DATA ENGINEERING (LTAT.O2.OO7)
GROUP I2
Timofei Ganjušev,
Siim Karel Koger,
Ida Rahu,
Ihar Suvorau

FINAL REPORT

"Data is like garbage. You'd better know what you are going to do with it before you collect it."
— Mark Twain

In this project, a data pipeline to analyse data about scientific publications was built. In figure 1, the overall pipeline design is shown. As one can see, the pipeline can be divided into three main parts/stages and the first part additionally into two subparts (1A so-called transformation pipeline and 1B so-called updating pipeline). In the first part, 1A, the new raw data in JSON format is ingested into the pipeline, where it is transformed and enriched and then loaded into the up-to-date database. In subpart 1B, the data in the up-to-date database is updated periodically. Also, in the first part of the pipeline, the data is prepared in the correct form (the CSV files are written), so it can be uploaded into the data warehouse (DWH) (part 2) and the graph database (DB) (part 3). In the final report, all these stages are thoroughly covered, including the DWH and graph DB designs, together with relevant queries that DWH and graph DB will answer. And last but not least, the guidelines for running the built data pipeline are given.
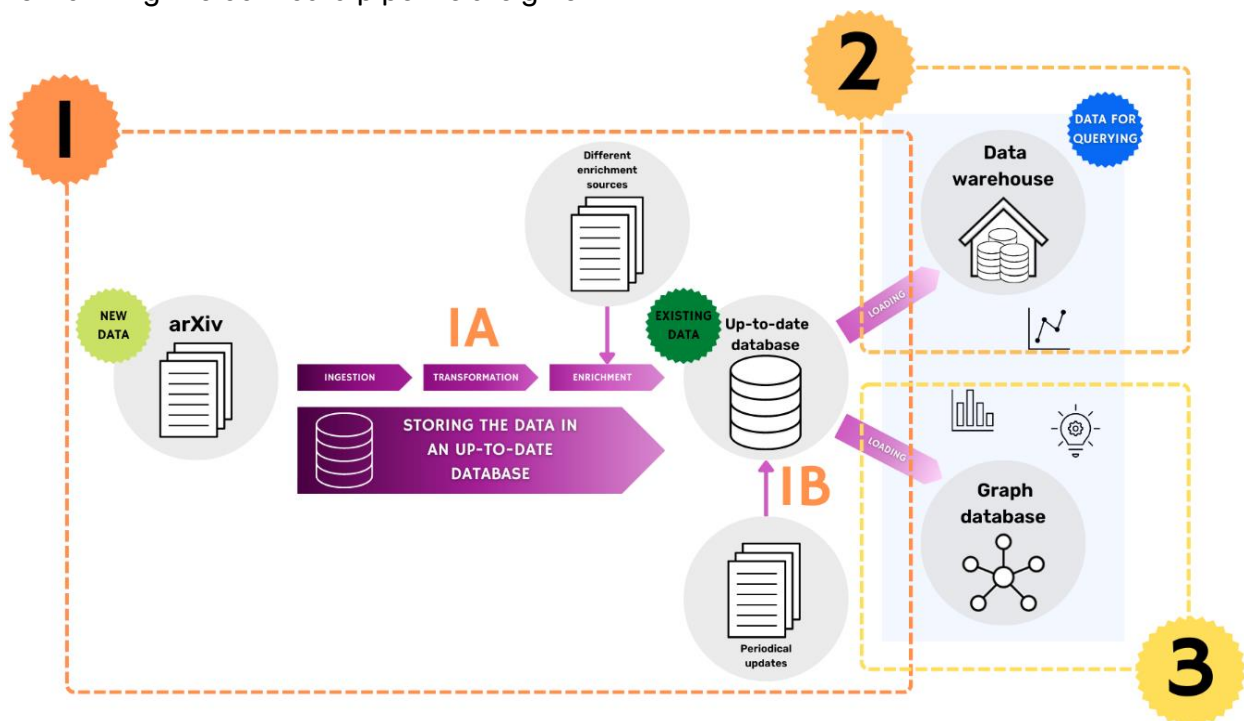

Figure 1 Design of the overall data pipeline

Part 1. Transformation and updating pipelines
Dataset and pre-pipeline processing

ArXiv is the open-access archive for scholarly articles from a wide range of different scientific fields. The dataset given for the project contains metadata of the original arXiv data, *i.e.* metadata of papers in the arXiv. The metadata is given in JSON format and contains the following fields:

- `id` – publication arXiv ID
- `submitter` – the name of the person who submitted the paper/corresponding author
- `authors` – list of the names of the authors of the paper (in some cases, this field includes additional information about the affiliations of the authors)
- `title` – the title of the publication
- `comments` – additional information about the paper (such as the number of figures, tables and pages)
- `journal-ref` – information about the journal where the article was published
- `doi` – Digital Object Identifier (DOI) of the paper
- `report-no` – institution's locally assigned publication number
- `categories` – categories/tags in the arXiv system, i.e. field of the current study
- `license` – license information
- `abstract` – abstract of the publication
- `versions` – history of the versions (version number together with timestamp/date)
- `update_date` – timestamp of the last update in arXiv
- `authors_parsed` – previous authors field in the parsed form

In this project's scope, the fields of `submitter`, `title`, `journal-ref`, `doi`, `categories`, `versions` and `authors_parsed` were used (all the others were dropped).

The original dataset contains information about more than 2 million publications. The original dataset was split into smaller parts (44 parts in total, each containing information about 50K publications) to simulate a real-life situation where new data is coming in continuously. (The dataset splitting was conducted using the `split_dataset.py` script.) Also, before ingesting the data into the pipeline, the publications whose data will be enriched were selected. This preselection was made because the enrichment process via API calls is very time-consuming; therefore, enriching all the publications' data is out of this project's scope. (While running the transformation pipeline, this difference is vividly illustrated when one compares the average run times between the first (ingested raw data contains the publication whose data is enriched via API calls) and all the other runs (data is not enriched via API calls).) For picking out the publications for enrichment, the publications with DOI (needed for API calls) were first filtered out. Afterwards, they were grouped by categories (major field + sub-category, see table 1), and 20 publications were selected from each category. (The details about how publications have been divided into these categories are given in the next chapter.) Finally, their DOIs were written into the `DOIs_for_enrichment.csv` file. This file is used in the transformation pipeline, as explained in the next chapter.

Table 1 Categories (major field together with one sub-category forms one category) that are used for selecting the publications that are going to be enriched

Major field	Sub-category
natural science	• mathematics • computer sciences • information sciences • earth sciences • biology science • physical sciences • chemical sciences
engineering and technology	• electrical, electronic and information engineering • mechanical engineering
social sciences	• economics, finance and business

Transformation pipeline (1A)

As mentioned before, the first part of the overall pipeline can be divided into two subparts: transformation pipeline (1A) and updating pipeline (1B). This chapter covers the concepts of the transformation pipeline. The details about the updating pipeline are given in the next chapter.

In the transformation pipeline, the raw data in JSON format (as explained in the previous chapter: 50K publications at once) is ingested into the overall pipeline; then, this data is cleaned and enriched if needed and loaded into the up-to-date database. Because this project uses the approach where the data will be stored in the up-to-date database to simulate the real-life setup where constant data updates are required because of its volatile nature (the number of citations of the publications changes and so on) before discussing all the details about transformation pipeline, main concepts of this database are given. The schema of this DB is shown in figure 2.
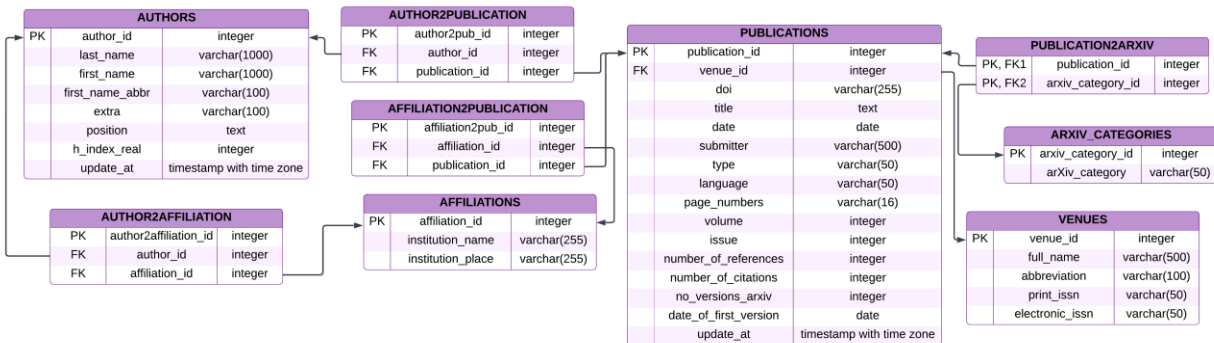

Figure 2 Schema of an up-to-date database

As one can see, the up-to-date DB consists of 9 relations. The table "AUTHORS" contains all the relevant information, such as last name, first name, the abbreviation of the first name, extra titles (such as Jr., I, II *etc*.), positions, and real-life h-index, about the publications' authors in the DB. In the table "AFFILIATIONS", the institution names and places where the authors of publications work are gathered. In the table "PUBLICATIONS", the data about publications (such as DOI, title, publishing date of first and last version,

name of the submitter, type and language of publication, volume, issue and page numbers, number of references and citations) are recorded. The relation "ARXIV_CATEGORIES" contains information about publications' arXiv categories. In the relation "VENUES", all the data about venues (full name, abbreviation, print and electronic ISSN) where publications were published are collected. All other relations are created to deal with the entities' m:n relationships.

