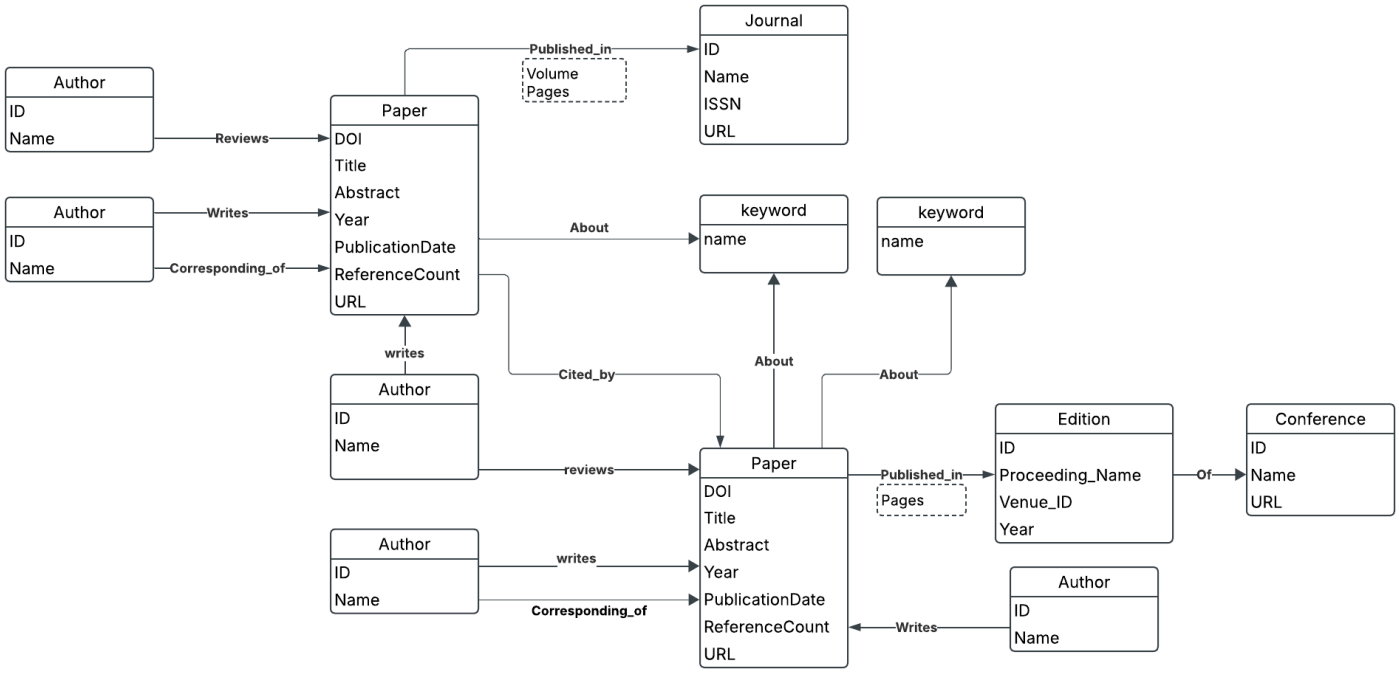
A.1 Modeling



Design Justifications:

- Papers, authors, journals, and conferences are created as nodes as they are the central concepts and it is very logical to represent them this way.
- Edition and conference nodes represent both conferences and workshops as both have the exact properties.
- Considering the queries in other parts,
    - Volume is set as a property of the ‘published_in’ relationship instead of a separate node as we don’t have queries using it.
    - Keywords and editions of conferences/workshops are set as separate nodes to be more convenient for the queries.
- We have 3 reviewers per paper, but they are not all shown in the diagram for simplicity.

