

**Program: CESS**

**Program: CESS**

***Course Code: CSE488***

***Course Name: Ontologies and the Semantic Web***

**Ain Shams University**

**Faculty of Engineering**

**Team 145**

**Hana Yasser 18P5007**

**Laila Mohamed 18P9654**

**Ilaria Refaat 18P3050**

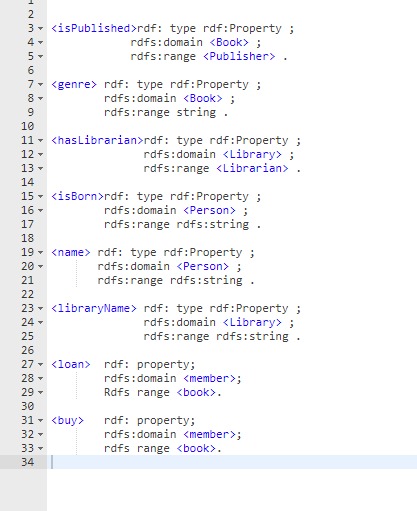
Library System Ontology

## Introduction:

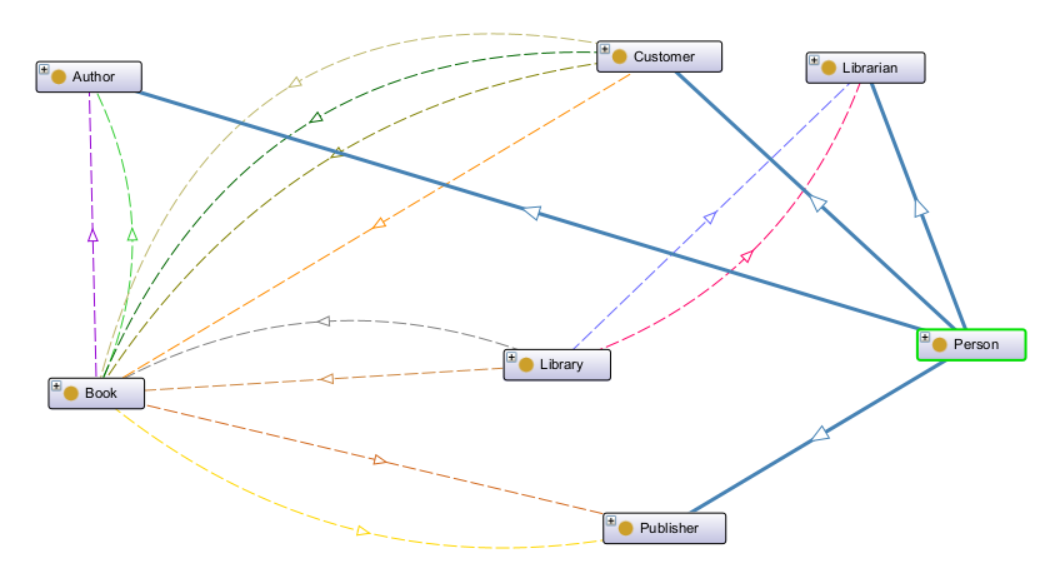
This project focuses on creating an ontology for a library system, defining the classes (e.g., Library, Book, Author) and their relationships. By leveraging ontology modeling, we establish a structured representation that enables efficient classification, organization, and retrieval of library information. This ontology serves as a foundation for enhancing search capabilities, enabling intelligent recommendations, and fostering interoperability within the library domain.

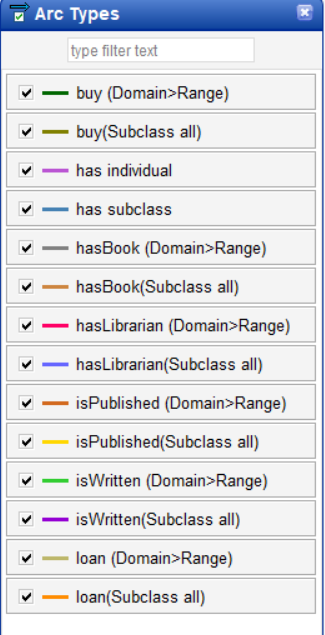
## RDF:





