# Property Graph Lab: Team OD-11E

MEYSAM ZAMANI FOROOSHANI - JAVIER MORENO FERNANDEZ

meysam.zamani@est.fib.upc.edu - javier.moreno.f@est.fib.upc.edu

## A) Modeling, Loading, Evolving

### A.1) Modeling

In this section we will create a visual representation of our graph and justify the most important design decisions considering maintainability and Part-B queries performance.

We have used draw.io, a very popular free online diagram software, to draw Figure 1. Following the statement we have included irrelevant data in the graph like "Npages"to make the exercise more realistic. We have decided that Author, Paper, Journal, Conference and keyword will be the main nodes. They are our domain principal components due their relationships and the B section queries. These are the main justifications and assumptions of our design:

- isCited -> In none of the required queries we will use need how many papers is a paper citing but we will need how many are citing it. (i.e. it's better than cites)
- isPresentedIn -> The direction of the edge was problematic, as the most natural queries for B.2 and B.3 were opposite. We have preferred keeping only this one to reduce the graph size and efficiently take advantage of the for each of B.3. (detailed in Part B)
- edition -> we have decided to put this information in an edge because even if we have some redundancy queries will access faster than using another node.
- isPublishedIn -> symmetric to isPresentedIn.
- keyword -> to facilitate Part D we will use a separate node to express keywords.

### A.2) Instantiating/Loading

In this section we will define the Cypher expressions to create and instantiate the solution created for the previous section. We will use Python scripts and Java for the solutions of this part. (Java for preprocessing of the datasets and Python for connecting and loading data in to the Neo4j)

First lets see how we found the data and preprocessed them for our project:
In terms of preparing the data, we have build a java program (you can find it in our zip file as preprocessing-dataset.zip) that extracts information from the URL of dropbox given by the project statement. Then, in order to prepare our dataset, we got the names of the authors from authors.csv, and we got their nationality , age and email made by a random function. From the articles.csv we got the title of the article and the year that was published. The values from papers that are generated randomly are Location,ISBN DOI, Volume, Npages, Edition. For the keywords, we have searched the major topics in computer science and we have searched the

major topics of studying for each topic and we have made a list of it.

If we join this all together we got 2 csv files, one for conferences and the other for journals, You can find these 2 csv files in the zip file and in the folder named "Final-datasets". Headers in our datasets are: AuthorName, Age, Nationality, Email, ArticleTitle, YearPublished, ConferenceName/JournalName, Location, ISBN, DOI, Volume, Edition, keyword1, keyword2, keyword3, keyword4.

Being keyword1,keyword2,keyword3,keyword4, the set of keywords that are attached to each article.

The solution can be found at PartA.2_MorenoZamani.py. It can roughly divided in 3 parts, database configuration, metadata loading and CSV loading part. In the configuration step, we create the driver to communicate with Neo4j and we remove all previous data from the database.In the loading step, we use the bulk load functionality of Neo4j for both CSV files of Conferences and Journals in a simple and efficient way. We also manually add the metadata graph.

## A.3) Evolving the graph

In this section we will show how flexible a graph database can be by expanding our model with new requirements without necessarily having to redo all previous steps.
We have two main changes in our new model: (Figure 2)

- Affiliations -> We have decided to add a new node type affiliation to represent this concept. We think it could be useful to make queries university depend although as we won't need to use it through the practice it's a bit subjective decision.
- Reviews -> We have decided to keep the same structure and add the new concepts content and decision as properties of the existing reviews relationship.

We have implemented the data/metadata transformation queries and example instances generation in the PartA.3_MorenoZamani.py script. In the transformation we need to tackled the reviews changes, we have added some properties to the relationship. We could leave old data without the new properties but for consistency we will set the properties of existing nodes to default values. In the generation step we show how easy is to add the new concept organization, we just have to create it and link it with the existing graph. We only add a single organization as an example but many could be done in bulk using similar code to Part A.2.

