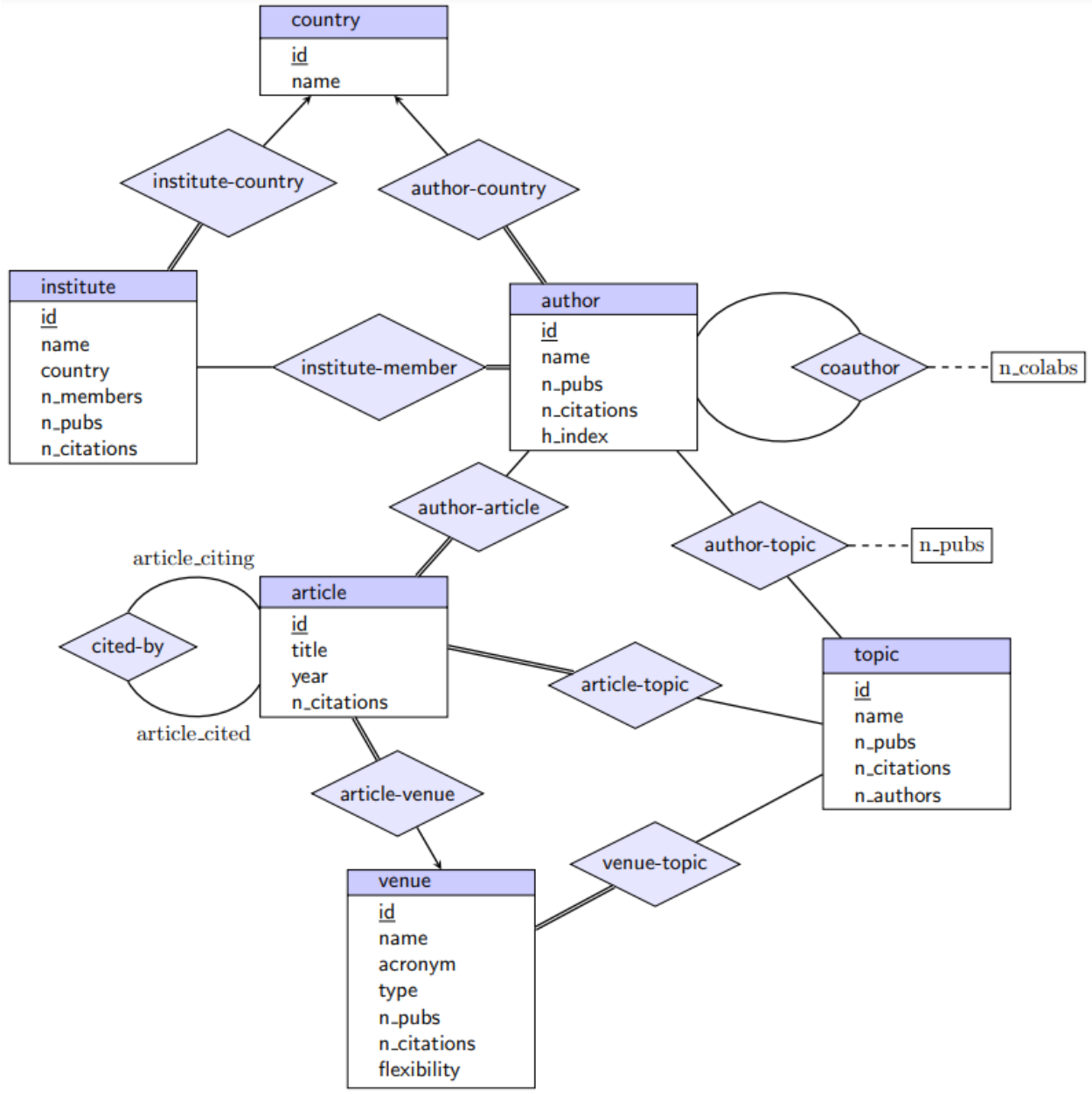
# CS387 Project: Design Document

## Team

| Name | Roll Number |
|---|---|
| Adarsh Kumar | 19D180003 |
| Adithya Bhaskar | 190050005 |
| Devansh Jain | 190100044 |
| Harshit Varma | 190100055 |

# ER Diagram


country
id
name
institute-country
author-country
institute
id
name
country
n_members
n_pubs
n_citations
institute-member
author
id
name
n_pubs
n_citations
h_index
coauthor
n_colabs
author-article
author-topic
n_pubs
article_citing
cited-by
article_cited
article
id
title
year
n_citations
article-topic
topic
id
name
n_pubs
n_citations
n_authors
article-venue
venue-topic
venue
id
name
acronym
type
n_pubs
n_citations
flexibility

