Universitat Politècnica de Catalunya

Facultad de Informática de Barcelona

SDM Lab Assignment 2

Semantic Data Management

Spring 2024

Authors:
Arijit Samal
arijit.samal@estudiantat.upc.edu
MD Kamrul Islam
md.kamrul.islam@estudiantat.upc.edu

Supervisors:
Prof. Oscar Romero

Contents

1 Section B.1 : TBOX Definition

1.0.1 Modelling the TBOX

To define the TBox (terminological component) of our ontology for our LAB 02, we adopted a systematic approach using Python's RDFLib library. Below is a detailed description of the steps we followed:

1. **Namespace and Graph Setup:**

   We started by setting up a namespace to ensure all URIs in our ontology are unique. We used the namespace `research`, defined as `Namespace("http://www.example.edu/research/")`. We then created an RDF graph using pydotplus's `Graph()` function to hold all our triples.

2. **Defining Classes and Properties:**

   We defined several key classes to represent the main entities in our research publication domain. These included `paper`, `author`, `conferences`, `workshops`, `journals`, `keywords`, `proceedings`, and `reviewers`. Each class was added to the graph as an instance of `RDFS.Class`.

   For example, for the `paper` class, we added:

```
g.add((research.paper, RDF.type, RDFS.Class))
g.add((research.paper_title, RDFS.domain, research.paper))
g.add((research.paper_title, RDFS.range, XSD.string))
g.add((research.paper_title, RDF.type, RDF.Property))

g.add((research.paper_abstract, RDFS.domain, research.paper))
g.add((research.paper_abstract, RDFS.range, XSD.string))
g.add((research.paper_abstract, RDF.type, RDF.Property))

g.add((research.paper_pages, RDFS.domain, research.paper))
g.add((research.paper_pages, RDFS.range, XSD.string))
g.add((research.paper_pages, RDF.type, RDF.Property))

g.add((research.paper_DOI, RDFS.domain, research.paper))
g.add((research.paper_DOI, RDFS.range, XSD.string))
g.add((research.paper_DOI, RDF.type, RDF.Property))

g.add((research.paper_link, RDFS.domain, research.paper))
g.add((research.paper_link, RDFS.range, XSD.anyURI))
g.add((research.paper_link, RDF.type, RDF.Property))

g.add((research.paper_date, RDFS.domain, research.paper))
g.add((research.paper_date, RDFS.range, XSD.date))
g.add((research.paper_date, RDF.type, RDF.Property))
```

   Listing 1: Defining the paper class and its properties

   Similarly, for the `author` class, we added:

```
g.add((research.author, RDF.type, RDFS.Class))
g.add((research.author_name, RDFS.domain, research.author))
g.add((research.author_name, RDFS.range, XSD.string))
g.add((research.author_name, RDF.type, RDF.Property))
g.add((research.author_email, RDFS.domain, research.author))
g.add((research.author_email, RDFS.range, XSD.string))
g.add((research.author_email, RDF.type, RDF.Property))
```

Listing 2: Defining the author class and its properties

3. **Establishing Relationships:**

   We defined relationships between these classes using properties. We defined the author class as the domain for the property and the range as the paper class. For instance, the `writes` property connects an `author` to a `paper`:

```
g.add((research.writes, RDFS.domain, research.author))
g.add((research.writes, RDFS.range, research.paper))
g.add((research.writes, RDF.type, RDF.Property))
```

   Listing 3: Example of defining a property **(write)**

   We also defined sub-properties to capture hierarchical relationships. For example, `corresponding_author` is a sub-property of `writes`, and this edge exists only between an author and a paper if the author is the corresponding author (i.e., the first author) of that paper.:

```
g.add((research.corresponding_author, RDFS.domain, research.author))
g.add((research.corresponding_author, RDFS.range, research.paper))
g.add((research.corresponding_author, RDF.type, RDF.Property))
g.add((research.corresponding_author, RDFS.subPropertyOf, research.writes))
```

   Listing 4: Defining the sub-property **(corresponding_author)**

   We also defined subclass to capture the information about the reviewers. We created a reviewer class, which is a subclassof author, and these are the authors who have reviewed at least one paper,
   `reviewer` is a subclass of `author`:

```
g.add((research.reviewer, RDFS.subClassOf, research.author))
g.add((research.reviewer, RDF.type, RDFS.Class))
```

   Listing 5: Defining the class **(reviewer)**

   We repeated this process for other classes and properties, in order to ensure comprehensive definition of the research publication domain.

