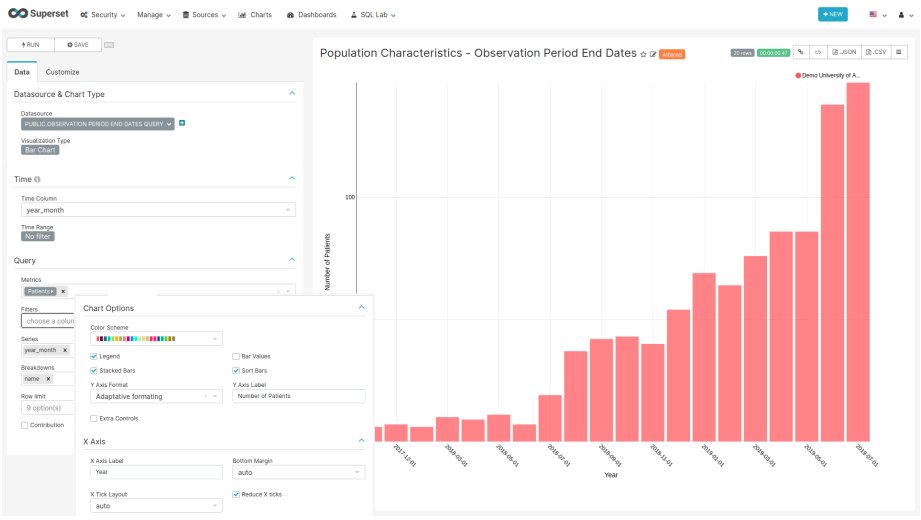


Figure 9.25: Settings for creating the Observation Period End Dates chart

SQL query

```
SELECT source.name,
       source.acronym,
       to_date(stratum_1, 'YYYYMM') AS year_month,
       count_value
FROM public.achilles_results AS achilles
INNER JOIN public.data_source AS source ON achilles.data_source_id=source.id
WHERE analysis_id = 112
```

Chart settings

- Data Tab
  - Datasource & Chart Type
    * Visualization Type: Bar Chart
  - Time
    * Time range: No filter
  - Query
    * Metrics: SUM(count_value) with label “Patients”
    * Series: year_month
    * Breakdowns: name

- Customize Tab
  - Chart Options
    - Stacked Bars: on
    - Sort Bars: on
    - Y Axis Label: Number of Patients
  - X Axis
    - X Axis Label: Year
    - Reduce X ticks: on

9.6 Visit [Deprecated]

This dashboard shows the different types of visits per data source (see Visit Occurence Table)

CSS

To hide the dashboard header insert the following css code to the `CSS` field on the edit page:

```
.dashboard > div:not(.dashboard-content) {  /* dashboard header */
  display: none;
}
```

With this every time you want to edit the dashboard layout you have to either comment the CSS inserted or remove it so the "Edit Dashboard" button can show again.

Data Source Filter

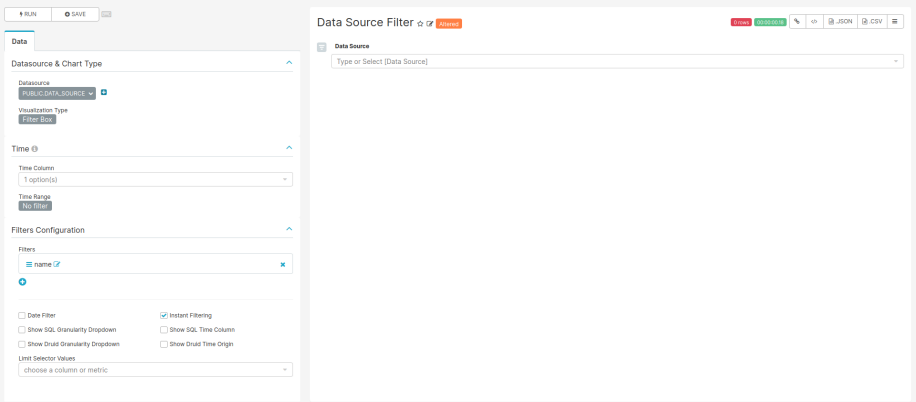


Figure 9.26: Settings for creating the Data Source filter chart

For the filter to work the name of the fields to filter should match in

all tables used on the charts of this dashboard.

SQL query

No SQL query, use the sql table `data_source` of the `achilles` database.