The Airflow DAG `transform_create_tables` (see figure 3) was written to create this Postgres database. This DAG should be run only once at the very beginning of the overall pipeline. During the run, all the database tables are generated. Besides, all the other SQL statements needed in the first part of the overall pipeline are also generated. These SQL statements include the ones used for populating the tables with new data or updating the existing data (considering all the constraints that are present in the DB), together with the statements used for generating the views (authors' view and venues' view) that are needed for getting the data in the correct form to load into the DWH and graph DB. (The mentioned authors' view contains authors' IDs, last, first, and full names, first name abbreviations, positions, real-life h-indices, and the h-indices calculated based on the data present in the DB. In the venues' view, besides the data that this in the "VENUES" table (venue ID, full name, abbreviation, print and electronic ISSN), also the calculated h-index for each venue is given.)
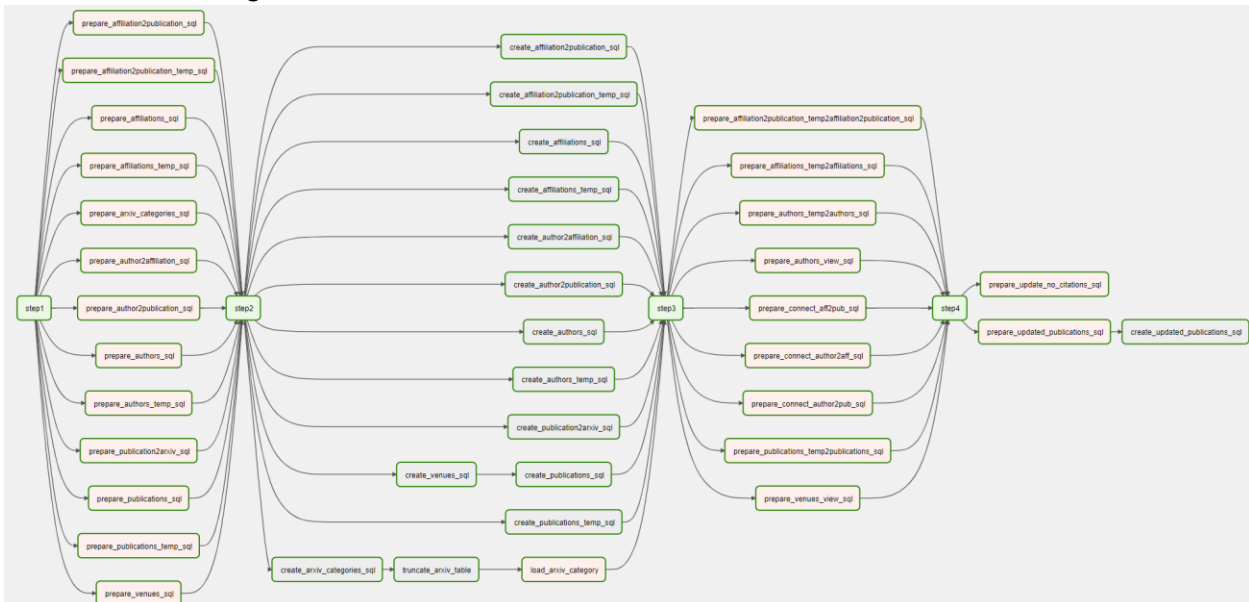

Figure 3 Airflow DAG `transform_create_tables`

The transformation pipeline itself is implemented as an Airflow DAG `transform_articles`. The graph representation of this DAG is shown in the following image.
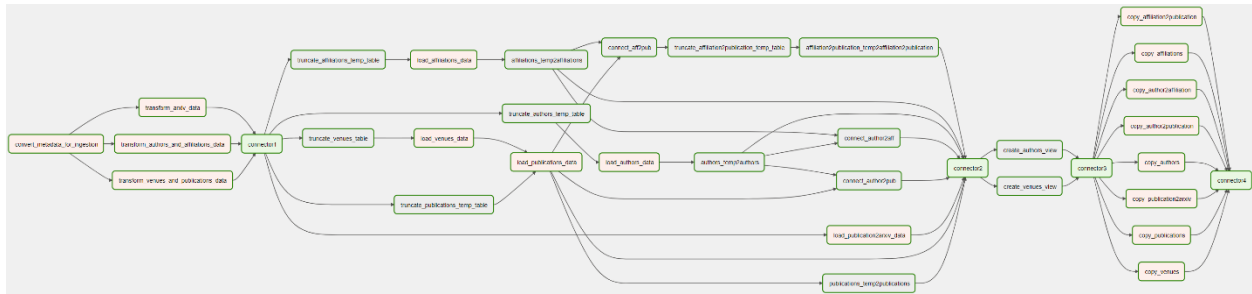
Figure 4 Airflow DAG `transform_articles`

The prerequisite for running this DAG is that the "data" folder should contain seven subfolders: "inputs", "setups", "metadata", "lookup_tables", "data2db", "sql", and "final_data" (see table 2).

Table 2 Data folders required for running the transformation pipeline

Subfolder	Explanation
`inputs`	The raw JSON datasets generated during pre-pipeline processing by splitting the original arXiv dataset are stored in this folder.
`setups`	It contains three files: "DOIs_for_enrichment.csv", "publication_ID.txt", and "split_no.txt". "DOIs_for_enrichment.csv" contains DOIs of the articles, which data will be enriched during the transformation pipeline (see chapter "Dataset and pre-pipeline processing"). The file "publication_ID.txt" is used for storing the last publication ID to ensure that all the publication IDs are unique. Before the first run of the pipeline, the ID in this file should be 0. The file "split_no.txt" is used for a similar purpose – storing the number of the previously used dataset to ensure that all datasets are ingested only once. Before the first run of the pipeline, the split number should be 0.
`metadata`	Before the first run of the `transform_articles` DAG, this folder should be empty. After, it should always contain one file, "metadata_df.tsv". This file is generated in task `convert_metadata_for_ingestion`, during which the relevant fields (see the previous chapter) for each publication are filtered out from the raw JSON file. (The received data, ready for further usage in the transformation pipeline, is stored in this TSV file.)
`lookup_tables`	It contains four tables: "cities_lookup.tsv", "lookup_table_domains.csv", "universities_lookup.tsv", and "venues_lookup.tsv". All these tables (except "lookup_table_domains.csv") are used for enriching and transforming the raw data. For enriching and transforming the: • affiliations' data the "cities_lookup.tsv" and "universities_lookup.tsv" are used. In the "cities_lookup.tsv", the names of 42905 world's largest cities (including the Unicode and ASCII strings) together with names of countries are given. This data was retrieved from SimpleMaps.com (https://simplemaps.com/data/world-cities). In the "universities_lookup.tsv", the data (names, countries, cities/locations and abbreviations of locations) of about 1152 world universities are stored. This data was retrieved by merging and preprocessing the World University Rankings (https://www.kaggle.com/datasets/mylesoneill/world-university-rankings) dataset and University Rankings 2017 (https://data.world/education/university-rankings-2017) dataset. • venues' data the "venues_lookup.tsv" is used. This dataset contains information such as the name and abbreviation (with and without dots) of 87224 venues. This dataset was retrieved from the Web Of Science (https://images.webofknowledge.com/images/help/WOS/A_abrvjt.html). To make the enrichment process easier and faster, the dataset was preprocessed,

	and therefore the lookup table also contains fields `abbrev_check` and `full_check`. The lowercase abbreviations and full names of venues without spaces are stored in these fields. The file "`lookup_table_domains.csv`" is used in task `copy_publication2arxiv`. Comprehensive information about scientific domains should be provided to populate the DWH and graph DB with data. However, in up-to-date DB, only arXiv categories are stored (see figure 2). Therefore, the data copied from the DB needs to be enriched before it can be used for DWH and graph DB. The "`lookup_table_domains.csv`" is employed for that purpose. (More details are provided below.)
`data2db`	The data that will be loaded into the up-to-date database are stored in this folder. Before the first run, it should contain two files: "`arxiv_categories.csv`" and "`venues_df.tsv`". By the way, the latter should contain only the header row (`venue_ID\tfull_name\tabbreviation\tprint_issn\telectronic_issn`) before the first run of the pipeline. In the "`arxiv_categories.csv`", the IDs and arXiv category names of 156 categories are stored.
`sql`	This folder is used for storing all the SQL statements. Before the first run, the folder should be empty, but after running the `transform_create_tables` DAG, the .sql files are generated and saved in this folder.
`final_data`	This folder should contain the data in the correct form, ready to be loaded into the DWH and graph DB. Before the first run of the pipeline, this folder should be empty.

The transformation pipeline starts with the task `convert_metadata_for_ingestion` (see figure 5). During this task, the new dataset (a 50K subset of the original dataset) is selected and ingested into the pipeline.
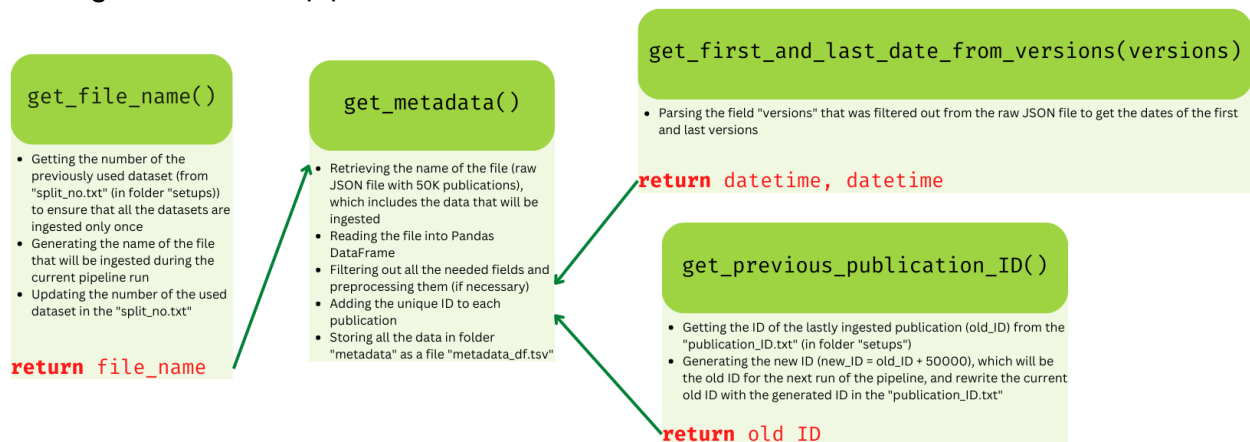

Figure 5 Schema of the task `convert_metadata_for_ingestion`

Firstly, the fields `submitter`, `title`, `journal-ref`, `doi`, `categories`, `versions` and `authors_parsed` are filtered out from the raw JSON file. Then the values of these fields are preprocessed if necessary. For example, the field `versions` is used to get the date of the first and last version and the total number of versions in arXiv required in the up-to-date DB in the table "PUBLICATIONS". Besides, a new unique ID (publication_ID) is generated for each publication. At the end of the `convert_metadata_for_ingestion`, all the relevant information retrieved from the original dataset is saved as a file "`metadata_df.tsv`" (in the folder "metadata"). This file contains the fields `publication_ID`, `submitter`, `authors`, `title`, `journal_ref`, `doi`, `categories`, `no_versions_arxiv`, `date_of_first_version`, and `date`. Some of the values of these fields (`submitter`, `doi`, `no_versions_arxiv`, `date_of_first_version`, and `date`) are used without further processing. (These values are

stored in their present form in up-to-date DB in the table "PUBLICATIONS"). The file "metadata_df.tsv" is used for passing the data to the following tasks.