## B) Querying

In this section we will provide efficient Cypher code to obtain different information.

## B.1) Find the h-indexes of the authors in your graph.

The h-index is one of the most used productivity metrics for researchers. It's not hard to understand but its computation is not trivial to obtain. We will use the Wikipedia method.

h-index = max min(f(i),i)

We want to find the h-index of each author in the dataset.

```
1 MATCH (a:Author)-[:writes]->(p:Paper), (p:Paper)-[r:isCited]->(:Paper)
2 WITH a.name AS author, p.title AS title, COUNT(r) AS numCitations ORDER BY numCitations DESC
3 WITH author, collect({title:title, numCitations:numCitations}) AS papers
4 WITH author, [i IN RANGE(1, SIZE(papers)) | CASE WHEN i > papers[i-1]["numCitations"] THEN papers[i-1]["numCitations"] ELSE i END] AS list
5 RETURN author, REDUCE(h = 0, x IN list | CASE WHEN h < x THEN x ELSE h END) AS hindex
```

1 -> Match the necessary elements, authors, papers and isCiteds.
2 -> Obtain the table with author, paper and its numCitations and order it.
3 -> Obtain the table with all the papers and citations per author.
4 -> Apply the min function to the list per author.
5 -> Apply the max function to the list per author.

For our dataset we obtain h-indexes between 28 and 1 represent in a list of all authors presented in our dataset.

**B.2) Find the top 3 most cited papers of each conference.**

Knowing which are the most important papers in a conference is a very curious statistic. Following the statement, we will find the 3 most cited papers for each conference.

```
1 MATCH (p:Paper)-[:isPresentedIn]->(c:Conference), (p:Paper)-[r:isCited]->(:Paper)
2 WITH p.title AS title, c.name AS conference, COUNT(r) AS numCitations ORDER BY numCitations DESC
3 RETURN conference, collect({title:title, numCitations:numCitations})[0..3] AS papers
```

1 -> Match the necessary elements, conferences, papers, and isCiteds.
2 -> Obtain the table with conferences, papers and numCitations and order it.
3 -> Obtain the table with all the papers and citations per author and get the first 3.

For our dataset we obtain top 3 most cited papers of 20 different conferences in a list.

**B.3) For each conference find its community.**

A useful thing to know about a conference is who form its community , defined by authors who have published in 4 or more editions (i.e. researchers that value the conference).

```
1 MATCH (a:Author)-[:writes]->(:Paper)-[r:isPresentedIn]->(c:Conference)
```

```
2 WITH c.name AS conference, a.name AS author, COUNT(DISTINCT r.
    edition) AS numEditions
3 WHERE numEditions >= 4
4 RETURN conference, collect(author) AS community
```

1 -> Match the necessary elements, conferences, authors, and isPresented.
2 -> Obtain the table with all conferences, authors and the # of editions they've presented.
3 -> Filter by conference/author that have at least 4 different editions.
4 -> Join all authors for every conference into the list community.

For our dataset we obtain in 12 different conferences their communities.

### B.4) Find the impact factors of the journals in your graph.

The Impact Factor (IF) is one of the most frequently used metrics to evaluate a journal relevance. Although it can be defined in different ways depending on the period observed we will use the most common definition which is in any given year, the impact factor of a journal is the number of citations, received in that year, of articles published in that journal during the two preceding years, divided by the total number of "citable items"published in that journal during the two preceding years.

We will use as an example year = 2015 in the script. Following the previous subsections we will compute the IF of all journal in the same query.

```
1 MATCH (p:Paper)-[r:isCited]->(:Paper), (p:Paper)-[t:isPublishedIn
    ]->(j:Journal)
2 WHERE t.year = "2013" OR t.year = "2014"
3 WITH p.title AS paper, j.name AS journal, COUNT(r) AS numCitations
4 RETURN journal, AVG(numCitations) AS ImpactFactor
```

1 -> Match the necessary elements, papers, journals, and isCiteds.
2 -> Remove all papers from unwanted years.
3 -> Count the number of citations of each paper.
4 -> Compute the IF by averaging the number of citations per paper.