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Filter Box
  - Time
    - Time range: No filter
  - Filters Configuration
    - Filters:
      - name
    - Date Filter: off
    - Instant Filtering: on

Visit Type Table {#visitTypeTable}

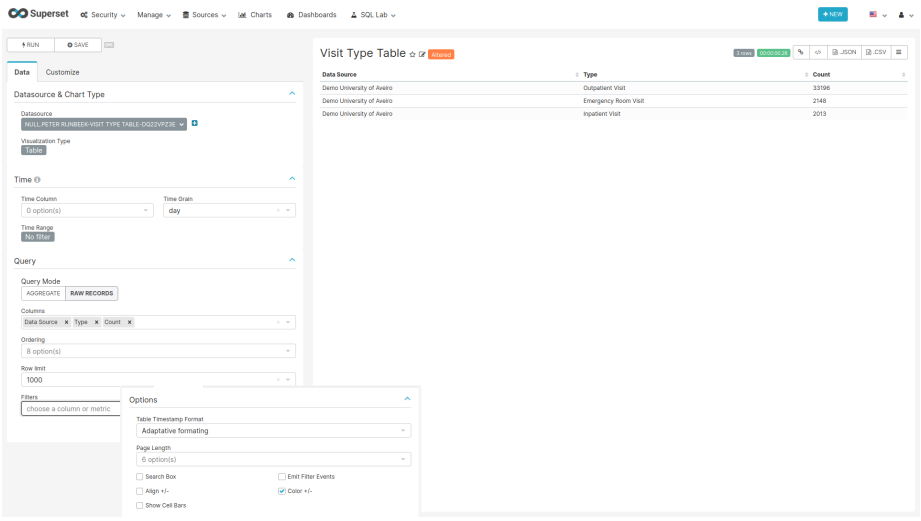


Figure 9.27: Settings for creating the Visit Type Table chart

SQL query

```
SELECT source.name,
       source.acronym,
       concept_name AS "Type",
```

```
       MAX(count_value) AS "Count"
FROM public.achilles_results AS achilles
INNER JOIN public.data_source AS source ON achilles.data_source_id=source.id
INNER JOIN public.concept ON CAST(stratum_1 AS BIGINT) = concept_id
WHERE analysis_id = 201
GROUP BY name, acronym, "Type"
ORDER BY "Count" DESC
```

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Table
  - Time
    - Time range: No filter
  - Query
    - Query Mode: Raw Records
    - Columns: name with label “Data Source”, Type, Count

Visit Types Bars

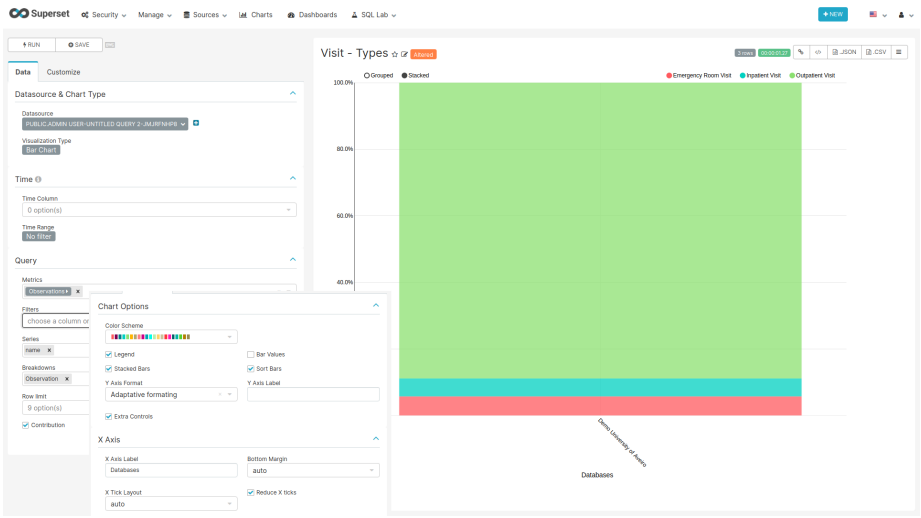


Figure 9.28: Settings for creating the Visit Types bar chart

SQL query

```
SELECT source.name,
       source.acronym,
```

```
       concept_name AS "Observation",
       count_value
FROM public.achilles_results AS achilles
INNER JOIN public.data_source AS source ON achilles.data_source_id=source.id
INNER JOIN public.concept ON CAST(stratum_1 AS BIGINT) = concept_id
WHERE analysis_id = 201
```

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Bar Chart
  - Time
    - Time range: No filter
  - Query
    - Metrics: MAX(count_value) with label Observations
    - Series: name
    - Breakdowns: Observation
- Customize Tab
  - Chart Options
    - Stacked Bars: on
    - Sort Bars: on
    - Extra Controls: on
  - X Axis
    - X Axis Label: Databases
    - Reduce X ticks: on

9.7 Death [Deprecated]

CSS

To hide the dashboard header insert the following css code to the `CSS` field on the edit page:

```
.dashboard > div:not(.dashboard-content) {  /* dashboard header */
  display: none;
}
```

With this every time you want to edit the dashboard layout you have to either comment the CSS inserted or remove it so the “Edit Dashboard” button can show again.