The next three tasks that run in parallel, transform_venues_and_publications_data (figure 6), transform_arxiv_data (figure 7) and transform_authors_and_affiliations_data (figure 8), can be considered the most critical tasks in the transformation pipeline. The main cleaning, transformation, and enrichment processes are performed during these tasks.
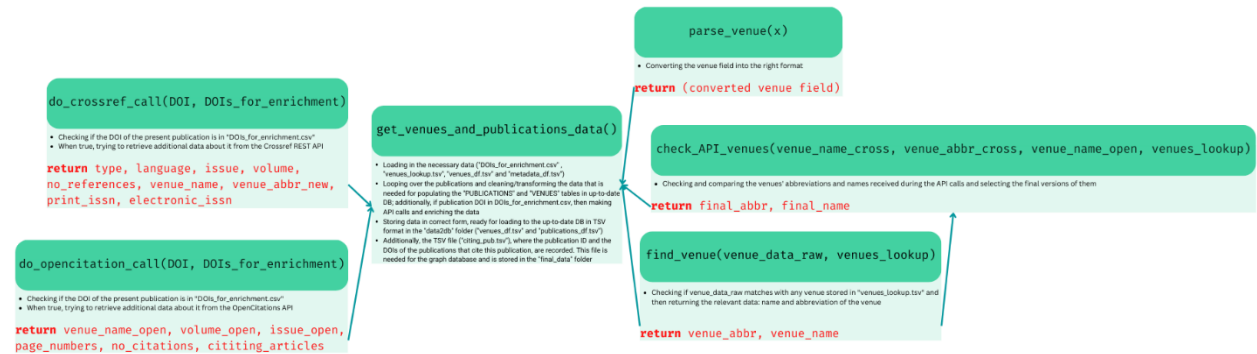

Figure 6 Schema of the task transform_venues_and_publications_data

During the task transform_venues_and_publications_data, the following steps are carried out.

1. The title of the publication is cleaned (the symbols of the newline are removed).
2. To get the information about venues (name and abbreviation), the field journal_ref (present in "metadata_df.tsv") is used. The data in this field is cleaned and then checked if it matches with any venue in the lookup table "venues_lookup.tsv" (by using the function find_venue(venue_data_raw)). (At the end, these values are stored in up-to-date DB in the table "VENUES").
3. The field doi is used for getting additional information (its type, number of references, citations/number of citations, page numbers, and different attributes relevant to journal articles, such as an issue number) about the publication. This method for enrichment is considered only when the DOI of publication is in the file "DOIs_for_enrichment.csv", as mentioned before (see chapter "Dataset and pre-pipeline processing"). Two APIs are used to retrieve a piece of extra information: Crossref REST API[1] and OpenCitations API[2] (see the following examples).

```python
# Example of retrieving the publication type based on DOI by using the Crossref REST API

crossref_results = crossref_commons.retrieval.get_publication_as_json('10.1103/PhysRevA.75.043613')
print(crossref_results['type'])

# Output:
journal-article

# Example of retrieving the number of references based on DOI by using the Crossref REST API

crossref_results = crossref_commons.retrieval.get_publication_as_json('10.1103/PhysReVA.75.043613')
print(crossref_results['reference-count'])

# Output:
23
```

```
# Example of retrieving the number of citations based on DOI by using the OpenCitations API

client = opencitingpy.client.Client()
open_citation_result = client.get_citation_count('10.1103/PhysReVA.75.043613')
print(open_citation_result)

# Output:
13
```

4. The unique ID is generated for each venue (field venue_ID). If the publication does not have information about the venue available, the 0 is used as a venue_ID.
5. The TSV files ("venues_df.tsv" and "publications_df.tsv") needed for populating the tables "VENUES" and "PUBLICATIONS" in up-to-date DB are written. Additionally, the TSV file ("citing_pub.tsv"), where the publication ID and the DOIs of the publications that cite this publication, are recorded. (This file is required for the graph database.)

During the task transform_arxiv_data, the field categories in "metadata_df.tsv" is used to map each publication with arXiv categories (stored in "arxiv_categories.csv"). The received information is saved as file "publication2arxiv_df.tsv". In the end, arXiv categories are stored in up-to-date DB in the table "ARXIV_CATEGORIES" and mapping data in the table "PUBLICATION2ARXIV".
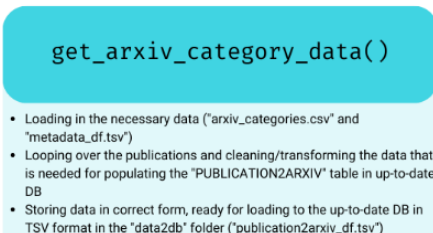
get_arxiv_category_data()

- Loading in the necessary data ("arxiv_categories.csv" and "metadata_df.tsv")
- Looping over the publications and cleaning/transforming the data that is needed for populating the "PUBLICATION2ARXIV" table in up-to-date DB
- Storing data in correct form, ready for loading to the up-to-date DB in TSV format in the "data2db" folder ("publication2arxiv_df.tsv")

Figure 7 Schema of the task transform_arxiv_data

And last but not least, the following list explains the steps carried out during the task transform_authors_and_affiliations_data.

1. The relevant fields, such as publication_ID and authors_parsed, are filtered out from "metadata_df.tsv".
2. The field authors_parsed contains information about the authors' names. Therefore, it is processed (by using functions check_first_name_raw(first_name_raw_to_check) and parse_first_name(first_name_raw_to_parse)) to get the last and first names together with the abbreviation of each author's first name. Sometimes, this field also contains some extra suffixes of the name, such as Jr or II *etc.*, which are also stored. (To check the extra suffixes, the function extra_or_affiliation(value1, value2) is employed.) (At the end, these values are stored in up-to-date DB in the table "AUTHORS").
3. The field authors_parsed can also include data about authors' affiliations. Thus, the function find_insitution_information(institution_name_raw, universites_lookup, cities_lookup) is called to find whether this raw data field matches any institution or location stored in the "universities_lookup.tsv" or "cities_lookup.tsv". It is important to note that data about location/place after the

8

transformation pipeline is always at the country level. (At the end, these values, institution name and place, are stored in up-to-date DB in the table "AFFILIATIONS").

4. *To enrich the authors' information (to receive their real-life h-indices or position), the scholarly[3] (and function do_scholarly_call(author)) is used. This module allows retrieving the authors' and publications' information from Google Scholar.*

```python
# Example of retrieving the author's real-life h-index by using a scholarly module

search_query = scholarly.search_author('S. Wichmann')
first_author_result = next(search_query)
author = scholarly.fill(first_author_result)
print(author['hindex'])

# Output:
39
```

However, since scholarly has a limited number of times to retrieve the data, this part of the pipeline's code is now commented in to prevent it from running. Therefore, the relevant lines in the function get_authors_and_affiliations_data() should be commented out before running the pipeline if one wants to use scholarly.

5. The TSV files ("authors_df.tsv" and "affiliations_df.tsv") needed for populating the tables "AUTHORS" and "AFFILIATIONS" in up-to-date DB are written.
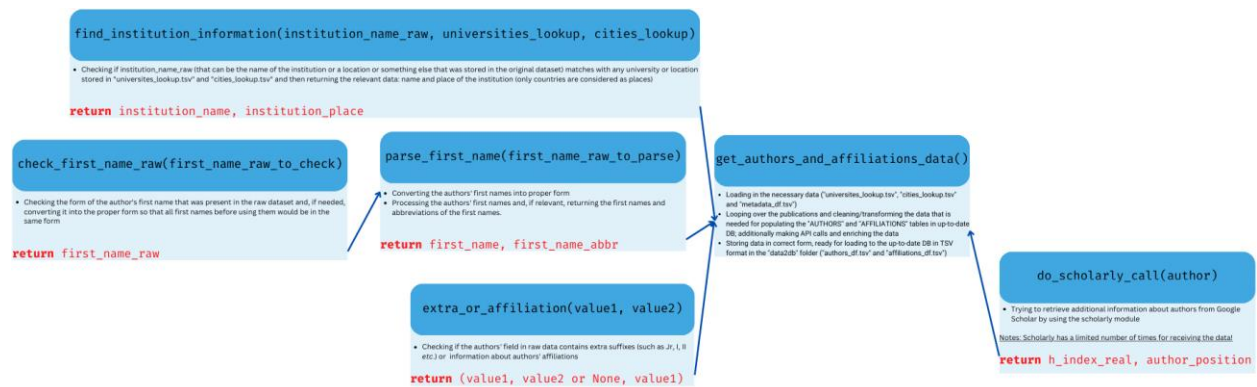

Figure 8 Schema of the task transform_authors_and_affiliations_data

(P.S. During these three tasks, all the missing strings are replaced with None and integers with -1.)

After these tasks, the data is loaded into the up-to-date DB. In the cases where it is necessary to check that only new data is loaded to ensure that there would be no duplicates in the DB, the additional temporary tables (such as "AFFILIATION2PUBLICATION_TEMP", "AFFILIATIOS_TEMP", "AUTHORS_TEMP" and "PUBLICATIONS_TEMP") are used. (These tables are always emptied before loading a new batch of data into the DB.) To illustrate how the temporary tables are used, the following example is given. Initially, data about authors is bulk inserted into the "AUTHORS_TEMP" table. Then the data of each author is compared with the data of each author in the "AUTHORS" table. If the author's information is not already present in the DB, a new author is inserted into the "AUTHORS" table. Otherwise, the data about the author is discarded. The same comparison is made between all the corresponding temporary and permanent tables.

In the last step of the transformation pipeline, the data in the database is copied and stored in CSV files. These files are used for loading the data into DWH and graph DB. Appropriate views are generated beforehand to get all the required information (like calculated h-indices of the venues and authors). Also, the field of study is normalised. For that, each arXiv category is mapped against the Scientific Disciplines classification table[4]. (This table is stored in suitable form in "`lookup_table_comains.csv`".) For example, if the value in the arXiv category field is "`cs.AI`" after the mapping, besides this tag, there are three new tags: `major_field`: "natural sciences", `sub_category`: "computer sciences" and `exact_category`: "artificial intelligence".