A.2 Instantiating/Loading

The Semantic Scholar API (through sch.search_paper() bulk search function of the semanticscholar python library) was used to generate the data. The data retrieved is for papers (jsons), providing the DOI, abstract, publication venue details, authors, url, fields of study, and the IDs for the referenced papers. We then loop on these papers and for each we parse the needed details for our graph. A csv for the paper nodes is created. The authors, venues, fields of study details are collected in the loop and transformed to csvs at the end.
For the referenced papers, we loop on collected referenced IDs and retrieve them using sch.get_paper(doi) and also process them the same way we did for the citing papers to get the

details we need for the nodes/relationships. Then, we combine the csvs of the citing papers and referenced papers to form the final csvs to be used in graph loading.

Synthetic data generation

Due to limitations of data available in the Python SemanticScholar library, we generated data for Conference Editions and Journal references so we could answer the queries in part B.

For Conference Editions, the name of a conference was used to create 4 conference editions going back 4 years. A paper was then linked to a conference edition node which was in turn linked to a conference.

A conference edition node has the following properties:

```
Conf_editions.append({
     "Edition_ID": f"{edition_year}{venue_id}",
     "Venue_ID": f"{venue_id}",
     "Conference_Edition_Name": f"{edition_year} {venue}",
     "Year": edition_year
 })
```

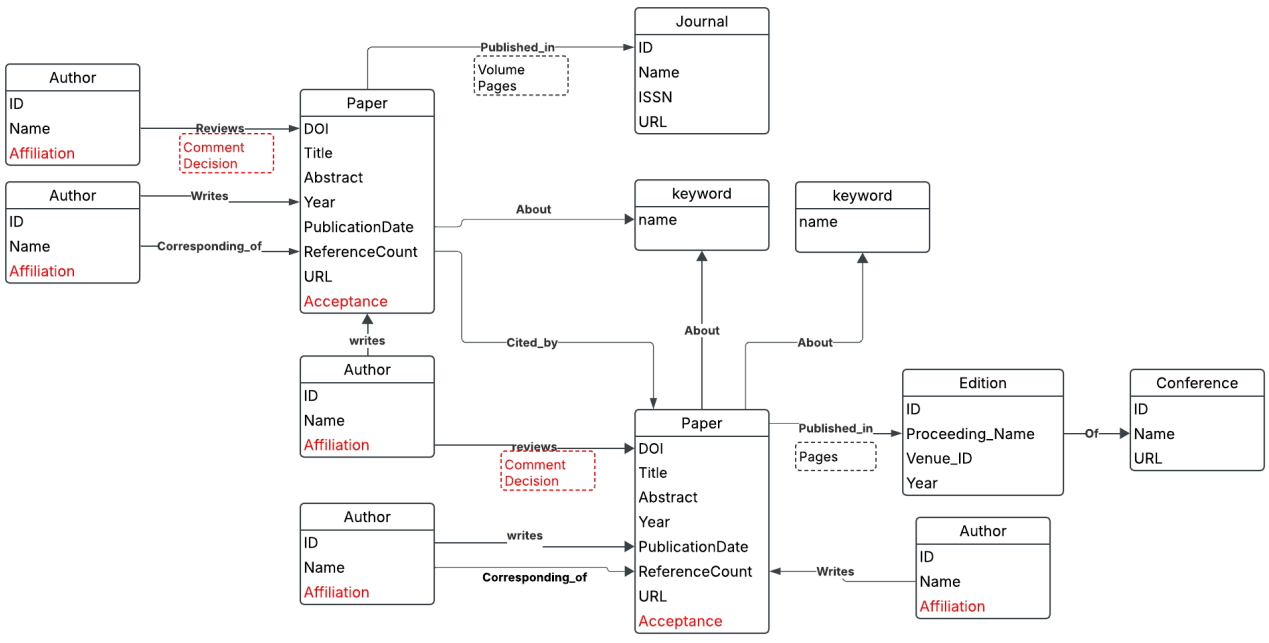
For journal references, data was generated for 2 random journals where papers not published in the journal are used to refer to papers published in these journals. This was done to ensure that journal impact factor can be calculated in part B.

For reviews, we select 3 random authors as reviewers for each paper with a condition that none of them is an author of the paper.