For our dataset we obtain impact factors of 20 different Journals.

## C) Graph algorithms

In this section we will use complex graph algorithms to extract useful information from our data. From the long list of possible algorithms we have chosen the Path Finding algorithm Single Source Shortest Path and the Betweenness Centrality algorithm.

NOTE: Because we have focused on efficient queries and made a minimalist graph (e.g. unidirectional edges) it's hard to come up with interesting metrics. We will work with paper and isCited nodes and understand what different algorithms do with the same data.

### C.1) Graph Algorithm 1: Single Source Shortest Path

We will apply this algorithm to a paper and we will constraint the path to isCited relationships, so we will only move through the paper nodes.

```
MATCH (p:Paper {title:'Minimum Feedback Vertex Sets in
    Cocomparability Graphs and Convex Bipartite Graphs.'})
CALL algo.shortestPath.deltaStepping.stream(p,"cost",1,{nodeQuery:'
    Paper', relationshipQuery:'isCited', defaultValue:1.0, direction
    :'OUTGOING'})
YIELD nodeId, distance
MATCH (n:Paper) WHERE id(n) = nodeId AND distance < 100
RETURN n.title AS paper, distance
```

With this algorithm we are obtaining for a paper p ("Minimum Feedback Vertex Sets in Cocomparability Graphs and Convex Bipartite Graphs." as a sample in our case) which papers has influenced (i.e. they cite it or cite papers that cite it...). It's a way to see the connections that a paper has and see how far two papers are from each other, its influence weight.

### C.2) Graph Algorithm 2: Betweenness Centrality

We will apply this algorithm to all papers using isCited relationships.

```
CALL algo.betweenness.stream('Paper','isCited',{})
YIELD nodeId, centrality
MATCH (p:Paper) WHERE id(p) = nodeId
RETURN p.title AS paper, centrality
```

For this query and this graph Centrality can be seen as a measure of two components together, how influential is our work (how many papers cite us) and how influential are the papers that we have cited. A high value of centrality means that our work is innovative, we are a bridge between past work and future papers (that are not related to the past ones).

### D) Recommender

In this section we will create a simple reviewer recommender. We will follow the 4 steps given in the problem statement. (each step will have its own Python script)

### D.1) Research Communities

NOTE: Through the exercise we have focused on journals but we could add conferences with very similar queries in D.2 and uniting the results in D.3.

```
MERGE (c:Community {name:"database"})
MATCH (k:Keyword {name:"sequence alignment"}), (c:Community {name:"
    database"}) MERGE (c)-[:definedBy]->(k)
MATCH (k:Keyword {name:"Machine Learning"}), (c:Community {name:"
    database"}) MERGE (c)-[:definedBy]->(k)
MATCH (k:Keyword {name:"type theory"}), (c:Community {name:"
    database"}) MERGE (c)-[:definedBy]->(k)
...
```

1 -> Create a community node (e.g. database)
2 -> Add keyword data management connection to the community
3 -> Add keyword indexing connection to the community
4 -> ...

**D.2) Related Journals**

```
1 MATCH (p:Paper)-[:isPublishedIn]->(j:Journal)
2 WITH j, p, EXISTS((:Community {name:"database"})-[:definedBy]->(:
    Keyword)-[:isIn]->(p:Paper)) AS inCommunity
3 WITH j, SUM(CASE WHEN inCommunity THEN 1 ELSE 0 END) AS
    numPapersCommunity, COUNT(*) AS numPapers
4 WHERE numPapersCommunity/numPapers >= 0.9
5 MATCH (c:Community {name:"database"}) MERGE (j)-[:isPartOf]->(c)
```