Updating pipeline (1B)

To fulfil the prerequisites of using the up-to-date DB approach, the data about publications in the database should be updated periodically (for example, monthly). For that reason, the Airflow DAG `transform_periodic_update` was built (see figure 9).
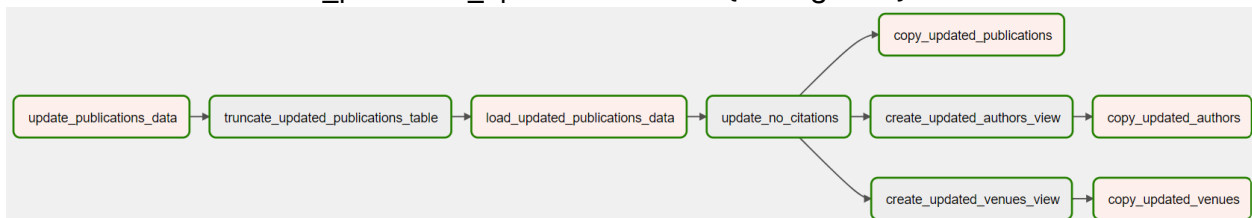

Figure 9 Airflow DAG `transform_periodic_update`

During the run of this DAG, the publications with DOIs, that are stored in DB are updated by using their DOIs and OpenCitations API. In this project, only the number of citations is considered as changing field. If the API call returns a new value for this variable, the data about publication is updated. Since venues' and authors' h-indices depend on the number of citations, the relevant views are refreshed after updates.

Similarly to the transformation pipeline, updating the pipeline ends with copying the data. However, at this time, only publications', authors' and venues' data is copied and saved as CSV files ready for the following pipeline parts (other tables in DB do not change).

(Since API calls are very time-consuming, additional DAG `transform_periodic_update_presentation` was generated. This DAG is a copy-paste version of the DAG update_articles_in_DB. The only difference is that instead of pulling all publications' data from DB, it limits the number of publications to 200. This DAG has only the illustrative purpose of how the updates should work – updating all the publications at once is out of this project's scope.)

Part 2. Data warehouse
Design

The data warehouse built in this project has the following schema.
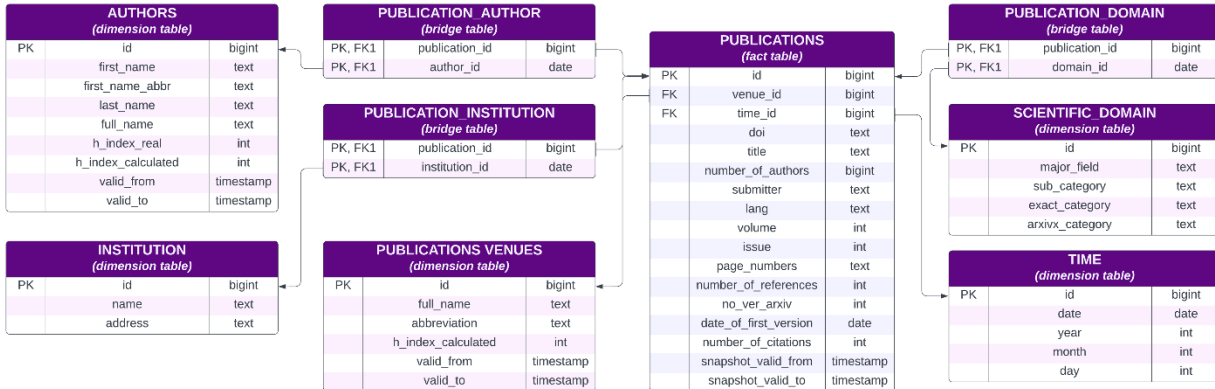

Figure 10 Schema of DWH

As one can see, minor changes are made compared to the schema presented in "Design Document – GROUP 12 ".

The proper schema of a data warehouse for storing data about scientific publications contains a fact table, "PUBLICATIONS", and five dimension tables: "AUTHORS", "INSTITUTION", "PUBLICATION VENUES", "SCIENTIFIC DOMAINS" and "TIME".

The fact table "PUBLICATIONS" stores the primary keys of dimension tables as foreign keys together with additional information about the record (see table 3 for more details).

Table 3 Attributes of the fact table together with explanations

Attribute	Explanation
id	generated primary key for fact table
venue_id	the primary key of the "PUBLICATION VENUES" dimension; is required to retrieve data about venues where the paper was published
time_id	the primary key of the "TIME" dimension; is required to query when the publication was published (time information about the last version in the arXiv dataset at the moment when data was added to the DWH)
doi	Digital Object Identifier (DOI) of the paper
title	the title of the publication
number_of_authors	number of authors
submitter	the name of the person who submitted the paper/corresponding author
lang	the language of the publication
volume	volume number; applicable when the paper is published in the journal[a]
issue	issue number; applicable when the paper is published in the journal[a]

page_numbers	publication page numbers in the journal or the other published scientific papers collection[a]
number_of_references	number of publications that present publication cites
no_ver_arXiv	number of versions of the current publication in the arXiv dataset; since the arXiv dataset is updated frequently, this field may change – a new version of the publication may be published
date_of_first_version	date when the first version (version v1 in arXiv) was created; is required for measuring the time interval between the first and current version of the publication
number_of_citations	number of publications that cite the present publication; this field may change over time
The following attributes are added for historical tracking.	
snapshot_valid_from	the date this row became effective
snapshot_valid_to	the date this row expired; is updated when a new row is added

[a]It is important to note that not all additional information fields are applicable in all cases. For example, workshop materials do not have an issue number. In these situations, the field will be filled as not-applicable.

One can notice that the timestamped accumulating snapshots concept[5,6] is used for historical data tracking. This approach is suitable because the data will change infrequently. For example, the number of citations of one publication may increase very often during some period, but at the same time, there may be long time intervals during which this value remains the same.

In the dimension table "AUTHORS", all the relevant data about the publications' authors (name and h-index) is stored. The difference between fields "h_index_real" and "h_index_calculated" is that the first h-index is retrieved by an API call and refers to the real-life h-index that the author has. The second h-index is calculated based on the data added to the DWH. (For more details, see the previous part of the "Final report".) The reason to keep both is that it is one way to immediately see if there is an error in the data pipeline – a calculated h-index could never be higher than a real-life one.

Since one author can have several publications and one publication can have several authors (many-to-many relationship), the bridge table[7] is used to connect the author's dimension with specific facts. Additionally, an author's h-index (both of them) is a variable that changes over time. For BI queries (for example, getting the author whose h-index increased the most during the last year), tracking that change is essential. Therefore, the type 2 slowly changing dimensions concept[8,9] is used – when the author's h-index changes, a new dimension record is generated. At the same time, the old record will be assigned a non-active effective date (valid_to =/= NULL), and the new record will be assigned an active effective date (valid_to == NULL).

In the dimension table "INSTITUTION", information about the institutions (name and location) of the authors of the publications will be gathered. Similarly to the authors' dimension, in this case, there could be a many-to-many relationship between the dimension and fact. In other words, there could be many publications from one institution,

and authors of the same publication can have different affiliations. Therefore, the bridge table is used to connect the dimension table records with the fact table records.

In this step, one may notice that there is no connection between the authors and the institutions. (For example, in the authors' dimension table, there is no information about the author's institution.) The reasoning behind this decision is that BI queries (see the folder `queries_dwh` in the project's GitHub page) do not require author-level information about affiliations. Based on the current schema, it is possible to fulfil all the queries requiring institutions' information.

Dimension table "PUBLICATION VENUES" stores data about the venues of the publications. In this table, there could be many fields that do not apply to all the records. For example, if the type is "book", the field `h_index_calculated` is irrelevant. However, if the field h-index is applicable (for journals), similarly to the "AUTHORS" dimension table, tracking its changes is essential from the BI point of view. Therefore, this table will also use the type 2 slowly changing dimensions concept.

In the dimension table "SCIENTIFIC DOMAINS", the categories of scientific disciplines of publications will be gathered. Again, the bridge table is used to overcome the shortcomings related to many-to-many relationships between publications and scientific domains (one publication can belong to many scientific domains, and many publications can have the same domain).

The "TIME" dimension holds all the relevant (from the BI point of view) time information about the publications. Besides the timestamp of the publication, it also has separate fields for year, month and day.

Implementation

To build the described DWH where the slowly changing dimensions (SCD) approach is supported while data is updated, the `load_dwh_db` Airflow DAG was generated (see figure 11).
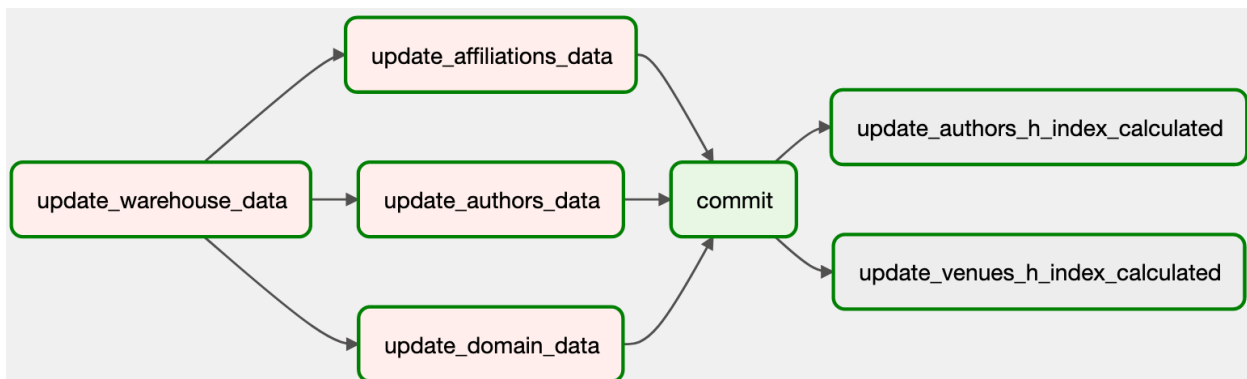

Figure 11 Airflow DAG `load_dwh_db`

This DAG consists of 6 "real" tasks and one connector task and guarantees that the data is saved to the database without duplicates with entity relations in 'UTF-8' format. All the DWH entities have a unique `id` that is generated via sequence. If the entity already stored in the database is tried to be inserted again, its `id` is returned.