Data loading

From the neo4j desktop app, we created a new local db with these details: Name: SDM_Lab1_Research, Username: neo4j, Password: pass1234. Then we used python to run the loading cypher queries on our db filling all the nodes and relationships.

A.3 Evolving the graph



For Affiliations, when retrieving the authors details using the semantic scholar api, they usually don't provide the affiliation. Therefore, we will first generate synthetic affiliations for the authors and then set them as properties for the author nodes.

For reviews, we generate synthetic reviews text and decision for individual reviews, and regarding the graph updates, we add a boolean property to the review relationship that represents the reviewer's decision of acceptance or rejection. The final decision of the paper is set as a boolean property of the paper node according to the majority (In our graph we always have 3 reviewers per paper, so there will be no ties). Since all the papers we have are already published, it's not logical to have a rejection decision assigned to them. However, we allowed it just for the sake of showing applicability.

B Querying

All the queries have been included in Cyphers B.ipynb file in the project folder.

1. **Find the top 3 most cited papers of each conference/workshop.**

   Here, we find the number of times a paper was cited and match the paper to its conference through the conference edition. We then order the papers for each conference by the citation count in descending order and get the top 3 papers.

```
MATCH (p:Paper)-[:Cited_by]→(citing:Paper)
       WITH p, count(citing) AS citation_count
       MATCH (p)-[:Published_in]→(ce:Conference_Edition)-[:Of]→(conf:Conference)
       WITH conf, p.title AS paper_name, citation_count
       ORDER BY conf, citation_count DESC
       WITH conf.Name AS conference, collect({paper: paper_name, citations: citation_count})[0..3] AS top_papers
       RETURN conference, top_papers
```

2. **For each conference/workshop find its community: i.e., those authors that have published papers on that conference/workshop in, at least, 4 different editions.**

   We find the authors that have published a paper in a conference and count the authors that have published on 4 distinct editions.

```
MATCH (a:Author)-[:Writes]→(p:Paper)-[:Published_in]→(ce:Conference_Edition)-[:Of]→(conf:Conference)
       WITH conf, a, count(DISTINCT ce) AS editions_count
       WHERE editions_count ⩾ 4
       RETURN conf.Name AS conference, collect(DISTINCT a.Name) AS community_authors
       ORDER BY conference
```

3. **Find the impact factor of the journals in your graph.**

   Impact factor of journals was calculated using the formula shown here for the y=2022 where y is the year.

$$\mathrm{IF}_y = \frac{\mathrm{Citations}_y}{\mathrm{Publications}_{y-1} + \mathrm{Publications}_{y-2}}.$$

```
WITH $year AS y
      // Step 1: Get citations in year y for papers published in (y-1) or (y-2)
      MATCH (citing:Paper)-[:Cited_by]→(p:Paper)-[:Published_in]→(j:Journal)
      WHERE p.year IN [y-1, y-2] AND citing.year = y
      WITH j, count(citing) AS total_citations, $year as y
      // Step 2: Count total papers published in (y-1) and (y-2)
      MATCH (p:Paper)-[:Published_in]→(j)
      WHERE p.year IN [y-1, y-2]
      WITH j, total_citations, count(p) AS total_papers_published
      WHERE total_papers_published > 0
      // Step 3: Compute impact factor
      RETURN j.ID, j.Name AS journal, total_citations, total_papers_published,
      total_citations * 1.0 / total_papers_published AS impact_factor
      ORDER BY impact_factor DESC
```

4. **Find the h-index of the authors in your graph.**

Citations for papers published by an author are counted and stored in descending order in a list for each author. We check if the citation count for the hth paper in the list of size h is at least h.

```
MATCH (a:Author)-[:Writes]→(p:Paper)
      OPTIONAL MATCH (p)-[:Cited_by]→(citing:Paper)
      WITH a, p, count(citing) AS citation_count
      ORDER BY a.Name, citation_count DESC // Sort papers by citations in descending order

      WITH a.Name AS author, collect(citation_count) AS citations_list
      UNWIND range(1, size(citations_list)) AS h // Generate index positions (h)
      WITH author, h
      WHERE citations_list[h-1] ⩾ h // Check if the h-th paper has at least h citations

      RETURN author, max(h) AS h_index
      ORDER BY h_index DESC
```

C Recommender

1. **The first thing to do is to find/define the research communities. A community is defined by a set of keywords.**

   Instead of creating a database community, we used the keywords data we stored in our graph as nodes to find a science community using specific keywords. We create a COMMUNITY node for the Science Community and link it to keywords nodes with :INCLUDES relationship.