# Normalized Logical Schema


This schema is used for the CSV files, which are in turn used for loading the data into the Neo4j database. Some relations (like author_country, article_venue) can be reduced to attributes, but are kept as relations since queries are more efficient that way in Neo4j (reference: https://neo4j.com/blog/dark-side-neo4j-worst-practices/)

# Integrity Constraints

- Primary and Foriegn key constraints are as per the schema given above
- Other constraints are given by the ER diagram
- No attribute is allowed to be null in any table

# Materialized Views

Materialized views are not needed.

## DB Creation and Inserting Data

Prior to loading the data in the Neo4j DB, we store it in CSV files according to the schema. Then a python script (similar to the one in Lab-6) is used to create cypher queries to insert the data into the Neo4j DB. The code for this is ready and has been tested.

## User Interface and the Business Logic Controller

- `(/home)`
  Contains tabs for `/institutes, /authors, /venues, /topics`

- `(/institutes)`
  Shows a table with columns corresponding to the institute entity's attributes
  Institute names will be clickable and will redirect to `/institutes/<institute_id>`
  Rows will be sorted according to a default metric (citations per member)
  User will be able to:
  - sort by any of the defined metrics (dropdown)
  - filter by the country (dropdown)
  - filter by topics (dropdown): only articles having these topics will be used for re-computing the metrics
  - search by name (text input)
  - specify a time range (a slider with 2 movable ends): metrics will be re-computed after this using only the articles which were published in this time range

- `(/institutes/<institute_id>)`
  Show the institute details (name, country, metrics)
  Show a table with all the members of the institute, member names redirect to `/authors/<author_id>`
  Charts:
  - bar graph: #publications vs year
  - bar graph: #citations vs year

- `(/authors)`
  Shows a table with columns corresponding to the author entity's attributes
  Author names will be clickable and will redirect to `/authors/<author_id>`
  Rows will be sorted according to a default metric (total citations)
  User will be able to:
  - sort by any of the defined metrics (dropdown)
  - filter by the country (dropdown)
  - search by name (text input)
  - filter by topics (dropdown): only articles having these these topics will be used for re-computing the metrics
  - specify a time range (a slider with 2 movable ends), metrics will be re-computed after this using only the articles which were published in this time range

- `(/authors/<author_id>)`
  Show author's details (name, country, institutes), metrics, top-5 topics, and a table containing publications (name, year, citations) authored by the author.
  Charts:
    - bar graph: publications vs year
    - bar graph: citations vs year
    - pie chart: number of publications vs topics (top-5 only)
    - line graph: h-index vs year
    - co-authorship graph for the author (nodes are authors, weighted undirected edge if they've collaborated, with weight = no. of co-authored articles)

- `(/articles/<article_id>)`
  Shows article entity's attributes along with the article's venue, topics and metrics.
  Charts:
    - bar graph: #citations vs year
    - citation graph for the article (nodes are articles, directed edge (v1, v2) if v1 cites v2)

- `(/venues)`
  Shows a table with columns corresponding to the venue entity's attributes and the list of topics for that venue
  Rows will be sorted according to a default metric (average citations)
  Venue names will be clickable and will redirect to `/venues/<venue_id>`
  User will be able to:
    - sort by any of the defined metrics (dropdown)
    - filter by the type (dropdown)
    - filter by topics (dropdown)
    - search by name (text input)

- `(/venues/<venue_id>)`
  Show venue details (name, acronym, type, topics, metrics)
  Charts:
    - bar graph: #publications vs year

- `(/topics)`
  Shows table with topic names and their metrics
  Rows will be sorted according to a default metric (average citations)
  Topic names will be clickable and will redirect to `/topics/<topic_id>`
  User will be able to:
    - sort by any of the defined metrics (dropdown)

- `(/topics/<topic_id>)`
  Shows topic name and metrics
  Charts (can be used to visualize a topic's popularity over time):
  - bar graph: #publications with this topic vs year
  - bar graph: #citations of publications with this topic vs year