The strategy for requesting the `id` of the entity combines upsert and select strategy at the same time:
* every insertion query has validation for the duplicate check;
* returning the `id` is inserted or saved for every entity

To run the `load_dwh_db` DAG, the files generated by `transform_articles` DAG mentioned in the previous part of the "Final Report" (CSV files in the `final_data` folder) as inputs are requested.
The first task of this pipeline, `update_warehouse_data`, reads the latest publication and venue files from the `final_data` directory. It saves and/or gets the `ids` of the venue entities and applies them row by row (venues are synchronised with DWH). Publications' information is saved using the same strategy. However, if necessary (for example, the number of citations has been changed), the data about publications that are already stored in the DWH is updated, and the timestamp `valid_to` is added. In the end, publications and their `ids` are saved as "`prepared_publication.csv`" in the folder `wh-data`. The other tasks use this file for getting the actual `publication_id` inside the DWH. Furthermore, the data from this task is used to calculate venue `h_index`.

The task `update_warehouse_data` is followed by three tasks that run in parallel:
* the `update_affilitions_data` task saves the affiliations and creates connections between affiliation and publication in the DWH;
* the `update_authors_data` task saves or updates the information related to the authors and creates connections between authors and publications; the information processed in this task is needed to calculate the `h_index` based on the data inside the warehouse;
* the `update_domain_data` task saves data related to the scientific domain and creates connections between the domain and publications.

In the last stage of the pipeline, the field `calculated_h_index` for venues and authors entities is updated.

PostreSQL
The Postgres with Citus extension was originally chosen as a data warehouse database (see "Design Document – GROUP 12"). However, the Official container for the Postgres extension does not support the ARM64 platform, and the emulator decreases the overall performance. Thus, this project used the official Postgres with indexes without the Citus extension.

Swifter

The Swifter[10] library was used in pair with the Pandas library for loading and transforming the datasets because it adds multicore usage for different Pandas DataFrame core transformation functions.
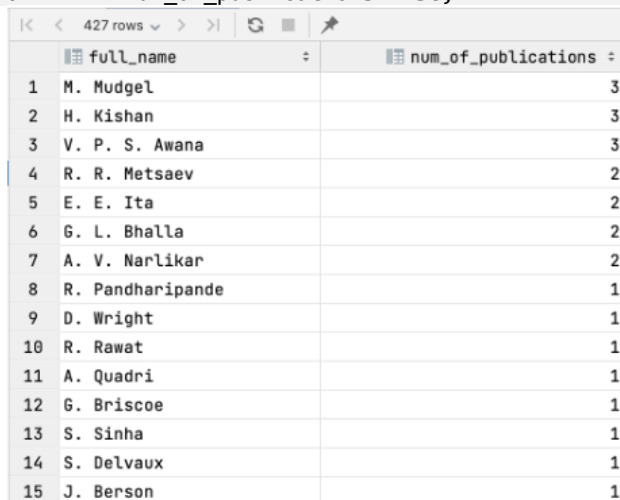
Queries

The DWH has been designed to answer many BI queries (see the folder `queries_dwh` on the project's GitHub page). This chapter gives two examples: one basic and one more challenging query, together with the results (figures 12 and 13).

Basic queries
Find authors with the most publications in a given year (2022)

```
-- Ranking authors with the most publications in a given year
SELECT
        aut.full_name,
        aut.first_name,
        COUNT(DISTINCT pub.doi) AS num_of_publications
FROM warehouse.publications pub
-- join authors with publications
JOIN warehouse.publication_author pub_auth
        ON pub_auth.publication_id = pub.id
JOIN warehouse.authors aut
        ON pub_auth.author_id = aut.id
-- join time with publications
JOIN warehouse.publication_time pub_time
        ON pub.time_id = pub_time.id
WHERE pub_time.year = '2012'
GROUP BY aut.id
ORDER BY num_of_publications DESC;
```

	full_name	num_of_publications
1	M. Mudgel	3
2	H. Kishan	3
3	V. P. S. Awana	3
4	R. R. Metsaev	2
5	E. E. Ita	2
6	G. L. Bhalla	2
7	A. V. Narlikar	2
8	R. Pandharipande	1
9	D. Wright	1
10	R. Rawat	1
11	A. Quadri	1
12	G. Briscoe	1
13	S. Sinha	1
14	S. Delvaux	1
15	J. Berson	1

Figure 12 Authors with the most publications in the year 2022

Challenging queries
Find the hottest topics (major field) in the year 2022

```
-- What are the year's hottest topics (categories of scientific disciplines)?
-- HOTTEST = most publications
-- by time
-- in major field
SELECT
        scientific_domain.major_field,
        COUNT(DISTINCT pub.pub_doi) AS num_of_publications
FROM
        -- use latest DOIs
        (
                SELECT
                        pu.id AS pub_id,
                        pu.time_id AS pub_time_id,
                        pu.number_of_citations AS pub_num_of_citations,
                        pu.venue_id AS pub_venue_id,
                        pu.title AS pub_title,
                        pu.doi AS pub_doi
                FROM warehouse.publications pu
                WHERE pu.snapshot_valid_to is NULL
                LIMIT 1
        ) pub
-- join scientific domains with publications
JOIN warehouse.publication_domain pub_domain
        ON pub_domain.publication_id = pub.pub_id
JOIN warehouse.scientific_domain scientific_domain
        ON pub_domain.domain_id = scientific_domain.id
-- join time with publications
JOIN warehouse.publication_time pub_time
        ON pub.pub_time_id = pub_time.id
WHERE
        pub_time.year = '2023'
GROUP BY scientific_domain.major_field
ORDER BY num_of_publications DESC;
```

	major_field	num_of_publications
1	natural sciences	11
2	engineering and technology	1

Figure 13 The hottest topic in the year 2022

Part 3. Graph database
Design

The labelled property graph model is used instead of RDF to design the graph database. It makes the graph look more concise and allows to specify properties next to nodes and edges. The initial design of the graph database, described in the "Design Document – GROUP 12", has not changed significantly. In the following list, all the changes are given:

- The relationship between the author and affiliation has been renamed from WORKS_AT to WORKS_IN
- WORKS_IN "data" attribute has been dropped to simplify the graph and reduce the team's workload. However, that information can still be retrieved by querying the author's publications with the year attribute.
- COVERED_BY relationship between ScientificDomain and Venue has been dropped. Venues for scientific domains can be retrieved by querying publications or affiliations that have direct relationships with venues.

The property graph diagram below (figure 14) shows the database entities and their relationships. The entities are represented as nodes; the relationships are represented as directed edges, and node properties are specified inside nodes. In Table 4, the same entities together with properties and in Table 5, the same relationships are also given.
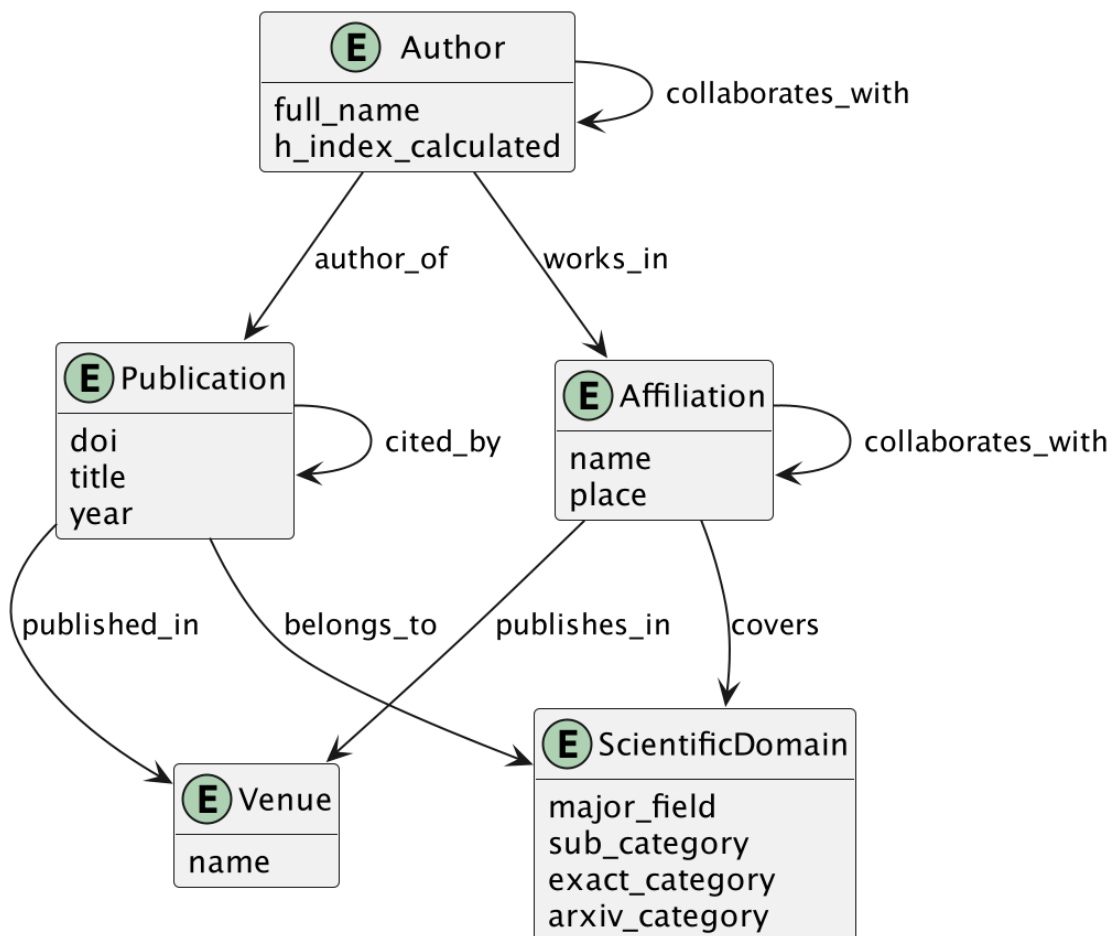

Figure 14 Schema of graph database

Table 4 Entities and their properties that are used in the graph database

Entity	Properties
Author	full_name, h_index_calculated
Affiliation	name, place
Publication	doi, title, year
ScientificDomain	major_field, sub_category, exact_category, arxiv_category
Venue	full_name