4. **Serialization and Visualization:**

   After defining the classes and properties, we serialized the RDF graph to an RDF file to ensure it could be loaded into GraphDB and use for visualization. This was done as follows:

```
directory = './TBOX_DATA/'
if not os.path.exists(directory):
    os.makedirs(directory)
filepath = os.path.join(directory, "tbox_final_test.rdf")
g.serialize(filepath, format="pretty-xml")
```

   Listing 6: Serializing the graph to RDF format

   We also visualized the graph to provide a graphical representation of the TBox using `rdf2dot` and `pydotplus`:

```
graph_directory= "./GRAPH/"
if not os.path.exists(graph_directory):
    os.makedirs(graph_directory)
def visualize(g):
    stream = io.StringIO()
    rdf2dot(g, stream, opts = {display})
    dg = pydotplus.graph_from_dot_data(stream.getvalue())
    png = dg.create_png()
    dg.write_png(os.path.join(graph_directory, "tbox_ontology.png"))
    display(Image(png))
visualize(g)
```

Listing 7: Visualizing the graph

1.0.2 Graphical Representation of the Tbox

To graphically represent our Tbox, we used WebVOWL which is web-based visualization of ontologies. Fig 1 illustrates our Tbox ontology.

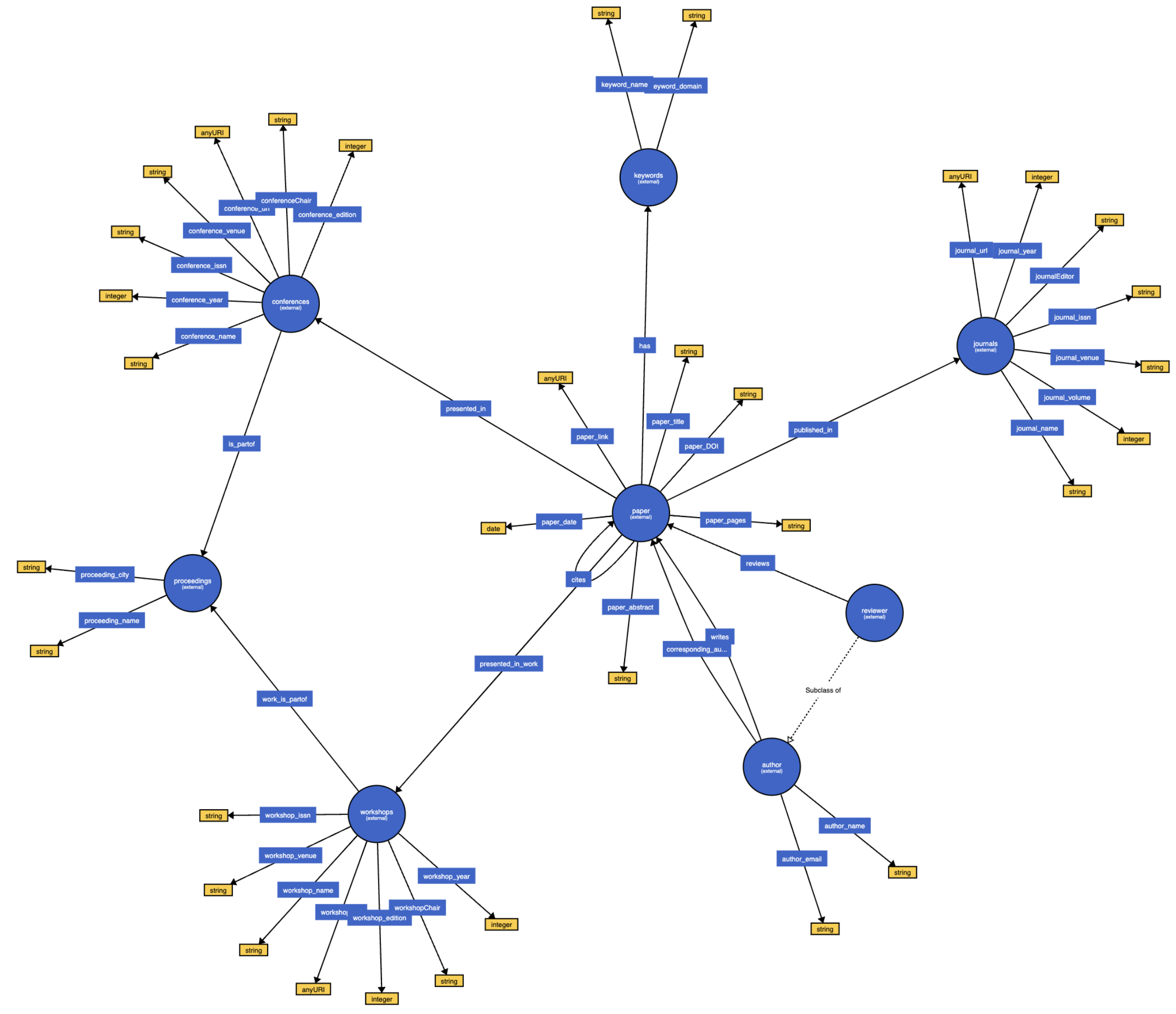


Figure 1: The Tbox using WebVOWL

2 Section B.2 : ABOX Definition

2.0.1 Modelling the Abox

To define the ABox (assertional component) of our ontology, we converted non-semantic data from CSV files into RDF triples using Python's RDFLib library. Here's a detailed explanation of our approach:

1. **Data Preparation:**

   We began with several CSV files containing the data for our research publication domain. These files included information about papers, authors, conferences, workshops, journals, keywords, proceedings, and their interrelationships. These CSV files were obtained from SDM LAB 1.

2. **Graph Initialization:**

   We created a namespace `research` using `Namespace("http://www.example.edu/research/")` to ensure that all URIs are unique and consistent. An RDF graph `g` was then created using RDFLib's `Graph()` function to store the triples.

3. **Loading and Processing Data:**

   Using the Pandas library, we read the CSV files into dataframes. For each class (e.g., papers, authors, conferences), we iterated through the rows of the corresponding dataframe and created RDF triples. Each row generated a URI for the instance and added properties using literals.

4. **Creating Instances and Properties:**

   For each class, we defined relevant properties and created instances. Here are some examples:

   Papers:

```
df_node_papers = pd.read_csv("DATA/CSV_FILES/papers.csv", delimiter=",")
for index, row in df_node_papers.iterrows():
    paper_id = URIRef(research + f"paper_{row['paperId']}")
    g.add((paper_id, RDF.type, research.paper))
    g.add((paper_id, research.paper_title, Literal(row["title"])))
    g.add((paper_id, research.paper_abstract, Literal(row["abstract"])))
    g.add((paper_id, research.paper_pages, Literal(row["pages"])))
    g.add((paper_id, research.paper_DOI, Literal(row["DOI"])))
    g.add((paper_id, research.paper_link, Literal(row["link"])))
    g.add((paper_id, research.paper_date, Literal(row["date"])))
```

Listing 8: Defining paper instances and their properties

   Authors:

```
df_node_authors = pd.read_csv("DATA/CSV_FILES/authors.csv", delimiter=",")
for index, row in df_node_authors.iterrows():
    author_id = URIRef(research + f"author_{row['authorId']}")
    g.add((author_id, RDF.type, research.author))
    g.add((author_id, research.author_name, Literal(row["name"])))
    g.add((author_id, research.author_email, Literal(row["email"])))
```

Listing 9: Defining author instances and their properties

5. **Establishing Relationships:**

   We defined relationships between entities using properties. For example, the `writes` property connects an author to a paper, and the `corresponding_author` property indicates the corresponding author for a paper.

```
df_node_author_writes_paper = pd.read_csv("DATA/CSV_FILES/author_writes.csv
    ", delimiter=",")
for index, row in df_node_author_writes_paper.iterrows():
    author_id = URIRef(research + f"author_{row['start_id']}")
    paper_id = URIRef(research + f"paper_{row['end_id']}")
    g.add((author_id, research.writes, paper_id))
    if str(row["corresponding_author"]).lower() == 'true':
        g.add((author_id, research.corresponding_author, paper_id))
```

Listing 10: Defining relationships between authors and papers

6. **Serializing the Graph:**

   After processing all entities and their relationships, we serialized the RDF graph to an XML file to ensure it can be easily shared and reused.

```
directory = './ABOX_DATA/'
if not os.path.exists(directory):
    os.makedirs(directory)
filepath = os.path.join(directory, "abox_final_test.rdf")
g.serialize(filepath, format="pretty-xml")
```

Listing 11: Serializing the graph to RDF format

3 Section B.3: Create the final ontology

To create the final ontology, we integrated the TBox (terminological component) and ABox (assertional component) into a unified RDF graph. This process involved linking the instances in the ABox to the classes and properties defined in the TBox, to ensure a coherent and consistent ontology. When the ABox and TBox are loaded into GraphDB, it automatically creates a link between ABox and TBox. We provide here a code to read all the triples from both the RDF files for TBox and ABox and load them into GraphDB using only this file. It is an optional step and not really needed as GraphDB handles the creation of the links internally. The steps below show the creation of the linked graph.