Data Source Filter

For the filter to work the name of the fields to filter should match in all tables used on the charts of this dashboard.

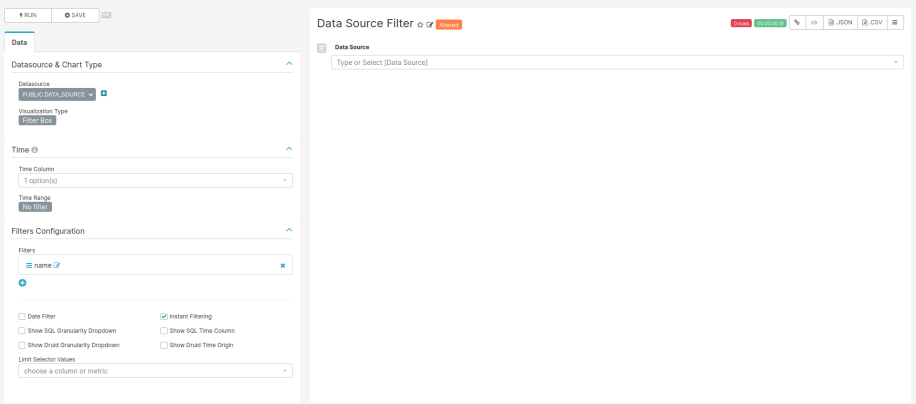


Figure 9.29: Settings for creating the Data Source filter chart

SQL query

No SQL query, use the sql table `data_source` of the `achilles` database.

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Filter Box
  - Time
    - Time range: No filter
  - Filters Configuration
    - Filters:
      - name
    - Date Filter: off
    - Instant Filtering: on

Number of Records

SQL query

```
SELECT source.name,
    count_value,
    source.acronym
FROM public.achilles_results AS achilles
INNER JOIN public.data_source AS source ON achilles.data_source_id=source.id
WHERE analysis_id = 501
```

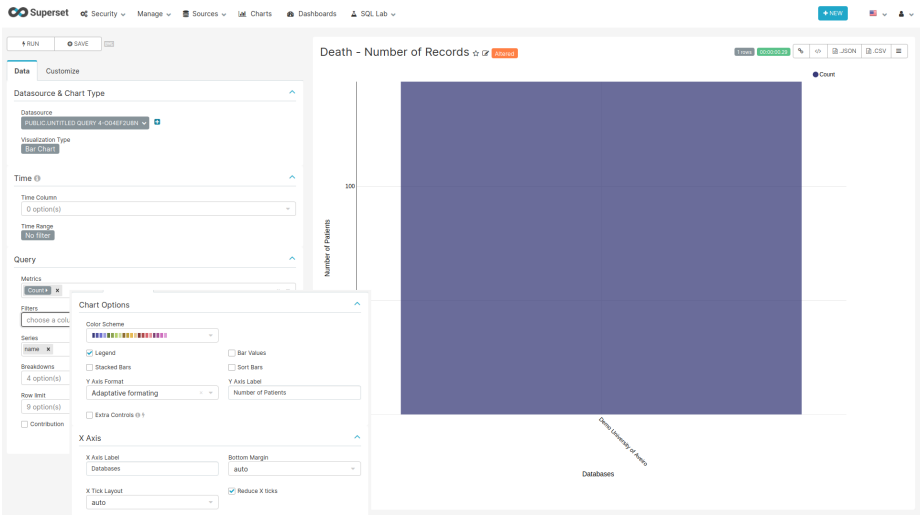


Figure 9.30: Settings for creating the Number of Records chart

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Bar Chart
  - Time
    - Time range: No filter
  - Query
    - Metrics: MAX(count_value) with label Count
    - Series: name
- Customize Tab
  - Chart Options
    - Y Axis Label: Number of Patients
  - X Axis
    - X Axis Label: Databases
    - Reduce X ticks: on