Table 5 Relationships between the entities in the graph DB

Relationship	Representation
AUTHOR_OF	(:Author)-[:AUTHOR_OF]->(:Publication)
COLLABORATES_WITH	(:Author)-[:COLLABORATES_WITH]->(:Author)
WORKS_IN	(:Author)-[:WORKS_IN}]->(:Affiliation)
PUBLISHED_IN	(:Publication)-[:PUBLISHED_IN]->(:Venue)
BELONGS_TO	(:Publication)-[:COVERS]->(:ScientificDomain)
CITED_BY	(:Publication)-[:CITED_BY]->(:Publication)]
COVERS	(:Affiliation)-[:COVERS]->(:ScientificDomain)
PUBLISHES_IN	(:Affiliation)-[:PUBLISHES_IN]->(:Venue)
COLLABORATES_WITH	(:Affiliation)-[:COLLABORATES_WITH]->(:Affiliation)

Implementation

After the initial transformation and data enrichment finishes, Airflow triggers the two DAGs:

- transform_for_graph_injection (figure 15)
- load_graph_db (figure 16)

The transform_for_graph_injection DAG prepares CSV files for the graph database injection. It determines the necessary relationships between entities and splits the data into the format required by the neo4j-admin import command.
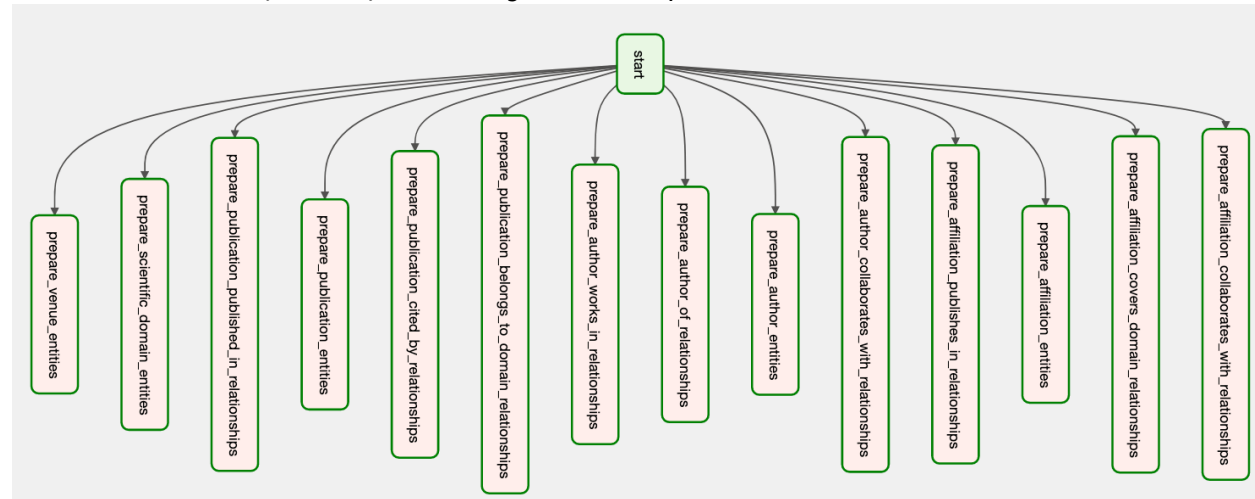

Figure 15 Airflow DAG transform_for_graph_injection

The load_graph_db DAG starts a container from the custom-built Docker image. First, the container runs the neo4j-admin import command to load the data into the graph database by overwriting the previously existing data. Then, it runs the neo4j command to

start the Neo4j server in `console` mode. The database is ready to be queried at
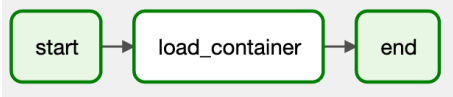http://localhost:7474.

Figure 16 Airflow DAG `load_graph_db`

Graph queries

The graph database has been designed to answer many questions about its entities and
the relationships between them. The questions and corresponding Cypher queries are
given in the table below.

Basic queries

This section describes basic queries that can be used to retrieve information about the
entities and their relationships.

Table 6 Basic queries (questions together with illustrative corresponding Cypher queries)

Question		Cypher Query
Getting an author who:	collaborates with a given author	MATCH (author1:Author)-[:COLLABORATES_WITH]->(author2:Author) WHERE author1.author_id = "224" RETURN author2 LIMIT 25
	collaborates with a given author in a given year	MATCH (author1:Author {author_id: "224"})-[:COLLABORATES_WITH]-(author2:Author)-[:AUTHOR_OF]-(p:Publication {year: 2007}) RETURN author2 LIMIT 25
	writes in a given scientific domain	MATCH (author:Author)-[:AUTHOR_OF]->(p:Publication)-[:BELONGS_TO]->(d:ScientificDomain) WHERE d.sub_category =~ "computer.*" RETURN author LIMIT 25
	writes in a given venue	MATCH (author:Author)-[:AUTHOR_OF]->(p:Publication)-[:PUBLISHED_IN]->(v:Venue) WHERE v.full_name = "Lecture Notes in Computer Science" RETURN author LIMIT 25
	writes for a given affiliation	MATCH (a:Author)-[:WORKS_IN]-(af:Affiliation) WHERE af.name = "Princeton University" RETURN a LIMIT 25
Getting a publication:	cited by a given publication	MATCH (p:Publication)-[:CITED_BY]->(:Publication {publication_id: "44324"}) RETURN p LIMIT 25
	cited by a given author	MATCH (p:Publication)-[:CITED_BY]->(:Publication)-[:AUTHOR_OF]-(a:Author {author_id: "6616"}) RETURN p LIMIT 25
	published in a given venue	MATCH (p:Publication)-[:PUBLISHED_IN]-(v:Venue {full_name: "Lecture Notes in Computer Science"}) RETURN p LIMIT 25
	affiliated with a given affiliation	MATCH (p:Publication)-[:AUTHOR_OF]-(a:Author)-[:WORKS_IN]-(af:Affiliation {name: "Princeton University"}) RETURN p LIMIT 25
	from a given scientific domain	MATCH (p:Publication)-[:BELONGS_TO]-(d:ScientificDomain) WHERE d.sub_category =~ "computer.*" RETURN p LIMIT 25
Getting an affiliation:	that covers a given scientific domain	MATCH (a:Affiliation)-[:COVERS]-(d:ScientificDomain) WHERE d.sub_category =~ "computer.*" RETURN a LIMIT 25
	publishes in a given publication venue	MATCH (a:Affiliation)-[:PUBLISHES_IN]-(v:Venue) WHERE v.full_name = "Physical Review D" RETURN a LIMIT 25
	employs a given	MATCH (a:Author)-[:WORKS_IN]->(af:Affiliation) WHERE a.full_name =

	author	"E. Bloomer" RETURN af LIMIT 25
Getting a scientific domain that is covered by a given:	affiliation	MATCH (:Affiliation {name: "Princeton University"})-[:COVERS]-(d:ScientificDomain) RETURN d LIMIT 25
	publication venue	MATCH (d:ScientificDomain)-[:COVERS]-(:Affiliation)-[:PUBLISHES_IN]-(v:Venue {full_name: "Physical Review D"}) RETURN d LIMIT 25
	author	MATCH (:Author {author_id: "224"})-[:AUTHOR_OF]-(:Publication)-[:BELONGS_TO]-(d:ScientificDomain) RETURN d LIMIT 25
Getting a publication venue that:	covers a given scientific domain	MATCH (:ScientificDomain {sub_category: "physical sciences"})-[:COVERS]-(:Affiliation)-[:PUBLISHES_IN]-(v:Venue) RETURN v LIMIT 25
	publishes for a given affiliation	MATCH (a:Affiliation)-[:PUBLISHES_IN]-(v:Venue) WHERE a.name = "Iowa State University" RETURN v LIMIT 25
	publishes for a given author	MATCH (a:Author {author_id: "224"})-[:AUTHOR_OF]-(:Publication)-[:PUBLISHED_IN]-(v:Venue) RETURN v LIMIT 25

Analytical queries

Besides the basic queries also, analytical queries were carried out.

Influential publications using PageRank
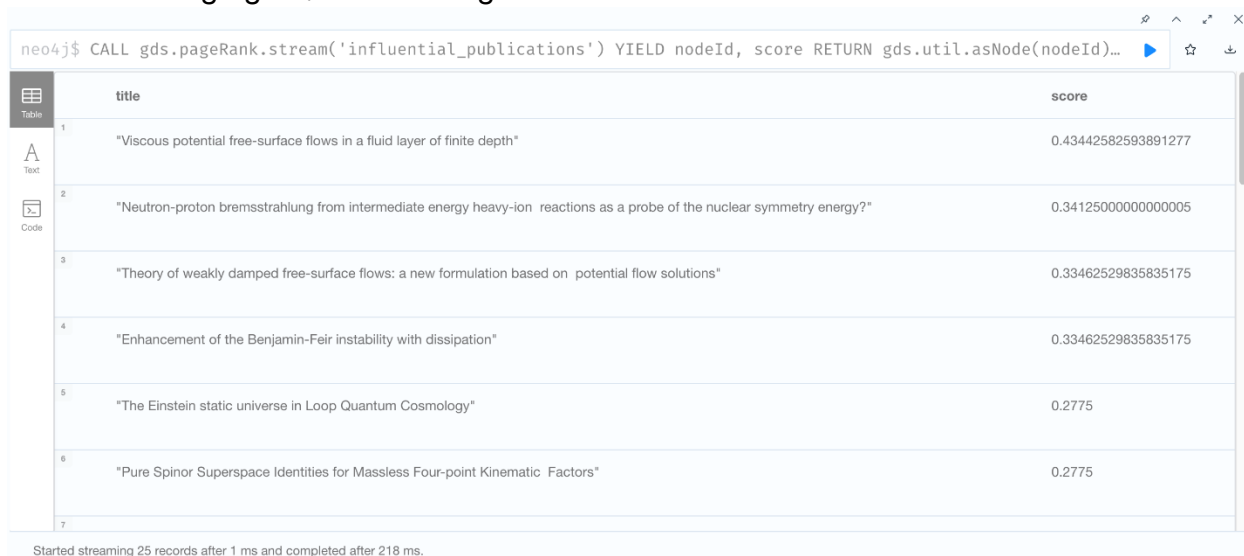
The Page Rank[11] algorithm was used to find the most influential publications. For that, firstly, the graph projection was created to use with the Graph Data Science Library v2.2 library[12]:

```
CALL gds.graph.project.cypher('influential_publications', 'MATCH (p:Publication) RETURN id(p) AS id', 'MATCH (p1:Publication)-[:CITED_BY]->(p2:Publication) RETURN id(p1) AS source, id(p2) AS target')
```

After that, the algorithm was run:

```
CALL gds.pageRank.stream('influential_publications')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).title AS title, score
ORDER BY score DESC
LIMIT 25
```

In the following figure, the resulting table is shown.