# Transaction Descriptions and Queries

## Cypher

- Institute
  - Fetch all institute's data
    ```
    MATCH (i : Institute)
    RETURN i;
    ```
  - Fetch a particular institute's data (including all members)
    ```
    MATCH (i:Institute{id : $1})-[:InstituteMember]->(j)
    RETURN i,j.name;
    ```
  - Fetch (n_pubs, year) data for a particular institute
    ```
    MATCH (i : Institute{id: $1})-[:InstituteMember]->(j :
    Author)-[:AuthorArticle]->(k: Article)
    RETURN  i AS institute, COUNT(k.id) AS n_pubs, k.year as year;
    ```
  - Fetch (n_citations, year) data for a particular institute
    ```
    MATCH (i : Institute{id: $1})-[:InstituteMember]->(j :
    Author)-[:AuthorArticle]->(k: Article) RETURN
    i AS institute, SUM(k.n_citations) AS n_citations, k.year as year;
    ```

- Author
  - Fetch all authors data
    ```
    MATCH (i : Author)
    RETURN i;
    ```
  - Fetch a particular author's data (including all articles authored)
    ```
    MATCH (i:Author{id : $1})-[:AuthorArticle]->(j)
    RETURN i,j;
    ```
  - Fetch (n_pubs, year) data for a particular author
    ```
    MATCH (i : Author{id: $1})-[:AuthorArticle]->(j : Article)
    RETURN i AS author, count(j.id) AS n_pubs, j.year as year;
    ```
  - Fetch (n_citations, year) data for a particular author
    ```
    MATCH(j : Author{id: $1})-[:AuthorArticle]->(k: Article)
    RETURN j AS author, SUM(k.n_citations) AS n_citations, k.year as year;
    ```
  - Fetch (n_pubs, topics) data for a particular author
    ```
    MATCH(j : Author{id: $1})-[:AuthorArticle]->(k:
    Article)-[:ArticleTopic]->(i:Topic)
    ```

```
RETURN j AS author, COUNT(k.id) AS n_pubs, i AS topic
```

- Fetch (coauthor, n_colabs) for a particular author

```
MATCH (j : Author{id : $1})-[k:Coauthor]->(i)
RETURN j, i, k.n_colab;
```

- Article
  - Fetch a particular article's data

```
MATCH (i : Article{id : $1})
RETURN i;
```

  - Fetch (n_citations, year) data for a particular article

```
MATCH (i : Article{id : $1})-[:CitedBy]->(j)
RETURN i, COUNT(j) as n_citations, j.year AS Year;
```

  - Fetch all articles that have cited the article

```
MATCH (i : Article{id : $1})-[:CitedBy]->(j)
RETURN j;
```

- Venue
  - Fetch all venues data

```
MATCH (i : Venue)
RETURN i;
```

  - Fetch a particular venue's data

```
MATCH (i : Venue{id : $1})
RETURN i;
```

  - Fetch (n_pubs, year) data for a particular venue

```
MATCH (i : Article{venue_id: $1})
RETURN count(i.id) AS n_pubs, i.year as year;
```

  - Fetch (n_citations, year) data for a particular venue

```
MATCH (i : Article{venue_id: $1})
RETURN sum(i.n_citations) AS n_citations, i.year as year;
```

- Topic
  - Fetch all topics data

```
MATCH (i : Topic)
RETURN i;
```

  - Fetch a particular topic's data

```
MATCH (i : Topic{id : $1})
RETURN i;
```

  - Fetch (n_pubs, year) data for a particular topic

```
MATCH (i : Topic{id: $1})<-[:ArticleTopic]-(j : Article)
RETURN i AS topic, count(j.id) AS n_pubs,  j.year as year;
```

  - Fetch (n_citations, year) data for a particular topic

```
MATCH (i : Topic{id: $1})<-[:ArticleTopic]-(j : Article)
RETURN i AS topic, sum(j.n_citations) AS n_citations, j.year as year;
```

### Spark

Spark is used for computing the various metrics and filling up some of the CSV files after extracting some basic data.

- n_citations for articles
- n_citations, n_pubs, h_index for authors
- n_pubs, n_members, n_citations for institutes
- n_articles, n_authors, n_citations for topics
- n_pubs, n_citations, flexibility for venues
- Topics an author publishes in with number of publications per topic, extracted from the author's articles (this is used to fill up article_topic.csv)

The PySpark code for the above points is ready.

## Indexes for Optimization

All our queries require operations that use id's. Thus, we can have single-property indices over each node id's. We can also have a secondary index over article.year as it's also involved in queries.
Reference: https://neo4j.com/docs/cypher-manual/current/indexes-for-search-performance/

## Technology

- Neo4j database
- Node.js backend
- Angular frontend
- PySpark
- FastAPI