Death By Year per Thousand People

SQL query

```
SELECT source.name,
    source.acronym,
    EXTRACT(year FROM TO_DATE(stratum_1, 'YYYYMM')) AS Date,
    count_value
FROM public.achilles_results as achilles
```

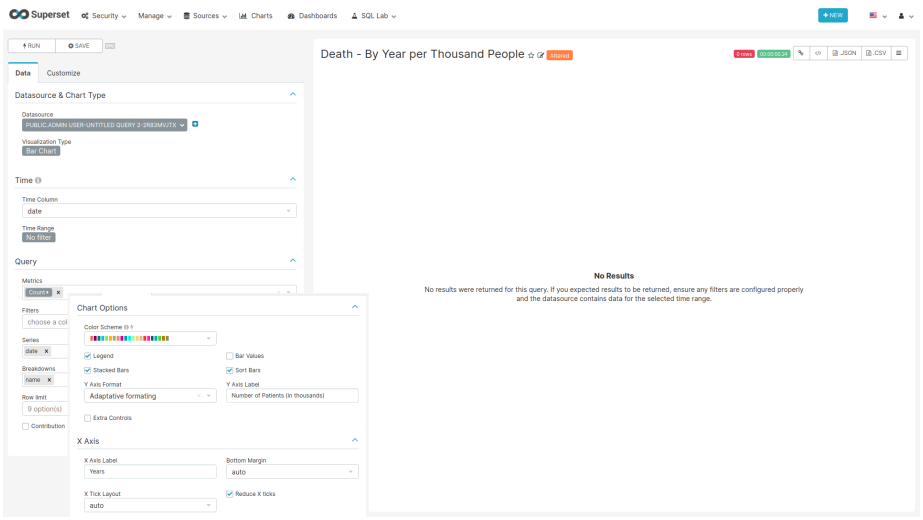


Figure 9.31: Settings for creating the Death by Year per Thousand People chart

```
INNER JOIN public.data_source as source ON achilles.data_source_id=source.id
WHERE analysis_id = 502
```

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Bar Chart
  - Time
    - Time range: No filter
  - Query
    - Metrics: MAX(count_value) with label Count
    - Series: date
    - Breakdowns: name
- Customize Tab
  - Chart Options
    - Stacked Bars: on
    - Sort Bars: on
    - Y Axis Label:Number of Patients (in thousands)
  - X Axis
    - X Axis Label: Years
    - Reduce X ticks: on

9.8 Concepts Browser [Deprecated]

The concepts browser allows you to search for concepts by name or concept_id in all the data sources you select. No exact number of patients or occurrences are provided but the magnitude of both.

CSS

To hide the dashboard header insert the following css code to the `CSS` field on the edit page:

```
.dashboard > div:not(.dashboard-content) {  /* dashboard header */
  display: none;
}
```

With this every time you want to edit the dashboard layout you have to either comment the CSS inserted or remove it so the "Edit Dashboard" button can show again.

Data Source and Domain Filters

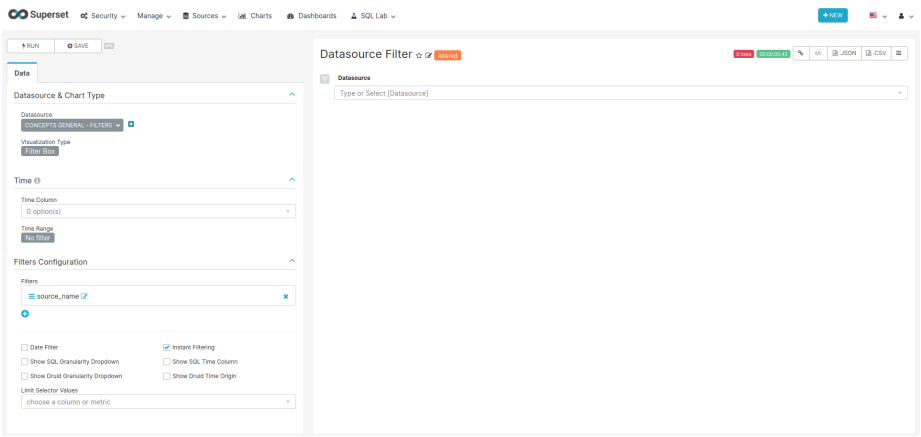


Figure 9.32: Settings for creating the Data Source and Domain filter charts

For the filters to work the name of the fields to filter should match in all tables used on the charts of this dashboard.