Figure 17 Influential publications using PageRank

Communities detection using Louvain

The Louvain[13] method from GDS with the following Cypher queries is used to find communities of authors that cover a particular scientific domain.
Firstly, the graph is projected:

```
CALL gds.graph.project.cypher('community_by_domain', 'MATCH (a:Author) RETURN id(a) AS id',
'MATCH    (a1:Author)-[:AUTHOR_OF]->(p:Publication)-[:BELONGS_TO]->(d:ScientificDomain)    WHERE
d.sub_category =~ "computer.*" MATCH (a2:Author)-[:AUTHOR_OF]->(p) WHERE a1 <> a2 RETURN
id(a1) AS source, id(a2) AS target')
```
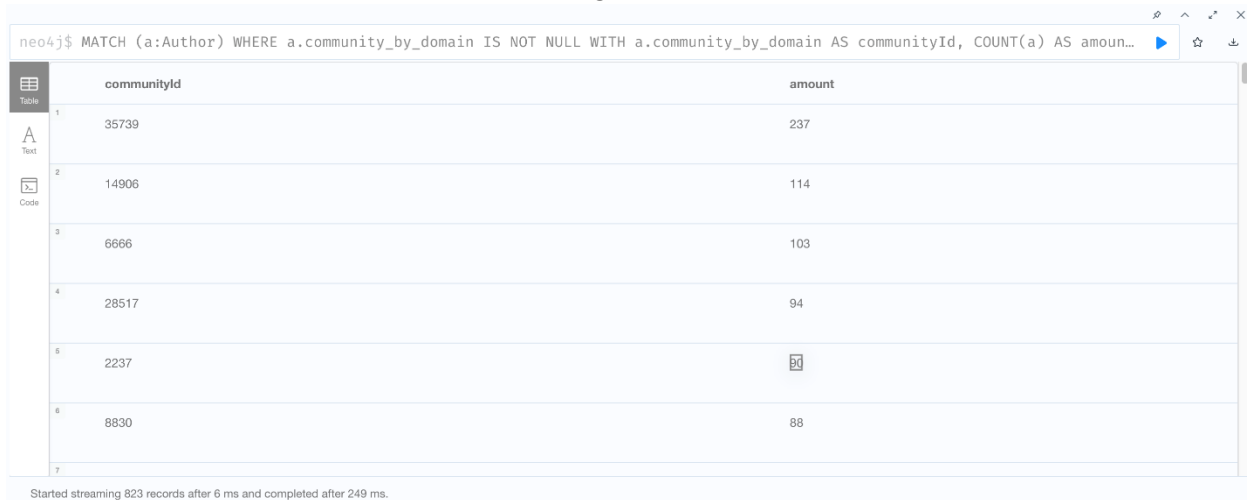
Then, it is possible to write the `community_by_domain` ID to the authors' nodes as a property:

```
CALL gds.louvain.stream('community_by_domain')
YIELD nodeId, communityId
WITH gds.util.asNode(nodeId) AS a, communityId AS communityId SET a.community_by_domain =
communityId
```

After that, the communities where the amount of authors is greater than 1 are queried:

```
MATCH (a:Author)
WHERE a.community_by_domain IS NOT NULL
WITH a.community_by_domain AS communityId, COUNT(a) AS amount WHERE amount > 1
RETURN communityId, amount
ORDER BY amount DESC
```

The results of this query are illustrated in figure 18.

Figure 18 Communities that have more than one authors

Finally, it is possible to filter out the biggest community and display it (see figure 19) with the query:

```
MATCH (a:Author {community_by_domain: 35739}) RETURN a LIMIT 25
```
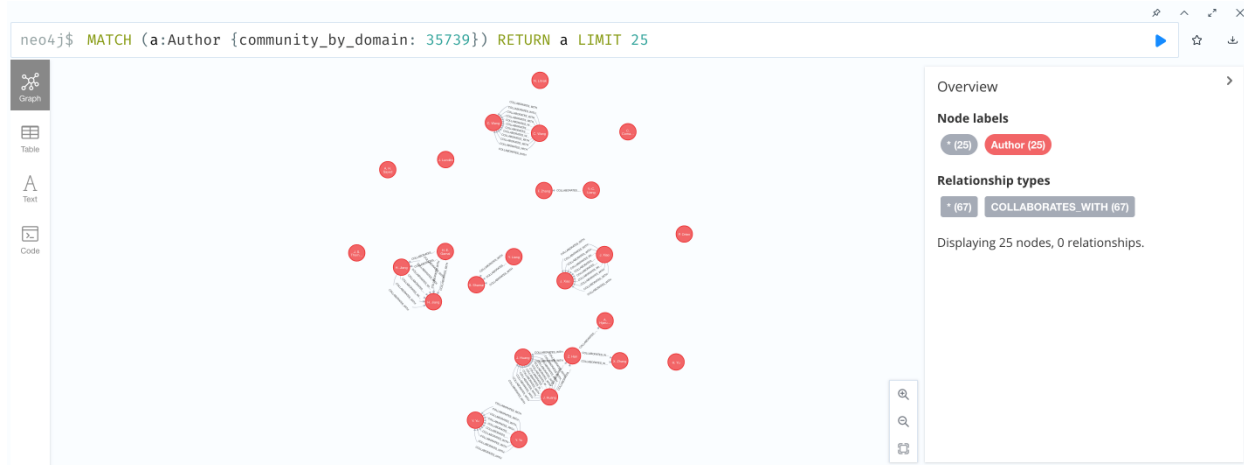
Figure 19 The biggest community in graph DB retrieved by using the Louvain method

Missing links between authors using Delta-Stepping Single-Source Shortest Path

The Single-Source Shortest Path[14] from GDS is used to search for a missing link between two authors.

In the first step, the projection is made:

```
CALL gds.graph.project.cypher('missing_link', 'MATCH (a:Author) RETURN id(a) AS id', 'MATCH (a1:Author)-[:COLLABORATES_WITH]-(a2:Author) RETURN id(a1) AS source, id(a2) AS target')
```

Then, two authors who have not collaborated with each other, *e.g.*, "T. Nagao" and "T.H. Puzia", are picked out (see figure 20).
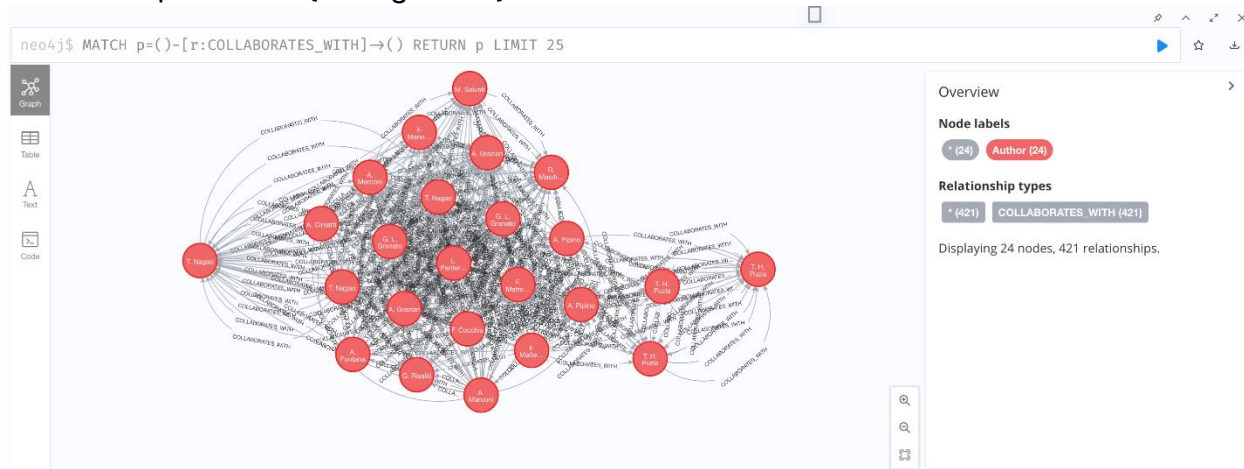

Figure 20 Authors nodes with two authors, "T. Nagao" and "T.H. Puzia", without direct `COLLABORATE_WITH` relationship