1 -> Match papers and journals
2 -> Mark papers that have a community keyword
3 -> Count community papers and total papers
4 -> Filter out journals with less than 90% community papers
5 -> Add relationship between a community and its related journals

**D.3) Top Papers**

```
1 CALL algo.pageRank.stream('MATCH (p:Paper)-[:isPublishedIn]->(:
    Journal)-[:isPartOf]->(:Community) RETURN id(p) as id', 'MATCH (
    p1:Paper)-[:isCited]->(p2:Paper) RETURN id(p1) as source, id(p2)
     as target', {iterations:20, dampingFactor:0.85, graph:'cypher'
    })
2 YIELD nodeId, score
3 MATCH (p:Paper) WHERE id(p) = nodeId
4 WITH p, score ORDER BY score DESC LIMIT 10
5 SET p.top = TRUE
```

1 -> Call PageRank algorithm for the related journals papers subgraph
2 -> Obtain the scores for all papers
3 -> Recover the papers from their ids
4 -> Order the papers by PageRank and get top-10 papers
5 -> Mark those top-10 papers in the graph

**D.4) Candidate Reviewers**

```
1 MATCH (a:Author)-[:writes]->(p:Paper {top : TRUE})
2 WITH a.name as author, COUNT(*) as nPaper
3 RETURN author, (CASE WHEN nPaper = 1 THEN "member" ELSE "guru" END)
     AS status
```

1 -> Find the authors of the marked papers
2 -> Get how many of those papers the author has written
3 -> Compute its status according to that number

