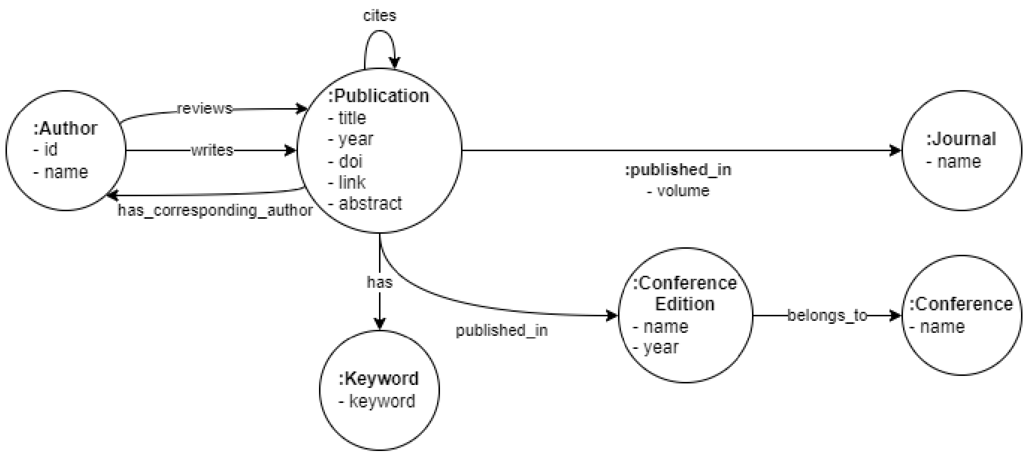
A Modeling, Loading, Evolving

A.1 Modeling

1. The metadata/schema visual representation of the graph is shown below.



2. The design decisions made are explained below.

- We decided to represent the corresponding author as an edge because such person doesn't have any differences in properties between them and other types of authors.
- Because there is a query that distinguishes conference editions, we decided to represent it as a node, while the volume of a journal is being represented in a simpler way as a property of the edge published_in, because we don't have queries specifying different volumes.
- Since we will eventually query a community of keywords we decided it's best to represent them as nodes.

We have made the following assumption.

- Since we are modeling a graph for publications, we assume that rejected articles (and therefore not published) won't be represented in this database.

A.2 Instantiating/Loading

1. The Cypher expressions to create and instantiate the solution are in a file named load.cypher. It is being executed by a Python application attached to the project.

We obtained the base for our data from the BYU Engineering Publications in Scopus 2017-21 Kaggle dataset[1] as a csv file, which presents data about publications and already provided real

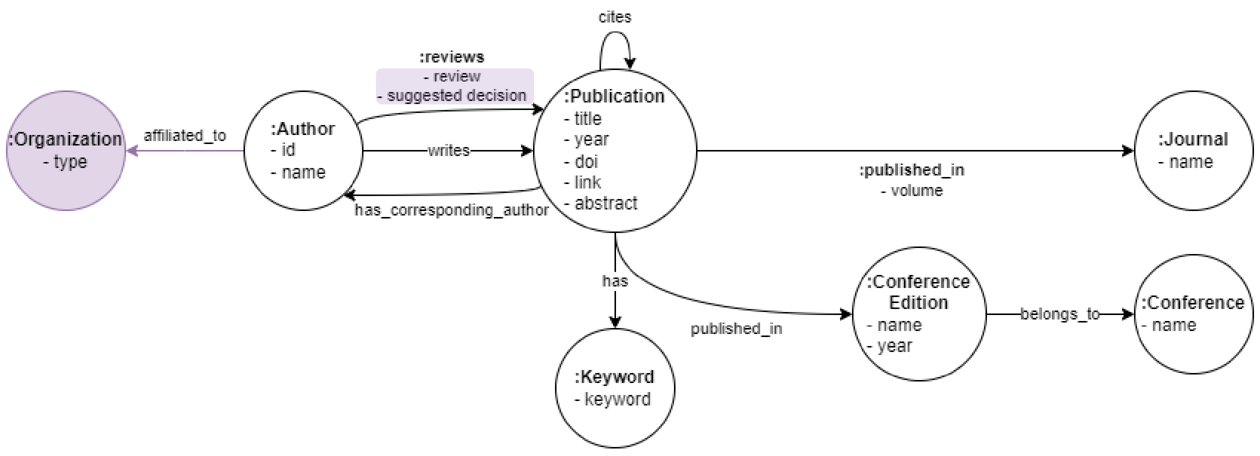
[1] https://www.kaggle.com/dpixton/byu-engineering-publications-in-scopus-201721/version/1

data regarding authors, publication title, publication year, volume, DOI, access link, author's affiliations, keywords, document type (if article, conference paper or other), and publication stage. The Python script generate_data.py processes this file with the following steps:

1. We only used articles and conference papers.
2. We discarded data that does not have one of the fields needed.
3. We removed the edition information such as year and edition number from the conference names
4. We parsed the author's affiliations and generated a separate csv file with them.
5. Because we lacked sufficient papers in the same conference but in four different years (for one of the queries requested), we also synthetically created extra papers for the previous and/or next years.
6. To have conferences/journals belonging to the database community (for section D), we selected the 15 conferences/journals with more publications, and added one of the community keywords to each of them.
7. We synthetically created the citation links between articles and stored them in a separate csv file. For creating such citations, we made sure that the papers being cited were from a previous year and had at least one keyword in common with the paper citing them. A paper could have from 0 to 20 citations.
8. Since we didn't have reviewer data, we chose up to 3 authors that have at least one keyword in common with the paper and assigned them as reviewers, saving this information in a separate csv file. We have also generated some text for the reviews themselves and stored them as a separate csv. Since we assumed all publications would be accepted in the end, there can be up to 1 reviewer rejecting each of them.

A.3 Evolving the graph

1. The modifications made on the graph model are shown below and highlighted in purple.



We decided to represent the university and company that an author is affiliated to by only one node called Organization and having it differentiated between university and company by its "type" property.

We also decided to represent the review and suggested decision of a reviewer as properties of the relationship "reviews" since we thought it wouldn't make sense to create a separate node just for a reviewer (since they are authors as well).

B Querying

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

```
MATCH (:Publication)-[r:cites]->(cited_paper:Publication)-
[:published_in]->(:ConferenceEdition)-[:belongs_to]->(conf:Conference)
WITH conf, cited_paper, COUNT(r) AS num_citations
ORDER BY conf, num_citations DESC
WITH conf, collect(cited_paper)[..3] AS top3_most_cited_papers
RETURN conf,
       top3_most_cited_papers[0] AS top1_paper,
       top3_most_cited_papers[1] AS top2_paper,
       top3_most_cited_papers[2] AS top3_paper;
```

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

```
MATCH (a:Author)-[:writes]->(p:Publication)-[:published_in]-
>(ce:ConferenceEdition)-[:belongs_to]->(c:Conference)
WITH c, a, COUNT(DISTINCT ce) AS distinct_editions
WHERE distinct_editions >= 4
RETURN c, a; //replace a with COLLECT(a) for a single row per conference
```

3. Find the impact factors of the journals in your graph.

```
MATCH (journal:Journal)<-[:published_in]-(p:Publication)<-[c:cites]-
(:Publication)
WITH journal, p.year AS year, count(c) AS citations
MATCH (journal)<-[:published_in]-(p:Publication)
  WHERE p.year = year - 1 OR p.year = year - 2
WITH journal, year, citations, count(p) AS publications
  WHERE publications > 0
RETURN journal, year, toFloat(citations) / publications AS ImpactFactor
  ORDER BY ImpactFactor DESC;
```

4. Find the h-indexes of the authors in your graph.

```
MATCH (author:Author)-[:writes]->(paper:Publication) <-[c:cites]-
(:Publication)
WITH author, paper, count(c) AS citations
```

```
  ORDER BY author, citations DESC
WITH author, collect(citations) AS paper_citations
RETURN author,
       reduce(hindex = 0,
       citations IN paper_citations |
       CASE WHEN citations > hindex THEN hindex + 1
         ELSE hindex
         END) AS `H-Index`;
```

C Graph Algorithms

We are going to apply the Page Rank algorithm in order to find the most relevant papers, and the Louvain algorithm to find communities of publications by comparing how densely they cite each other against how connected they would be in a random network. Since for both we use the Publication nodes and the Cites edges, we use the same projection for both:

```
CALL gds.graph.create(
  'publications',
  'Publication',
  'cites'
)
```

Page Rank (Centrality Algorithm)

We use the Page Rank algorithm to find the most relevant papers using the citation network. The score provided by the Page Rank algorithm is probably more relevant than the number of citations, since it takes into account the importance of the citing papers. For example, a paper that is cited 5 times by papers that nobody cites is probably less relevant than a paper that is cited by only 2 papers that a lot of papers cite.