1. **Linking ABox Instances to TBox Classes:** Next, we linked the ABox instances to their corresponding TBox classes using the `rdf:type` property. This step ensures that each instance in the ABox is properly typed according to the definitions in the TBox.

```
print("Processing the classes...")
for abox_node in abox_graph.subjects(RDF.type, None):
    for tbox_class in tbox_graph.objects(abox_node, RDF.type):
        abox_graph.add((abox_node, RDF.type, tbox_class))
```

Listing 12: Linking ABox instances to TBox classes

2. **Linking ABox Edges to TBox Properties:** We then linked the edges (relationships) in the ABox to the properties defined in the TBox. This step ensures that all relationships between instances are correctly represented according to the ontology schema.

```
print("Processing the properties...")
for subject, predicate, object_ in abox_graph:
    if predicate is not None:  # Check if the predicate is not None
        if isinstance(predicate, Node):  # Ensure the predicate is an
    rdflib term
            abox_graph.add((subject, predicate, object_))
```

Listing 13: Linking ABox edges to TBox properties

3. **Importing into GraphDB:** Finally, we imported the serialized RDF graph into GraphDB. This was done using GraphDB's interface, enabling us to perform SPARQL queries on the complete ontology.

3.0.1 Inference Regime Entailment

We are using RDFS-optimized inference in GraphDB, which includes several key entailment rules that automatically generate new RDF triples based on existing ones. We don't have to explicitly mention rdf:type links to generate Some rules which include:

1. **Class Hierarchy (rdfs:subClassOf)**: If ClassA is a subclass of ClassB, then any instance of ClassA is also an instance of ClassB.
   Example-
   research:reviewer is a subclass of research:author, and instance1 is of type research:reviewer, then the inference rule will generate: **instance1 rdf:type research:author**

2. **Property Hierarchy (rdfs:subPropertyOf)**: If PropertyA is a subproperty of PropertyB, then any triple using PropertyA can also be inferred to use PropertyB.
   Example-
   research:corresponding_author is a subproperty of research:writes, and we have the triple: **author1 research:corresponding_author paper1** The inference rule will generate: **author1 research:writes paper1**

3. **Domain and Range**: If a property has a specified domain or range, any use of that property implies that the subject or object, respectively, is an instance of the domain or range class.
   Example-
   the domain of research:writes is research:author and the range is research:paper, and we have the triple: **author1 research:writes paper1** The inference rule will generate: **author1 rdf:type research:author paper1 rdf:type research:paper**

4. **Type Inference (rdf:type)**: Inferences can be made about the types of resources based on the properties they have and the classes they belong to.

3.0.2 Summary Table of Instances

The table 1 Provides a summary table of the instances. It shows the number of classes, the number of properties, number of instances for the classes and the number of triples for the properties.

Statistic	**Count**
Number of classes	13
Number of properties	48
Number of triples	100995
Instances per Class	
Paper	499
Author	4084
Conference	65
Workshop	504
Journal	268
Keyword	4269
Proceeding	4574
Reviewer	1136
Triples per Property	
writes	3461
corresponding_author	487
presented_in	96
presented_in_work	6
published_in	174
has	23588
cites	12102
reviews	1500

Table 1: Summary statistics of the knowledge graph.

4 Section B.4: Querying the ontology

1. **Query 1: Retrieve All Authors**

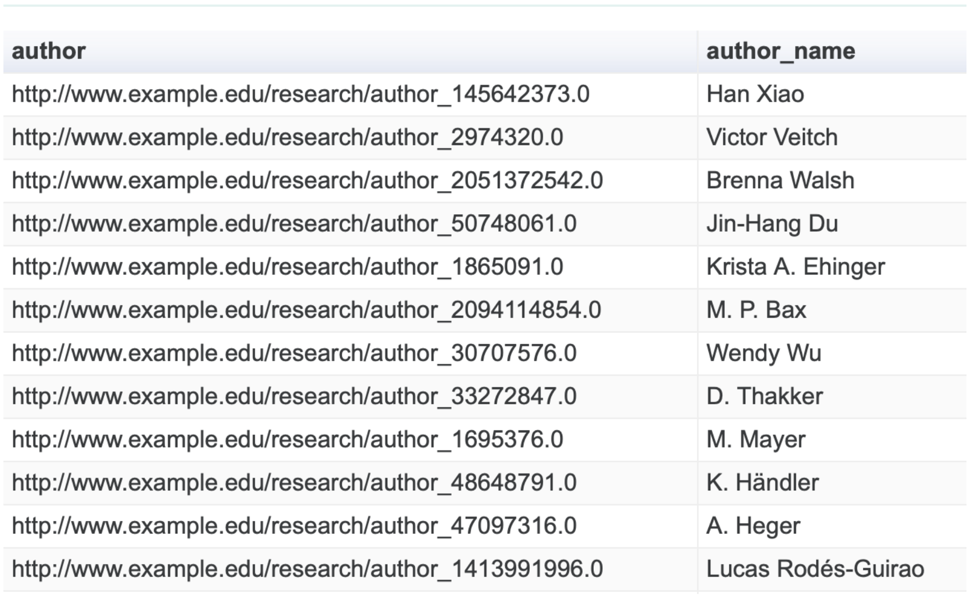
   This query retrieves all authors and their names from the ontology.

```
PREFIX research: <http://www.example.edu/research/>

SELECT ?author ?author_name
WHERE {
    ?author a research:author ;
            research:author_name ?author_name .
}
```