Finally, the shortest path between the authors with `author_id` 36102 and 34512 is found:

```
MATCH (source:Author {author_id: "36102"})
CALL gds.allShortestPaths.delta.stream('missing_link_2', {sourceNode: source})
YIELD index, sourceNode, targetNode, path
WHERE gds.util.asNode(targetNode).author_id = "34512"
RETURN      index,       gds.util.asNode(sourceNode).full_name      AS      sourceNodeName,
gds.util.asNode(targetNode).full_name AS targetNodeName, nodes(path) as path
ORDER BY index
LIMIT 25
```

```
1  MATCH (source:Author {author_id: "36102"})
2  CALL gds.allShortestPaths.delta.stream('missing_link_2', {sourceNode: source})
3  YIELD index, sourceNode, targetNode, path
4  WHERE gds.util.asNode(targetNode).author_id = "34512"
5  RETURN index, gds.util.asNode(sourceNode).full_name AS sourceNodeName, gds.util.asNode(targetNode).full_name AS
   targetNodeName, nodes(path) as path
6  ORDER BY index
7  LIMIT 25
```
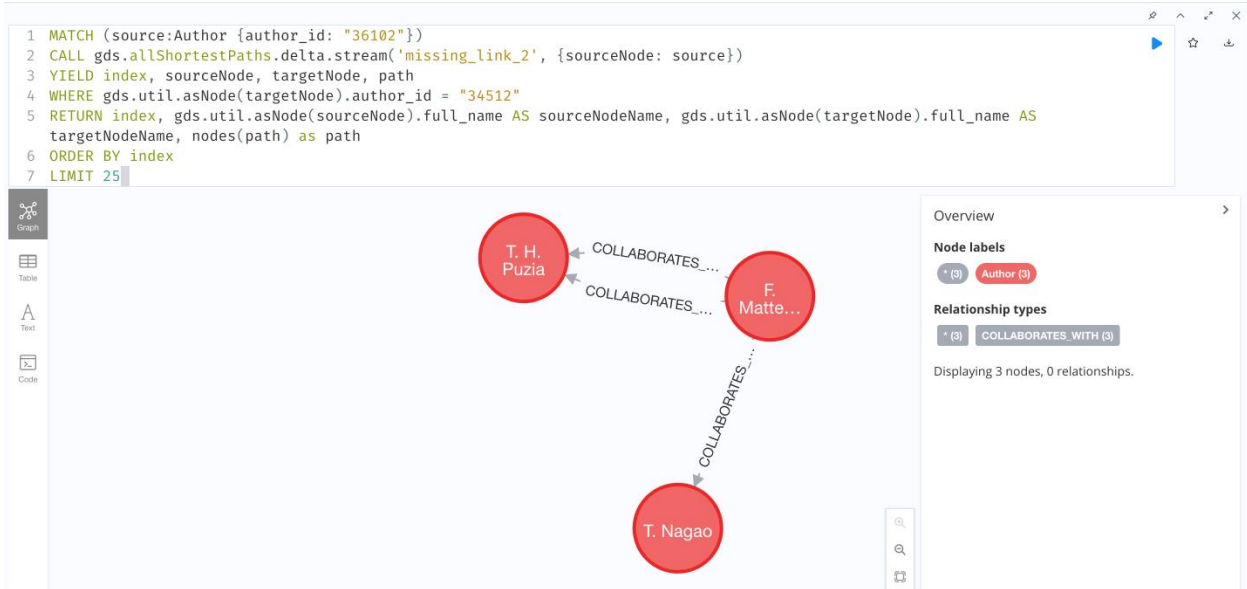
Figure 21 The missing link between the two authors, "T. Nagao" and "T.H. Puzia", received by using the Delta-Stepping Single-Source Shortest Path algorithm
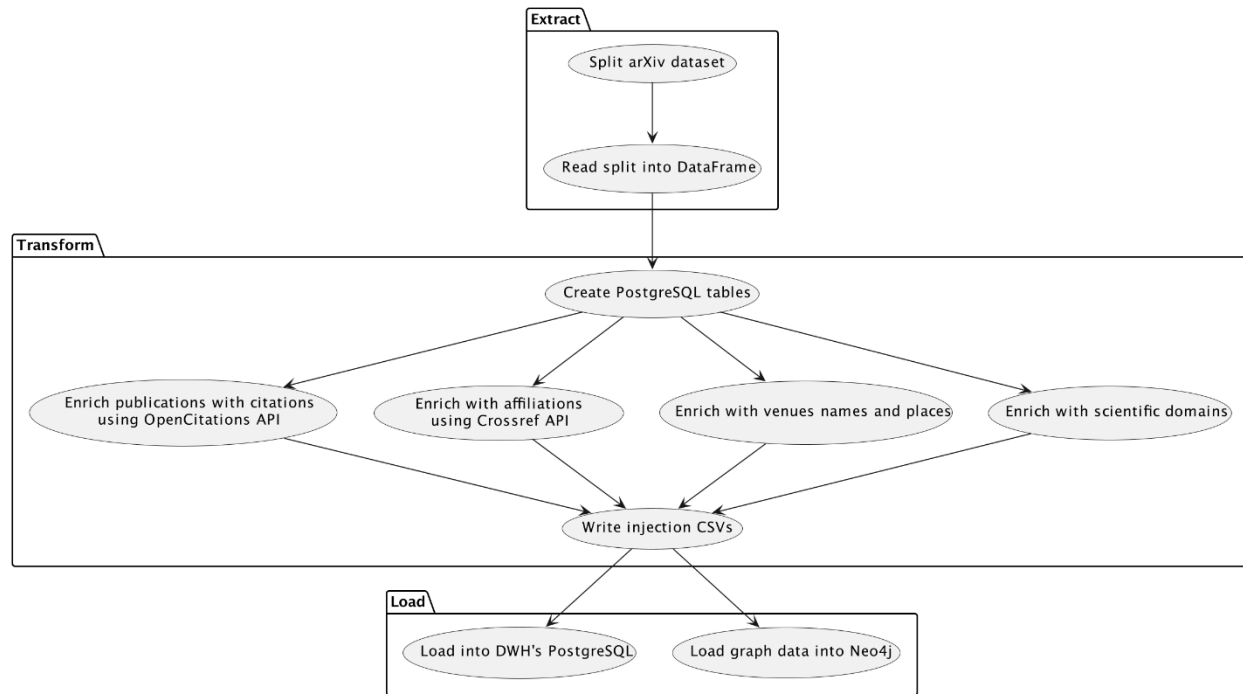
Guidelines for running the overall pipeline

Figure 22 Schema of the overall pipeline

Extract

The arXiv dataset is available at https://www.kaggle.com/datasets/Cornell-University/arxiv.

To simulate periodic updates, the raw dataset is split. The following command splits the dataset into splits of 50k publications and saves them to `./airflow/data/inputs`:

```
$ python split_dataset.py --arxiv_path <path-to-arxiv-dataset-json> --output_dir ./airflow/data/inputs
```

Transform

Prerequisites

1. [manual task] Airflow connection to PostgreSQL database:

```
connection_id: airflow_pg
conn_type: Postgres
host: postgres
login: airflow
password: airflow
port: 5432
```

2. [builds with docker-compose] Custom-built Airflow Docker. It should be built with `docker-compose`. Manual build can be done with the following commands:

```
$ cd docker_context_airflow
$ docker build -t airflow-custom .
```

Run (in Airflow's UI)

Run the following DAGs in the following order:

1. `transform_create_tables`
2. `transform_articles`

Load
Data warehouse
Prerequisites

1. [manual task] Airflow connection to PostgreSQL database:

```
connection_id: citus_warehouse
conn_type: Postgres
host: citus-db
schema: warehouse
login: citususer
password: cituspass
port: 5432
```

Run (in Airflow's UI)

After the transformation step, run:

1. load_dwh_db
2. Connect to the database that was mapped to the `localhost:25432` using the following connection string:

```
postgresql://citususer:cituspass@localhost:25432/warehouse
```

The schema warehouse should be added manually if tables aren't shown in your database client.

Neo4j Graph Database
Prerequisites

1. [manual task] Make sure `neo4j-script` custom Docker image is present:

```
$ cd ./docker_context_neo4j
$ docker build -t neo4j-script -f neo4j-script.dockerfile .
```

2. [comes with repo] Make sure neo4j plugins are located at ./neo4j/plugins
    i.   APOC
    ii.   Graph Data Science Library
3. [builds with docker-compose] Note that `docker-proxy` container exposes the host's `/var/run/docker.sock` to containers via TCP (don't know where `docker.sock` is located on Windows machines, the following StackOverflow thread might help (https://stackoverflow.com/questions/36765138/bind-to-docker-socket-on-windows))

Run (in Airflow's UI)

After the transformation step, run:

1. Run the `transform_for_graph_injection` DAG. It will create import CSVs at ./neo4j/import.
2. Run the `load_graph_db` DAG. This will start a new container with ports 7474 and 7687 exposed. The container might take some seconds to start.
3. Go to http://localhost:7474/browser/ and run the following Cypher query to check that the graph was loaded correctly:

```
MATCH (n) RETURN n LIMIT 25
```

4. When finished, stop the DAG by marking its `load_container` task with the "Mark Success" button in Airflow's UI. This will remove the container. Graph database files are preserved and located at ./neo4j/data.

References

(1) Bartell, A. *REST API*. Crossref. https://www.crossref.org/documentation/retrieve-metadata/rest-api/ (accessed 2022-11-07).

(2) Peroni, S.; Shotton, D. Open Citation: Definition. **2018**, 95436 Bytes. https://doi.org/10.6084/M9.FIGSHARE.6683855.

(3) Kannawadi, S. A. C., Panos Ipeirotis, Victor Silva, Arun. Scholarly: Simple Access to Google Scholar Authors and Citations. https://github.com/scholarly-python-package/scholarly (accessed 2022-11-07).

(4) *Scientific Disciplines - EGI Glossary - EGI Confluence*. https://confluence.egi.eu/display/EGIG/Scientific+Disciplines (accessed 2022-11-07).

(5) Mundy, J. *Design Tip #145 Timespan Accumulating Snapshot Fact Tables*. Kimball Group. https://www.kimballgroup.com/2012/05/design-tip-145-time-stamping-accumulating-snapshot-fact-tables/ (accessed 2022-11-07).

(6) *Timespan Tracking in Fact Tables | Kimball Dimensional Modeling Techniques*. Kimball Group. https://www.kimballgroup.com/data-warehouse-business-intelligence-resources/kimball-techniques/dimensional-modeling-techniques/timespan-fact-table/ (accessed 2022-11-07).

(7) Thornthwaite, W. *Design Tip #142 Building Bridges*. Kimball Group. https://www.kimballgroup.com/2012/02/design-tip-142-building-bridges/ (accessed 2022-11-07).

(8) Kimball, R. *Slowly Changing Dimensions*. Kimball Group. https://www.kimballgroup.com/2008/08/slowly-changing-dimensions/ (accessed 2022-11-07).

(9) Kimball, R. *Slowly Changing Dimensions, Part 2*. Kimball Group. https://www.kimballgroup.com/2008/09/slowly-changing-dimensions-part-2/ (accessed 2022-11-07).

(10) Carpenter, J. Swifter, 2023. https://github.com/jmcarpenter2/swifter (accessed 2023-01-15).

(11) *PageRank - Neo4j Graph Data Science*. Neo4j Graph Data Platform. https://neo4j.com/docs/graph-data-science/2.2/algorithms/page-rank/ (accessed 2023-01-15).

(12) *The Neo4j Graph Data Science Library Manual v2.2 - Neo4j Graph Data Science*. Neo4j Graph Data Platform. https://neo4j.com/docs/graph-data-science/2.2/ (accessed 2023-01-15).

(13) *Louvain - Neo4j Graph Data Science*. Neo4j Graph Data Platform. https://neo4j.com/docs/graph-data-science/2.2/algorithms/louvain/ (accessed 2022-11-07).

(14) *Delta-Stepping Single-Source Shortest Path - Neo4j Graph Data Science*. Neo4j Graph Data Platform. https://neo4j.com/docs/graph-data-science/2.2/algorithms/delta-single-source/ (accessed 2023-01-15).