SQL query

```
SELECT concept_name,
     domain_id,
     source.name AS source_name,
     source.acronym
```

```
FROM achilles_results
JOIN concept ON cast(stratum_1 AS BIGINT) = concept_id
INNER JOIN public.data_source AS source ON data_source_id=source.id
WHERE analysis_id in (201, 401, 601, 701, 801, 901, 1001, 1801, 200, 400, 600, 700, 80
```

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Filter Box
  - Time
    - Time range: No filter
  - Filters Configuration
    - Filters:
      - source_name or domain_id
    - Date Filter: off
    - Instant Filtering: on

Number of Concepts

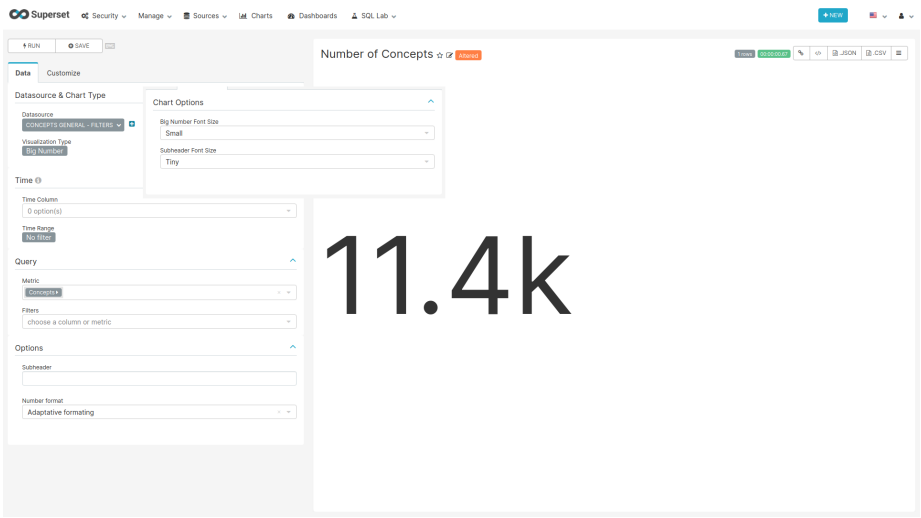


Figure 9.33: Settings for creating the Number of Concepts chart

SQL Query

Same as Data Source and Domain filters query

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Big Number
  - Time Time range: No filter
  - Query
    - Metric: COUNT_DISTINCT(concept_name) with label Concepts
- Customize Tab
  - Big Number Font Size: Small
  - Subheader Font Size: Tiny