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

Louvain (Community Detection)

We use the Louvain algorithm to find communities among the papers, by detecting communities of papers that cite each other more densely than if they were cited in a random network. Since it is a hierarchical clustering algorithm, it can show not only the main communities but also secondary communities found in the graph.

```
CALL gds.louvain.stream('publications')
YIELD nodeId, communityId, intermediateCommunityIds
RETURN gds.util.asNode(nodeId).title AS title, communityId,
intermediateCommunityIds
ORDER BY title ASC
```

D Recommender

To get the potential reviewers for the database community, we used the following queries. For sake of brevity and space, we are omitting some of the intermediate queries that are used as part of other queries (e.g. get top 100 papers is used in get recommended reviewers). All the queries can be found in the attached file “recommender_queries.cypher”.

- Find the database community defined through keywords

```
MATCH (n:Publication)-[h:has]->(k:Keyword)
WHERE k.keyword IN ["data management", "indexing", "data modeling", "big
data", "data processing", "data storage","data querying"]
RETURN *;
```

- Find the conferences and journals related to the database community

```
MATCH (p:Publication)-[:published_in]->()-[:belongs_to*0..1]->(c)
 WHERE c:Conference OR c:Journal
WITH c, count(DISTINCT p) as total_papers
MATCH (k:Keyword)<-[:has]-(p_db:Publication)-[:published_in]->()-
[:belongs_to*0..1]->(c)
  WHERE k.keyword IN ["data management", "indexing", "data modeling",
"big data", "data processing", "data storage", "data querying"]
WITH c, total_papers, count(DISTINCT p_db) as database_community_papers
  WHERE toFloat(database_community_papers)/total_papers > 0.9
RETURN c;
```

- Find the top papers of these conferences/journals

  We assumed that the page rank calculation must only be done by citations from the papers of the same community, hence the projection having only such papers.

```
CALL gds.graph.create.cypher(
  // graph name
  'papers_in_database_communities_journals_or_conferences',
  // node query: return the publications published in
journals/conferences from the database community
 ' ... PREVIOUS QUERY EXCEPT RETURN ...
   WITH c
    MATCH (p:Publication)-[:published_in]->()-[:belongs_to*0..1]->(c)
      WHERE c:Conference OR c:Journal
    RETURN id(p) AS id',
  // relationship query
  'MATCH (n:Publication)-[r:cites]->(m:Publication) RETURN id(n) AS
source, id(m) AS target',
  {validateRelationships:false}
);
```

We run the PageRank algorithm and write the page rank as a property for the papers described above

```
CALL
gds.pageRank.write('papers_in_database_communities_journals_or_conferenc
es', {
  maxIterations: 20,
  dampingFactor: 0.85,
  writeProperty: 'pagerank'
});
```

With the page rank, we can find the 100 most relevant publications of the community (we assume that we are obtaining 100 papers in total, not 100 per conference/journal), which is then used to obtain the potential reviewers and to identify gurus among them.

```
MATCH (p:Publication)-[:published_in]->()-[:belongs_to*0..1]->(c)
      WHERE c:Conference OR c:Journal
WITH c, count(DISTINCT p) as total_papers
MATCH (k:Keyword)<-[:has]-(p_db:Publication)-[:published_in]->()-
[:belongs_to*0..1]->(c)
  WHERE k.keyword IN ["data management", "indexing", "data modeling",
"big data", "data processing", "data storage","data querying"]
WITH c, total_papers, count(DISTINCT p_db) as database_community_papers
  WHERE toFloat(database_community_papers)/total_papers > 0.9
WITH c
MATCH (p:Publication)-[:published_in]->()-[:belongs_to*0..1]->(c)
  WHERE c:Conference OR c:Journal
WITH p
   ORDER BY c
   LIMIT 100
MATCH (a:Author)-[w:writes]->(p:Publication)
WITH a, count(w) AS number_of_written_papers
RETURN a AS PotentialReviewer,
      number_of_written_papers >= 2 AS Guru;
```