   ```
   MATCH (k:Keyword)
         WHERE k.Name IN [
           'Physics','Chemistry','Biology',
           'Computer Science','Political Science','Agricultural and Food Sciences'
         ]
         WITH collect(k) AS Keywords
         MERGE (c:Community {name: "Science Community"})
         WITH c, Keywords
         UNWIND Keywords AS keyword
         MERGE (c)-[:INCLUDES]→(keyword)
         RETURN c.name AS Community, count(Keywords) AS KeywordCount
   ```

2. **Next, we need to find the conferences, workshops and journals related to the database community (i.e., those that are specific to the field of databases). Assume that if 90% of the papers published in a conference/workshop/journal contain one of the keywords of the database community we consider that conference/workshop/journal as related to that community.**

   We find the papers that are linked to the keywords in the science community. We then find the conferences/journals of these papers. We count the number of papers that share this keyword and the total number of papers for each conference/journal. If 90% of the papers in a conference/journal are linked to keywords of the science community, we link the conferences/journals to the COMMUNITY node for the science community using the :RELATED_TO relationship.

```
MATCH (c:Community {name: "Science Community"})-[:INCLUDES]→(k:Keyword)←[:About]-(p:Paper)
      WITH c, collect(DISTINCT p) AS communityPapers

      // For conferences
      MATCH (conf:Conference)←[:Of]-(edition:Conference_Edition)←[:Published_in]-(paper:Paper)
      WITH c, communityPapers, conf, collect(DISTINCT paper) AS confPapers
      WHERE size(confPapers) > 0 AND
            size([p IN confPapers WHERE p IN communityPapers]) ⩾ 0.9 * size(confPapers)
      MERGE (conf)-[:RELATED_TO]→(c)

      // For journals (separate handling)
      WITH c, communityPapers
      MATCH (journal:Journal)←[:Published_in]-(paper:Paper)
      WITH c, communityPapers, journal, collect(DISTINCT paper) AS journalPapers
      WHERE size(journalPapers) > 0 AND
            size([p IN journalPapers WHERE p IN communityPapers]) ⩾ 0.9 * size(journalPapers)
      MERGE (journal)-[:RELATED_TO]→(c)

      RETURN "Related venues identified" AS Result
```

3. **Next, we want to identify the top papers of these conferences/workshops/journals. We need to find the 100 papers with the highest number of citations from papers of the database community). As a result we will obtain the top-100 papers of the database community.**

   We use the conferences/journals related to the science community using the relationship established in the previous step. We then find the top 100 papers published in journals/conferences in the science community and link these papers to the science community using the :TOP_PAPER_IN relationship. This relationship is also assigned an attribute with the number of citations for the paper.

```
MATCH (c:Community {name: "Science Community"})
MATCH (venue)-[:RELATED_TO]→(c)
WHERE venue:Conference OR venue:Journal
WITH c, collect(venue) AS relatedVenues

// For conferences, we need to go through Conference_Edition
MATCH (conf:Conference)
WHERE conf IN relatedVenues
MATCH (paper:Paper)-[:Published_in]→(edition:Conference_Edition)-[:Of]→(conf)
WITH c, relatedVenues, collect(DISTINCT paper) AS confPapers

// For journals
MATCH (journal:Journal)
WHERE journal IN relatedVenues
MATCH (paper:Paper)-[:Published_in]→(journal)
WITH c, confPapers, collect(DISTINCT paper) AS journalPapers

// Combine all community papers
WITH c, confPapers + journalPapers AS communityPapers

// Count citations from papers within the community
UNWIND communityPapers AS paper
MATCH (paper)-[:Cited_by]→(citingPaper)
WHERE citingPaper IN communityPapers
WITH paper, count(DISTINCT citingPaper) AS citationCount
ORDER BY citationCount DESC
LIMIT 100

// Create relationship between these top papers and the community
MATCH (c:Community {name: "Science Community"})
MERGE (paper)-[:TOP_PAPER_IN {citations: citationCount}]→(c)

RETURN "Top papers identified" AS Result
```