Listing 14: Retrieve all authors and their names

   The query selects all instances of the class `research:author` and their corresponding names.



author	author_name
http://www.example.edu/research/author_145642373.0	Han Xiao
http://www.example.edu/research/author_2974320.0	Victor Veitch
http://www.example.edu/research/author_2051372542.0	Brenna Walsh
http://www.example.edu/research/author_50748061.0	Jin-Hang Du
http://www.example.edu/research/author_1865091.0	Krista A. Ehinger
http://www.example.edu/research/author_2094114854.0	M. P. Bax
http://www.example.edu/research/author_30707576.0	Wendy Wu
http://www.example.edu/research/author_33272847.0	D. Thakker
http://www.example.edu/research/author_1695376.0	M. Mayer
http://www.example.edu/research/author_48648791.0	K. Händler
http://www.example.edu/research/author_47097316.0	A. Heger
http://www.example.edu/research/author_1413991996.0	Lucas Rodés-Guirao

Figure 2: Query 1: Retrieve All Authors

2. **Query 2: Find Properties with Domain Author**

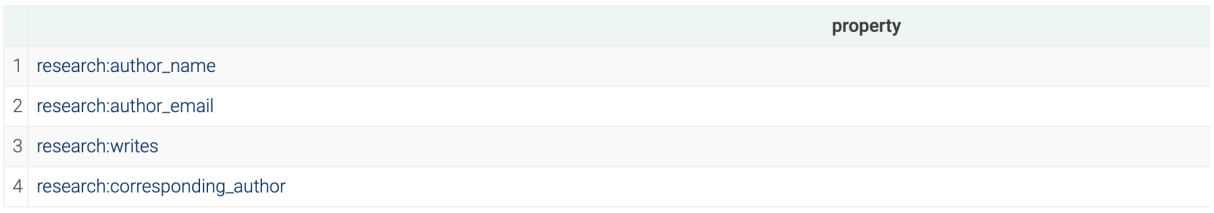
   This query identifies all properties whose domain is `research:author`.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX research: <http://www.example.edu/research/>

SELECT ?property
WHERE {
    ?property rdf:type rdf:Property ;
              rdfs:domain research:author .
}
```

Listing 15: Find properties with domain author

   The query selects all properties that have `research:author` as their domain, helping to understand which attributes or relationships are associated with authors.



	property
1	research:author_name
2	research:author_email
3	research:writes
4	research:corresponding_author

Figure 3: Query 2: Find Properties with Domain Author

3. **Query 3: Find Properties with Domain Conference or Journal**

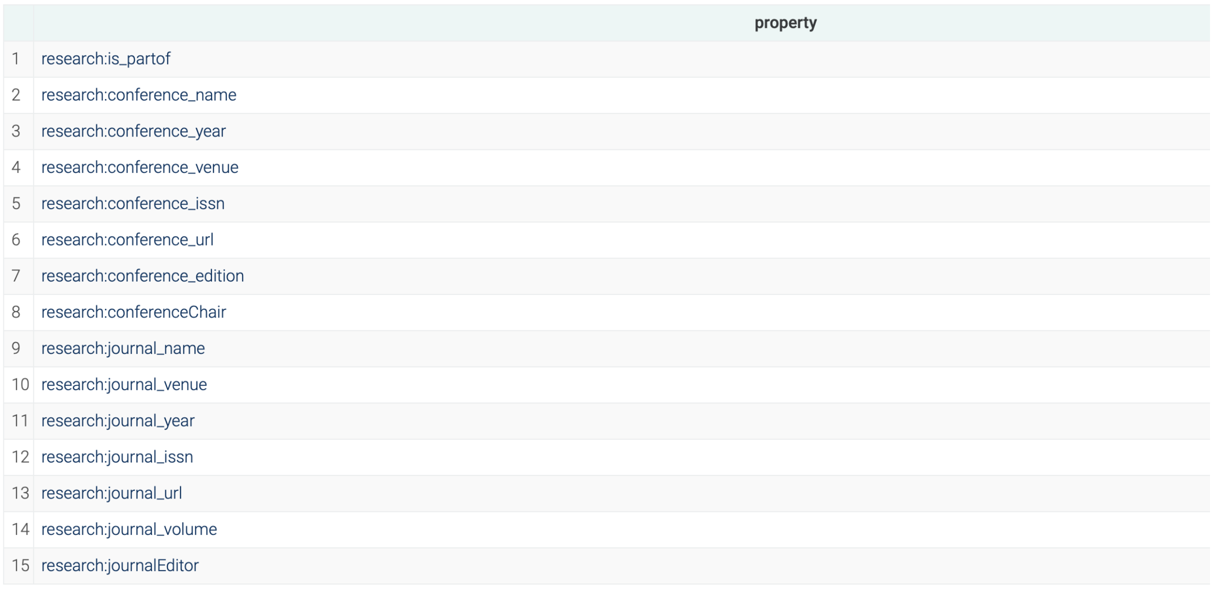
   This query finds all properties whose domain is either `research:conferences` or `research:journals`.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX research: <http://www.example.edu/research/>
```

```
SELECT ?property
WHERE {
  {
    ?property rdf:type rdf:Property ;
              rdfs:domain research:conferences .
  }
  UNION
  {
    ?property rdf:type rdf:Property ;
              rdfs:domain research:journals .
  }
}
```