**FIGURA 1 – Visual graph representation (metadata + data example).**

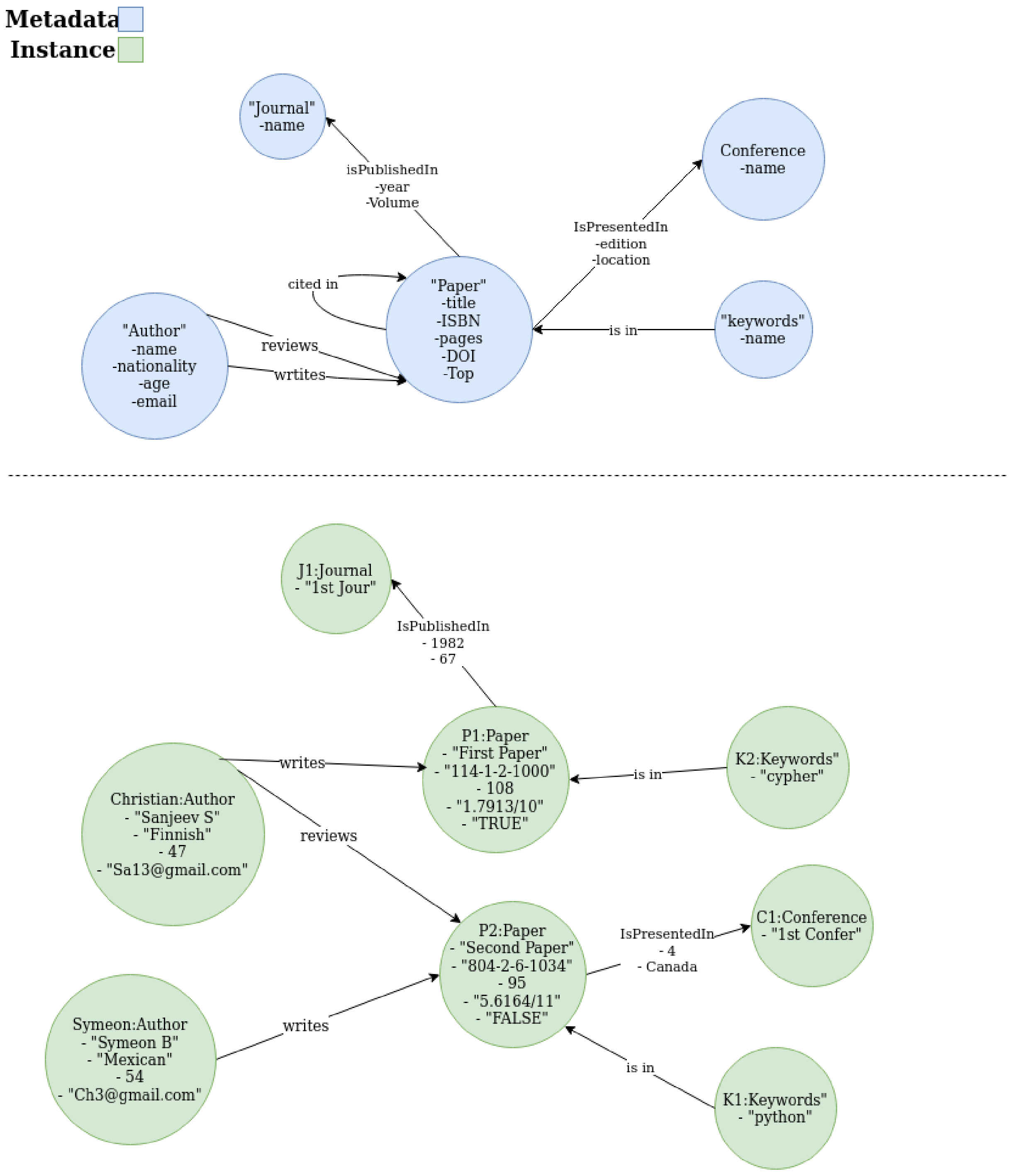
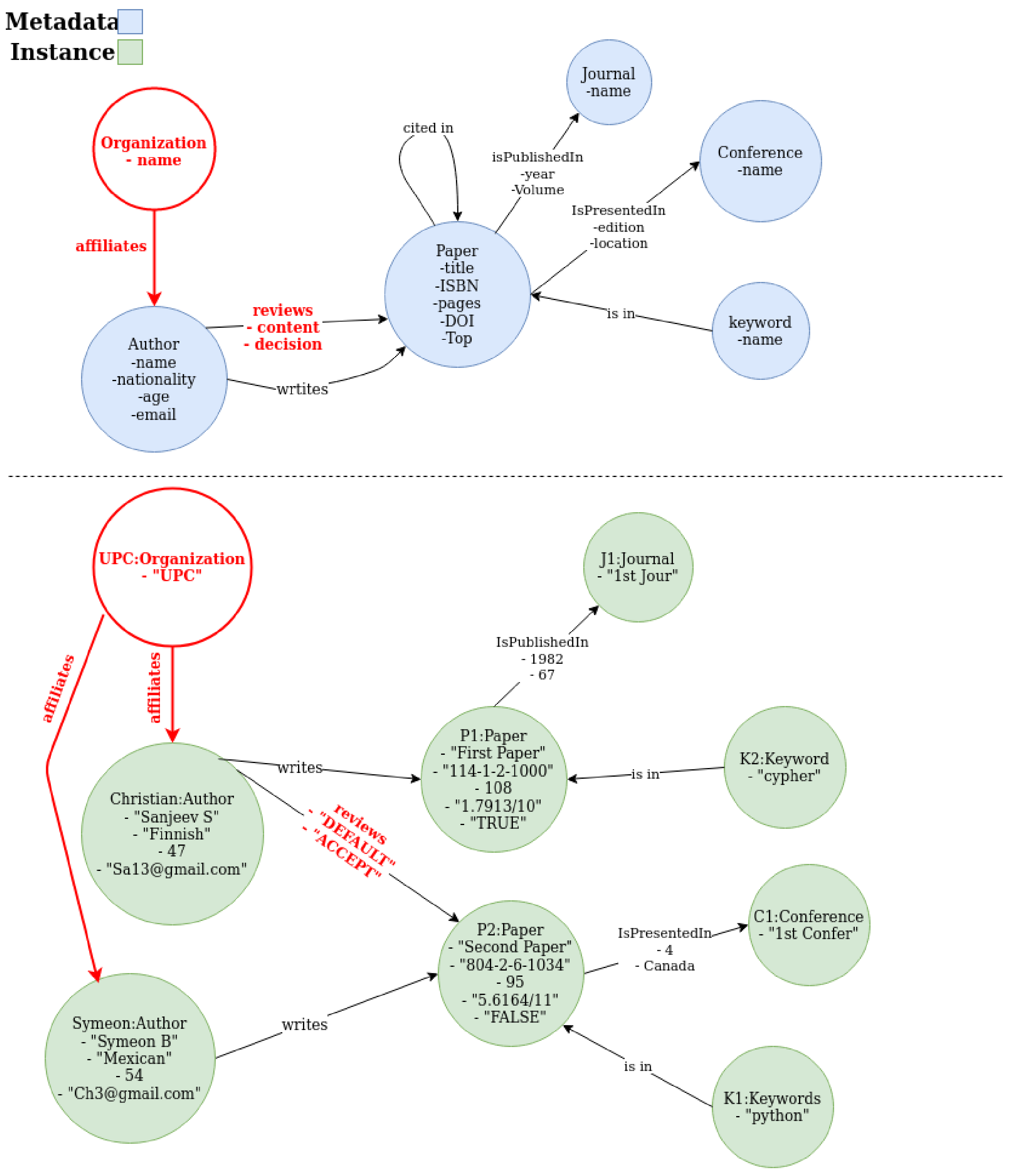