Concept Browser Table {#conceptBrowserTable}

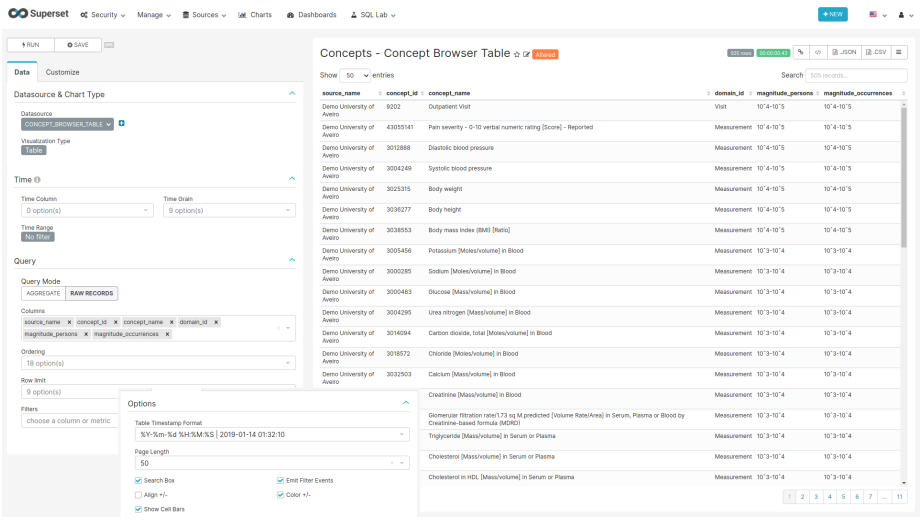


Figure 9.34: Settings for creating the Concepts Table chart

```
SELECT
    q1.concept_id AS concept_id,
    q1.concept_name AS concept_name,
    q1.domain_id,
    source.name AS source_name,
    source.acronym,
    sum(q1.count_value) as "Occurrence_count",
    sum(q1.count_person) as "Person_count",
    CASE
        WHEN sum(q1.count_value)<=10 THEN '<=10'
        WHEN sum(q1.count_value)<=100 THEN '11-10^2'
        WHEN sum(q1.count_value)<=1000 THEN '10^2-10^3'
```

```
        WHEN sum(q1.count_value)<=10000 THEN '10^3-10^4'
        WHEN sum(q1.count_value)<=100000 THEN '10^4-10^5'
        WHEN sum(q1.count_value)<=1000000 THEN '10^5-10^6'
        ELSE '>10^6'
    END as "magnitude_occurrences",
    CASE
        WHEN sum(q1.count_person)<=10 THEN '<=10'
        WHEN sum(q1.count_person)<=100 THEN '11-10^2'
        WHEN sum(q1.count_person)<=1000 THEN '10^2-10^3'
        WHEN sum(q1.count_person)<=10000 THEN '10^3-10^4'
        WHEN sum(q1.count_person)<=100000 THEN '10^4-10^5'
        WHEN sum(q1.count_person)<=1000000 THEN '10^5-10^6'
        ELSE '>10^6'
    END AS "magnitude_persons"
FROM (SELECT analysis_id,
             stratum_1 concept_id,
             data_source_id,
             concept_name,
             domain_id,
             count_value, 0 as count_person
    FROM achilles_results
    JOIN concept ON cast(stratum_1 AS BIGINT)=concept_id
    WHERE analysis_id in (201, 301, 401, 601, 701, 801, 901, 1001, 1801)
    UNION (SELECT  analysis_id,
                   stratum_1 concept_id,
                   data_source_id,
                   concept_name,
                   domain_id,
                   0 as count_value,
                   sum(count_value) as count_person
            FROM  achilles_results
            JOIN concept on cast(stratum_1 as BIGINT)=concept_id
            WHERE analysis_id in (202, 401, 601, 701, 801, 901, 1001, 1801)
            GROUP BY analysis_id,stratum_1,data_source_id,concept_name,domain_id) ) as
    INNER JOIN public.data_source AS source ON q1.data_source_id=source.id
GROUP BY q1.concept_id,q1.concept_name,q1.domain_id,source.name, acronym
ORDER BY "Person_count" desc
```

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Table
  - Time
    - Time range: No filter

  - Query
    - Query Mode: Raw Records
    - Columns: source_name, concept_id, concept_name, domain_id, magnitude_persons, magnitude_occurrences
- Customize Tab
  - Options
    - Table Timestamps Format: %Y-%m-%d %H:%M:%S | 2019-01-14 01:32:10
    - Page Length: 50
    - Search Box: on
    - Emit Filter Events: on

9.9 Provenance [Deprecated]

This Dashboard shows the provenance of the data in the different data domains.

CSS

To hide the dashboard header insert the following css code to the `CSS` field on the edit page:

```
.dashboard > div:not(.dashboard-content) {  /* dashboard header */
  display: none;
}
```

With this every time you want to edit the dashboard layout you have to either comment the CSS inserted or remove it so the “Edit Dashboard” button can show again.

Data Source Filter

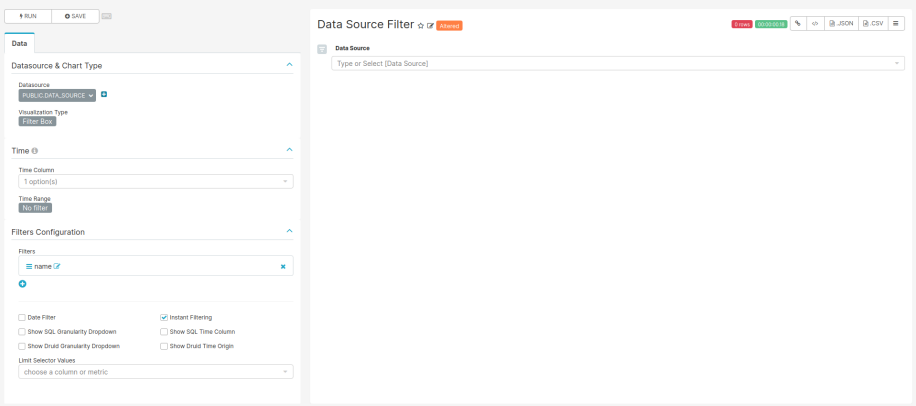


Figure 9.35: Settings for creating the Data Source filter chart

For the filter to work the name of the fields to filter should match in all tables used on the charts of this dashboard.

SQL query

No SQL query, use the sql table `data_source` of the `achilles` database.