Listing 16: Find properties with domain conference or journal

By using a `UNION` clause, the query combines results for properties related to both conferences and journals.



	property
1	research:is_partof
2	research:conference_name
3	research:conference_year
4	research:conference_venue
5	research:conference_issn
6	research:conference_url
7	research:conference_edition
8	research:conferenceChair
9	research:journal_name
10	research:journal_venue
11	research:journal_year
12	research:journal_issn
13	research:journal_url
14	research:journal_volume
15	research:journalEditor

Figure 4: Query 3: Find Properties with Domain Conference or Journal

4. **Query 4: Find Papers by Authors Presented at Database Conferences**

   This query retrieves distinct authors and their papers that were presented at database conferences, ordering the results by author name.

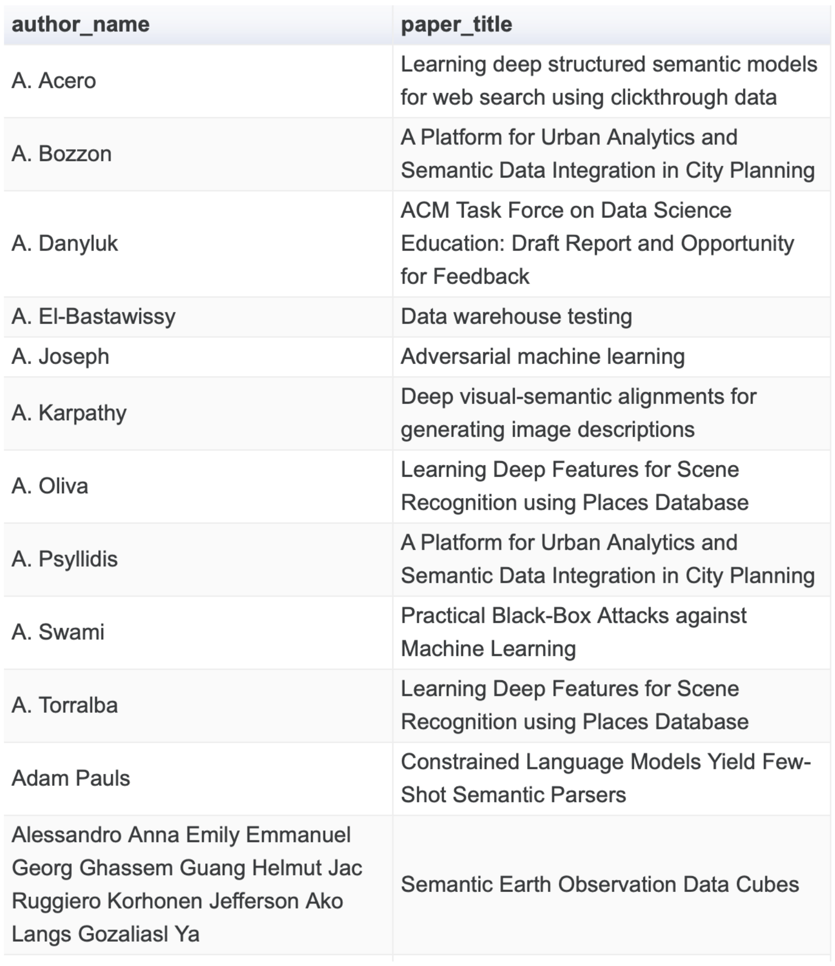
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX research: <http://www.example.edu/research/>