Metadata
Instance
"Journal" -name
isPublishedIn -year -Volume
Conference -name
IsPresentedIn -edition -location
"Paper" -title -ISBN -pages -DOI -Top
cited in
"Author" -name -nationality -age -email
reviews
wrtites
is in
"keywords" -name
J1:Journal - "1st Jour"
IsPublishedIn - 1982 - 67
P1:Paper - "First Paper" - "114-1-2-1000" - 108 - "1.7913/10" - "TRUE"
K2:Keywords" - "cypher"
is in
writes
Christian:Author - "Sanjeev S" - "Finnish" - 47 - "Sa13@gmail.com"
reviews
P2:Paper - "Second Paper" - "804-2-6-1034" - 95 - "5.6164/11" - "FALSE"
IsPresentedIn - 4 - Canada
C1:Conference - "1st Confer"
Symeon:Author - "Symeon B" - "Mexican" - 54 - "Ch3@gmail.com"
writes
is in
K1:Keywords" - "python"

FIGURA 2 – Visual evolved graph representation (metadata + data example).



Metadata
Instance
Organization - name
affiliates
Author -name -nationality -age -email
reviews - content - decision
wrtites
Paper -title -ISBN -pages -DOI -Top
cited in
isPublishedIn -year -Volume
Journal -name
IsPresentedIn -edition -location
Conference -name
is in
keyword -name
UPC:Organization - "UPC"
affiliates
affiliates
Christian:Author - "Sanjeev S" - "Finnish" - 47 - "Sa13@gmail.com"
writes
reviews - "DEFAULT" - "ACCEPT"
P1:Paper - "First Paper" - "114-1-2-1000" - 108 - "1.7913/10" - "TRUE"
IsPublishedIn - 1982 - 67
J1:Journal - "1st Jour"
is in
K2:Keyword - "cypher"
P2:Paper - "Second Paper" - "804-2-6-1034" - 95 - "5.6164/11" - "FALSE"
IsPresentedIn - 4 - Canada
C1:Conference - "1st Confer"
is in
K1:Keywords - "python"
Symeon:Author - "Symeon B" - "Mexican" - 54 - "Ch3@gmail.com"
writes