4. **Finally, an author of any of these top-100 papers is automatically considered a potential good match to review database papers, and we want to include this information in the graph. In addition, we want to identify and store gurus, i.e., reputable authors that would be able to review for top events or journals. Gurus are those authors of, at least, two papers among the top-100 identified.**

   We find the authors who Write the papers that are TOP_PAPER_IN the Science community and link them to the Science community using the POTENTIAL_REVIEWER_FOR relationship. We then find authors who wrote at least 2 papers that are top papers in the science community and link them to the science community node using the GURU_OF relationship with top_paper_count as an attribute of the relationship.

```
MATCH (c:Community {name: "Science Community"})
MATCH (paper:Paper)-[:TOP_PAPER_IN]→(c)
MATCH (author:Author)-[:Writes]→(paper)

// Mark all authors of top papers as potential reviewers
MERGE (author)-[:POTENTIAL_REVIEWER_FOR]→(c)

// Count papers per author to identify gurus (authors with at least 2 top papers)
WITH c, author, count(DISTINCT paper) AS paperCount
WHERE paperCount ⩾ 2

// Mark these authors as gurus
MERGE (author)-[:GURU_OF {top_paper_count: paperCount}]→(c)

RETURN count(DISTINCT author) AS GuruCount
```

D Algorithms

1. Node Similarity Algorithm: For Paper Similarity Based on Topics

```
query = """
        CALL gds.graph.project(
        'paperTopicGraph',
        ['Paper', 'Keyword'],
        {   About: {
            type: 'About',
            orientation: 'UNDIRECTED'
            } } )
        """
```

```
query = """

        CALL gds.nodeSimilarity.stream('paperTopicGraph')
        YIELD node1, node2, similarity
        RETURN
        gds.util.asNode(node1).DOI AS Paper1_DOI,
        gds.util.asNode(node2).DOI AS Paper2_DOI,
        similarity
        ORDER BY similarity DESC
        Limit 10;

        """
```

First, we created a projected graph called 'paperTopicGraph' with Paper and Keyword nodes connected via About relationships. This projection captures which papers are associated with which research topics (keywords). The Node Similarity algorithm calculates the similarity between node pairs based on their shared neighbors, which in our case shared keywords. If two papers are both connected to similar sets of keywords, their similarity score will be high. This algorithm helps identify papers that are semantically similar based on the research topics they cover, which can assist scholars in finding related work or alternative methods addressing similar problems. It can also be used to assign reviewers to submissions based on topic similarity.

2. PageRank: Identifying the most influential papers in the network considering citations. This will help determine which papers are most influential based on how many other papers cite them and how important those citing papers are.

```
query = """
CALL gds.graph.project(
  'citationGraph',
  'Paper',
  {   Cited_by: {
      type: 'Cited_by',
      orientation: 'REVERSE'
    }})
"""
```

```
query = """
CALL gds.pageRank.stream('citationGraph')
YIELD nodeId, score
MATCH (p:Paper) WHERE id(p) = nodeId
RETURN p.DOI AS paper, score
ORDER BY score desc
LIMIT 10
"""
```

First, we created a projected graph called 'citationGraph' containing only Paper nodes and Cited_by relationships (reversed to model flow of influence from citing to cited papers). PageRank then measures the influence of nodes in a directed graph by considering not just the number of incoming citations, but also the quality or influence of the citing papers. A paper cited by highly-cited papers will receive a higher PageRank score. This algorithm provides insight into which papers are the most influential or foundational within the network. A high PageRank score in our case implies that a paper has had widespread impact in the research community.