SELECT distinct ?author_name ?paper_title
WHERE {
  ?author rdf:type research:author ;
          research:author_name ?author_name ;
          research:writes ?paper .
  ?paper rdf:type research:paper ;
         research:paper_title ?paper_title ;
         research:presented_in ?conferences .
  ?conference rdf:type research:conferences ;
```

```
                research:conference_name ?conference_name .
}
order by ?author_name
```

Listing 17: Find papers by authors presented at database conferences

This query not only retrieves the necessary data but also ensures that the results are sorted alphabetically by the author's name for easier analysis.



author_name	paper_title
A. Acero	Learning deep structured semantic models for web search using clickthrough data
A. Bozzon	A Platform for Urban Analytics and Semantic Data Integration in City Planning
A. Danyluk	ACM Task Force on Data Science Education: Draft Report and Opportunity for Feedback
A. El-Bastawissy	Data warehouse testing
A. Joseph	Adversarial machine learning
A. Karpathy	Deep visual-semantic alignments for generating image descriptions
A. Oliva	Learning Deep Features for Scene Recognition using Places Database
A. Psyllidis	A Platform for Urban Analytics and Semantic Data Integration in City Planning
A. Swami	Practical Black-Box Attacks against Machine Learning
A. Torralba	Learning Deep Features for Scene Recognition using Places Database
Adam Pauls	Constrained Language Models Yield Few-Shot Semantic Parsers
Alessandro Anna Emily Emmanuel Georg Ghassem Guang Helmut Jac Ruggiero Korhonen Jefferson Ako Langs Gozaliasl Ya	Semantic Earth Observation Data Cubes

Figure 5: Query 4: Find Papers by Authors Presented at Database Conferences

5. **Query 5: Identify the Top 100 Conferences, Journals, and Workshops by Number of Papers**

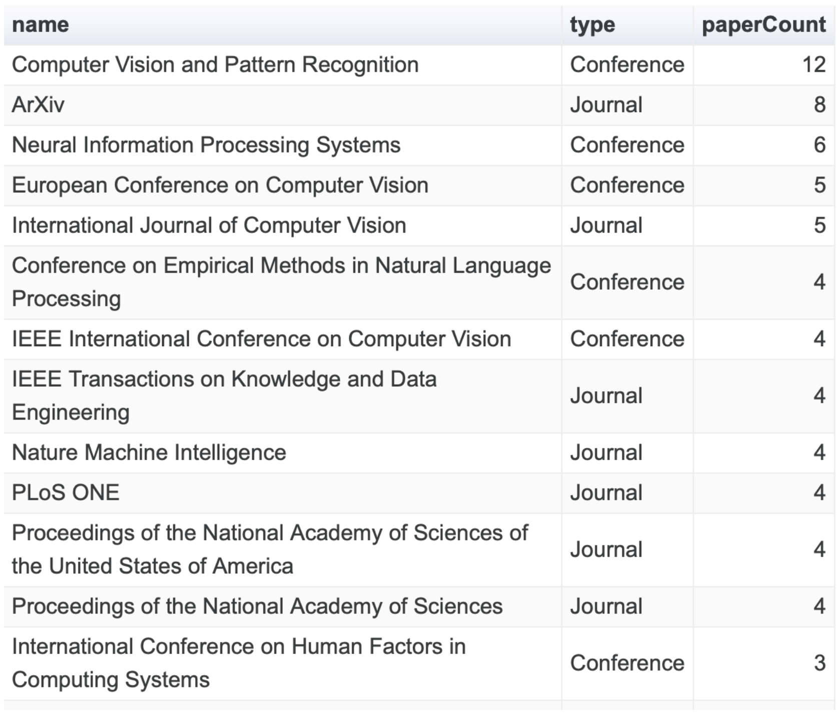
   This query retrieves the names and types of the top 10 conferences, journals, and workshops based on the number of papers presented or published. It groups the results by the venue name and type, and orders them by the count of papers in descending order.