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Filter Box
  - Time
    - Time range: No filter
  - Filters Configuration
    - Filters:
      - name
    - Date Filter: off
    - Instant Filtering: on

Condition & Drug & Procedure & Device & Measurement & Observation Types {#dataProvenanceCharts}

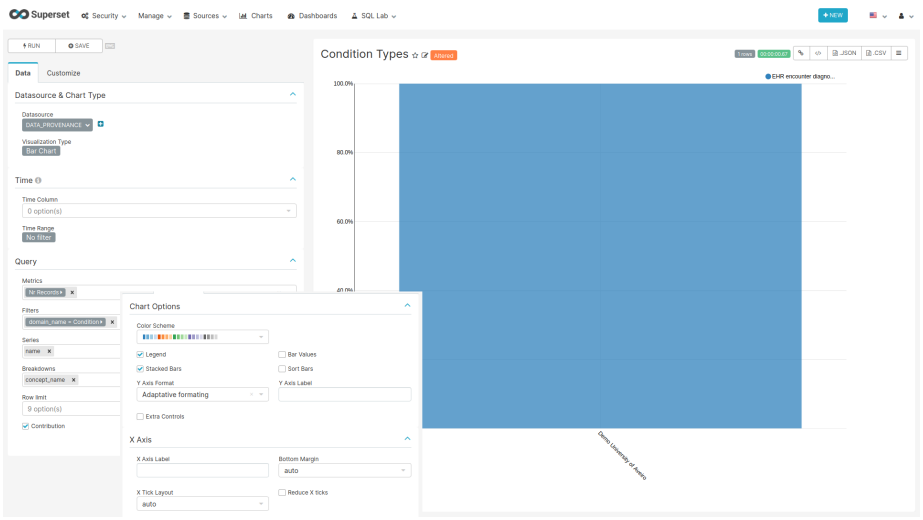


Figure 9.36: Settings for creating the Condition, Drug, Procedure, Device, Measurement and Observation charts

SQL query

All 6 charts use the same sql query.

```
SELECT source.name,
    source.acronym,
    CASE WHEN analysis_id = 405 THEN 'Condition'
    WHEN analysis_id = 605 THEN 'Procedure'
    WHEN analysis_id = 705 THEN 'Drug'
    WHEN analysis_id = 805 THEN 'Observation'
    WHEN analysis_id = 1805 THEN 'Measurement'
    WHEN analysis_id = 2105 THEN 'Device'
    ELSE 'Other' END AS domain_name,
    concept_name,
    SUM(count_value) AS num_records
FROM public.achilles_results AS achilles
INNER JOIN public.data_source AS source ON achilles.data_source_id=source.id
INNER JOIN public.concept AS c1 ON CAST(stratum_2 AS BIGINT) = concept_id
WHERE analysis_id IN (405,605,705,805,1805,2105)
GROUP BY source.name, source.acronym, concept_name,
    CASE WHEN analysis_id = 405 THEN 'Condition'
    WHEN analysis_id = 605 THEN 'Procedure'
    WHEN analysis_id = 705 THEN 'Drug'
    WHEN analysis_id = 805 THEN 'Observation'
    WHEN analysis_id = 1805 THEN 'Measurement'
    WHEN analysis_id = 2105 THEN 'Device'
    ELSE 'Other' END
```

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Bar Chart
  - Time
    - Time range: No filter
  - Query
    - Metrics: SUM(num_records) with label Nr Records
    - Filters: domain_name=Condition or domain_name=Drug or domain_name=Procedure or domain_name=Device or domain_name=Measurement or domain_name=Observation
    - Series: name
    - Breakdowns: concept_name
    - Contribution: on
- Customize Tab
  - Chart Options
    - Stacked Bars: on

9.10 Data Domains [Deprecated]

CSS

To hide the dashboard header insert the following css code to the `CSS` field on the edit page:

```
.dashboard > div:not(.dashboard-content) {  /* dashboard header */
  display: none;
}
```

With this every time you want to edit the dashboard layout you have to either comment the CSS inserted or remove it so the "Edit Dashboard" button can show again.

Data Source Filter

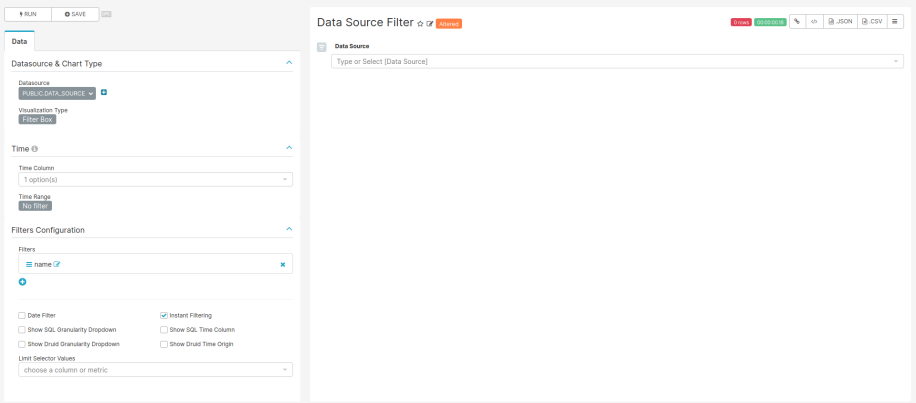


Figure 9.37: Settings for creating the Data Source filter chart

For the filter to work the name of the fields to filter should match in all tables used on the charts of this dashboard.

SQL query

No SQL query, use the sql table `data_source` of the `achilles` database.

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Filter Box
  - Time
    - Time range: No filter

- Filters Configuration
  - Filters:
    - name
  - Date Filter: off
  - Instant Filtering: on

Average Number of Records per Person {#avgRecordsPerPerson}

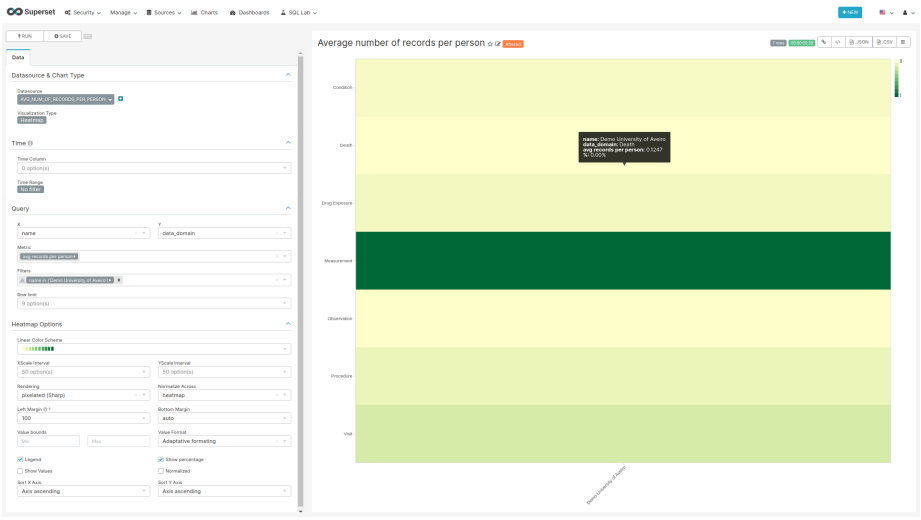


Figure 9.38: Settings for creating the Data Source filter chart

SQL query

```
SELECT
    source.name,
    source.acronym,
    CASE
    WHEN analysis_id = 201 THEN 'Visit'
    WHEN analysis_id = 401 THEN 'Condition'
    WHEN analysis_id = 501 THEN 'Death'
    WHEN analysis_id = 601 THEN 'Procedure'
    WHEN analysis_id = 701 THEN 'Drug Exposure'
    WHEN analysis_id = 801 THEN 'Observation'
    WHEN analysis_id = 1801 THEN 'Measurement'
    WHEN analysis_id = 2101 THEN 'Device'
    WHEN analysis_id = 2201 THEN 'Note'
    END AS Data_Domain,
```

```
    SUM(count_value) /AVG(num_persons) AS "records_per_person"
FROM public.achilles_results AS achilles
INNER JOIN public.data_source AS source ON achilles.data_source_id=source.id
INNER JOIN (
  SELECT data_source_id , count_value as num_persons
  FROM achilles_results
  WHERE analysis_id = 1) counts ON achilles.data_source_id = counts.data_source_id
GROUP BY analysis_id, source.name, source.acronym
HAVING analysis_id IN (201, 401, 501, 601, 701, 801, 1801, 2101, 2201)
```

Chart settings

- Data Tab
  - Datasource & Chart Type
    - Visualization Type: Heatmap
  - Time
    - Time range: No filter
  - Query
    - X: name
    - Y: data_domain
    - Metric: AVG(records_per_person) with a label avg records per person
    - Row limit: None
  - Heatmap Options
    - Left Margin: 100
    - Show Percentage: off