```
PREFIX research: <http://www.example.edu/research/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?name ?type (COUNT(?paper) AS ?paperCount)
WHERE {
  {
    ?paper research:presented_in ?conference .
    ?conference rdf:type research:conferences .
    ?conference research:conference_name ?name .
    BIND("Conference" AS ?type)
  }
  UNION
  {
    ?paper research:presented_in_work ?workshop .
    ?workshop rdf:type research:workshops .
    ?workshop research:workshop_name ?name .
    BIND("Workshop" AS ?type)
  }
  UNION
  {
    ?paper research:published_in ?journal .
    ?journal rdf:type research:journals .
    ?journal research:journal_name ?name .
    BIND("Journal" AS ?type)
  }
}
GROUP BY ?name ?type
ORDER BY DESC(?paperCount)
LIMIT 100
```

Listing 18: Identify the top 10 conferences, journals, and workshops by number of papers

This query highlights the most active and possibly prestigious venues based on the number of papers they attract. It provides insights into where the bulk of research dissemination is happening and identifies key conferences, workshops, and journals.



name	type	paperCount
Computer Vision and Pattern Recognition	Conference	12
ArXiv	Journal	8
Neural Information Processing Systems	Conference	6
European Conference on Computer Vision	Conference	5
International Journal of Computer Vision	Journal	5
Conference on Empirical Methods in Natural Language Processing	Conference	4
IEEE International Conference on Computer Vision	Conference	4
IEEE Transactions on Knowledge and Data Engineering	Journal	4
Nature Machine Intelligence	Journal	4
PLoS ONE	Journal	4
Proceedings of the National Academy of Sciences of the United States of America	Journal	4
Proceedings of the National Academy of Sciences	Journal	4
International Conference on Human Factors in Computing Systems	Conference	3

Figure 6: Query 5: Identify the Top 100 Conferences, Journals, and Workshops by Number of Papers

6. **Query 6: Identify the Top 10 Most Cited Papers along with Authors**

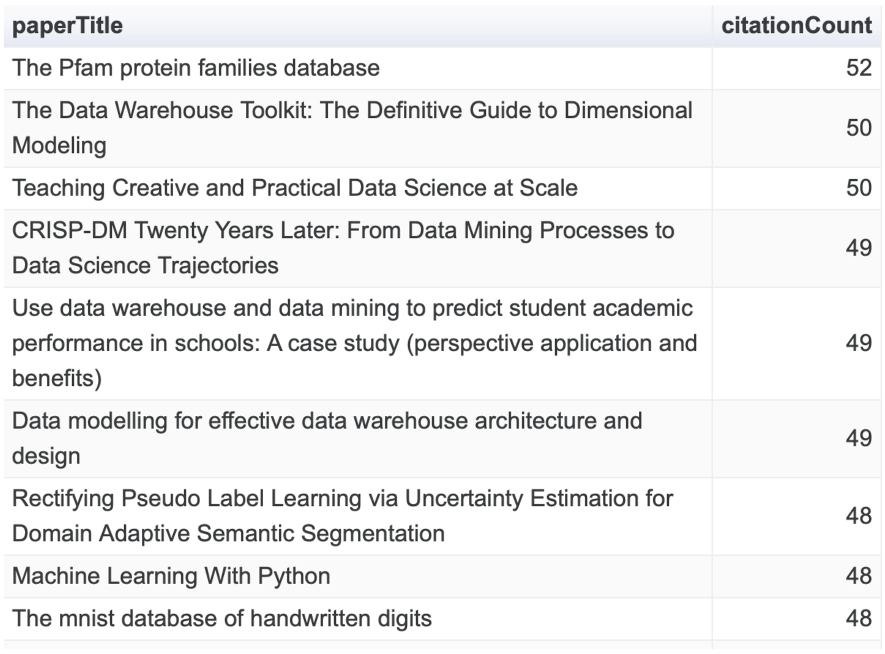
This query identifies the top 10 most cited papers by counting the number of citations each paper has received. It also retrieves the titles of these papers and the names of the authors who wrote them. The results are grouped by the paper title and author name, and ordered by the citation count in descending order.

```
PREFIX research: <http://www.example.edu/research/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?paperTitle (COUNT(?citingPaper) AS ?citationCount)
WHERE {
  ?paper rdf:type research:paper .
  ?paper research:paper_title ?paperTitle .
  ?citingPaper research:cites ?paper .
  ?author research:writes ?paper .
  ?author research:author_name ?authorName .
}
GROUP BY ?paperTitle ?authorName
ORDER BY DESC(?citationCount)
LIMIT 10
```

Listing 19: Identify the top 10 most cited papers along with authors

This query provides insights into the most influential papers in the research community based on the number of citations, highlighting the authors of these significant contributions.



paperTitle	citationCount
The Pfam protein families database	52
The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling	50
Teaching Creative and Practical Data Science at Scale	50
CRISP-DM Twenty Years Later: From Data Mining Processes to Data Science Trajectories	49
Use data warehouse and data mining to predict student academic performance in schools: A case study (perspective application and benefits)	49
Data modelling for effective data warehouse architecture and design	49
Rectifying Pseudo Label Learning via Uncertainty Estimation for Domain Adaptive Semantic Segmentation	48
Machine Learning With Python	48
The mnist database of handwritten digits	48

Figure 7: Query 6: Identify the Top 10 Most Cited Papers along with Authors