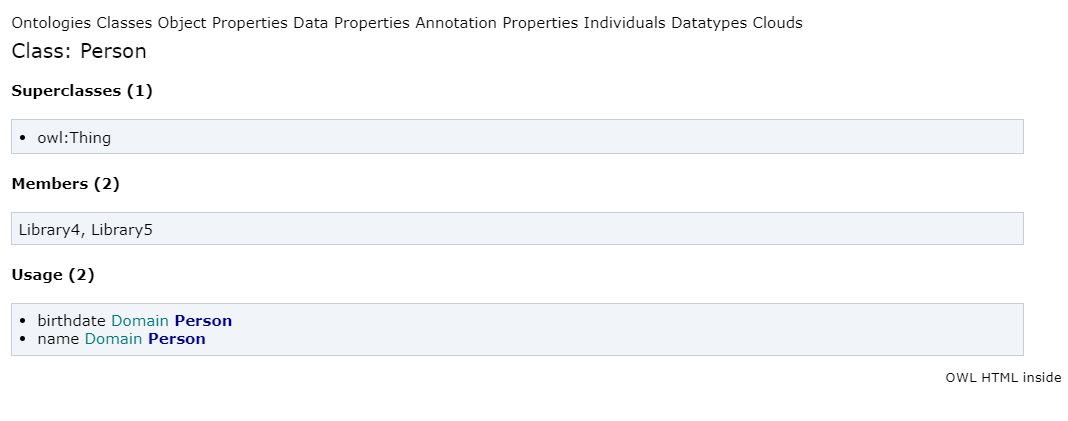
## RDF Graph:



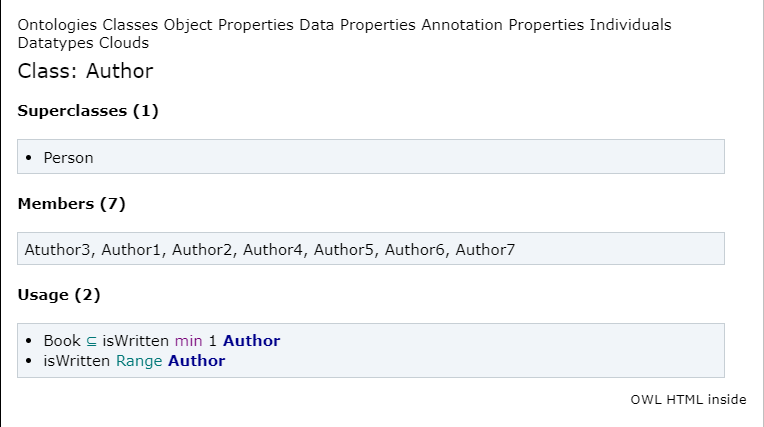


## Classes:

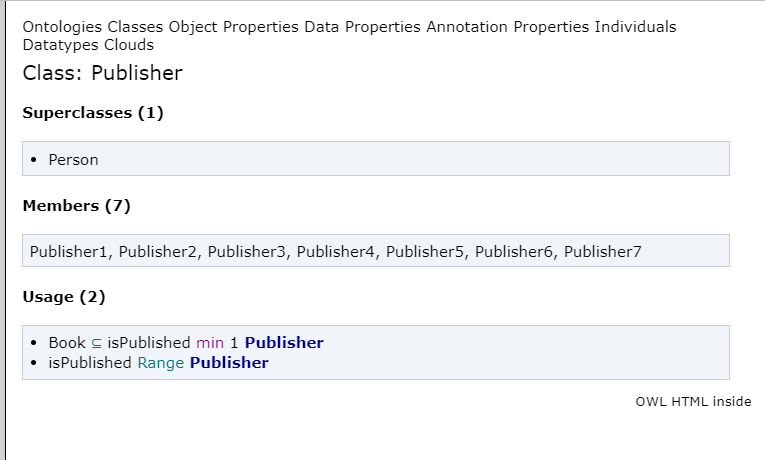
### Class Person:



### Class Author:



### Class Publisher:



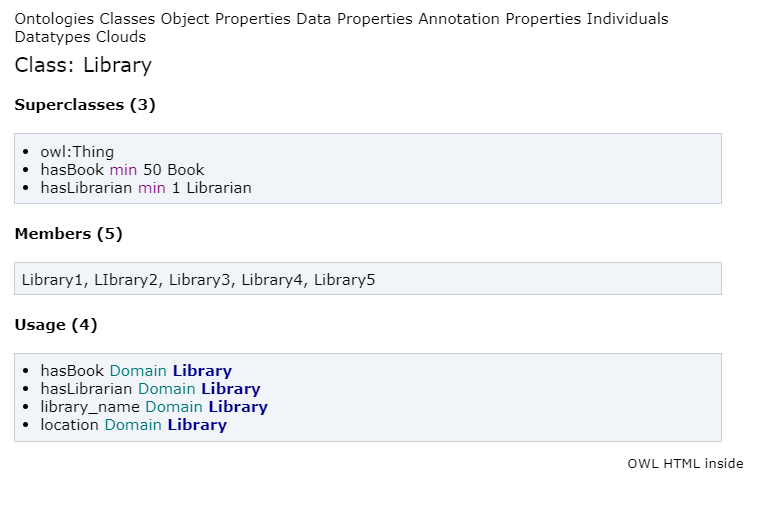
### Class Customer:



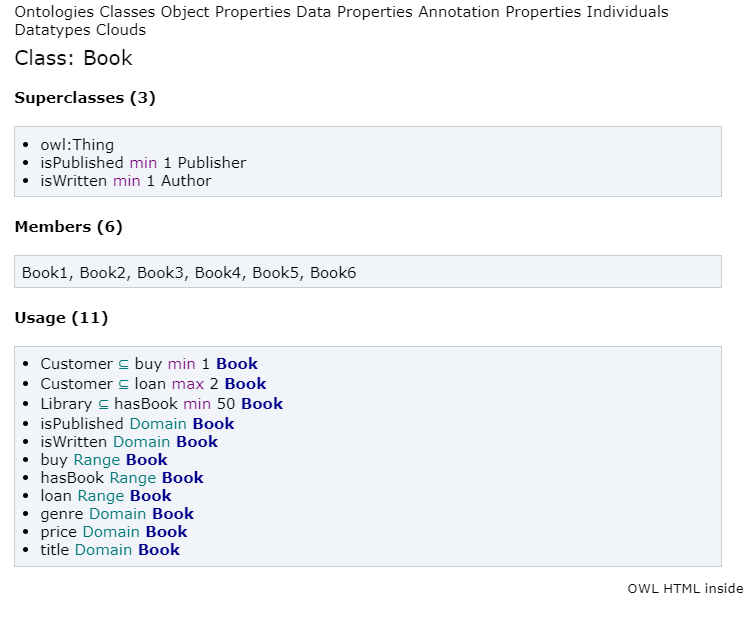
### Class Librarian:



### Class Library:



### Class Book:

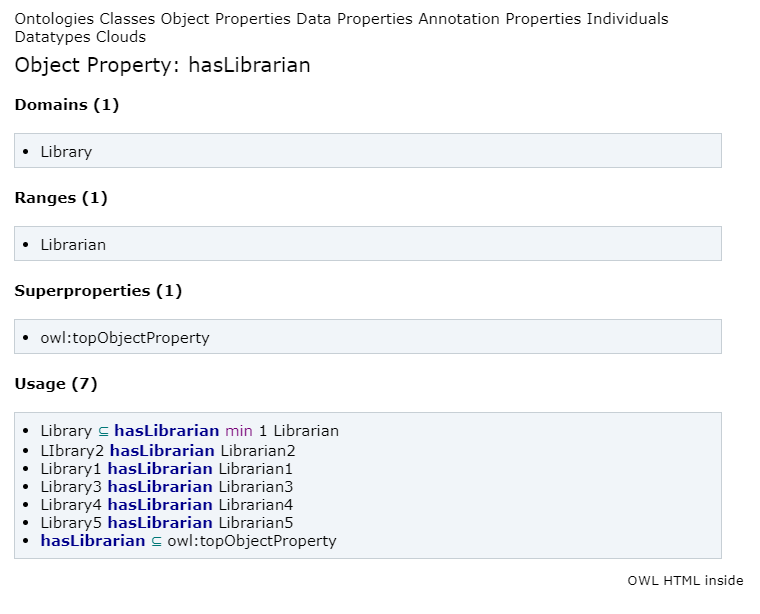


## Object Properties:

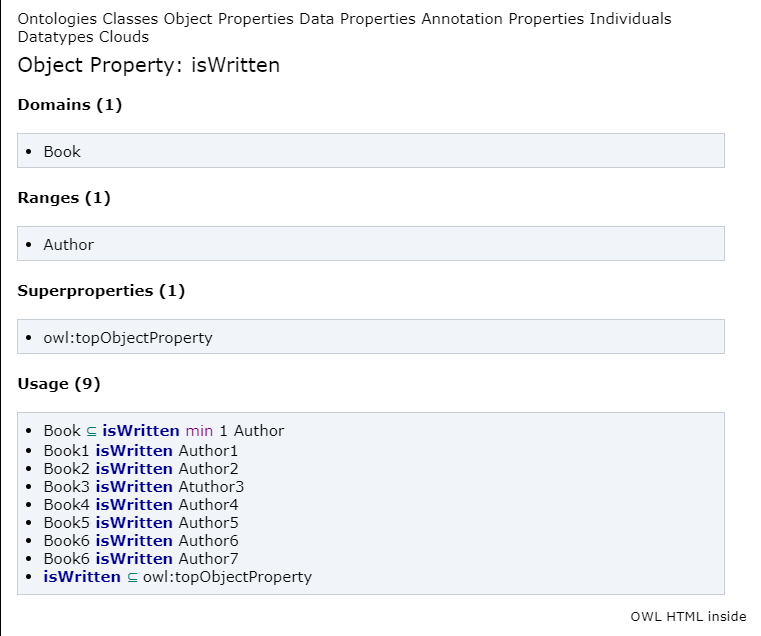
### Constraints:

* Each Book is published by 1 or more Publisher.
* Each Book is written by 1 or more Writer.
* Each Library has minimum 50 Book.
* Each Library has minimum 1 Librarian.
* Each Customer buy minimum 1 Book.
* Each Customer loan maximum 2 Book.

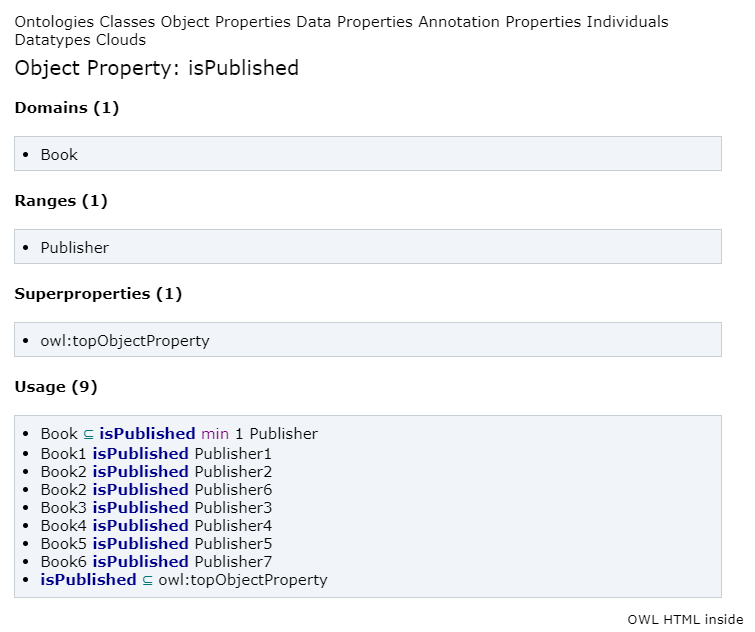
### hasLirarian:



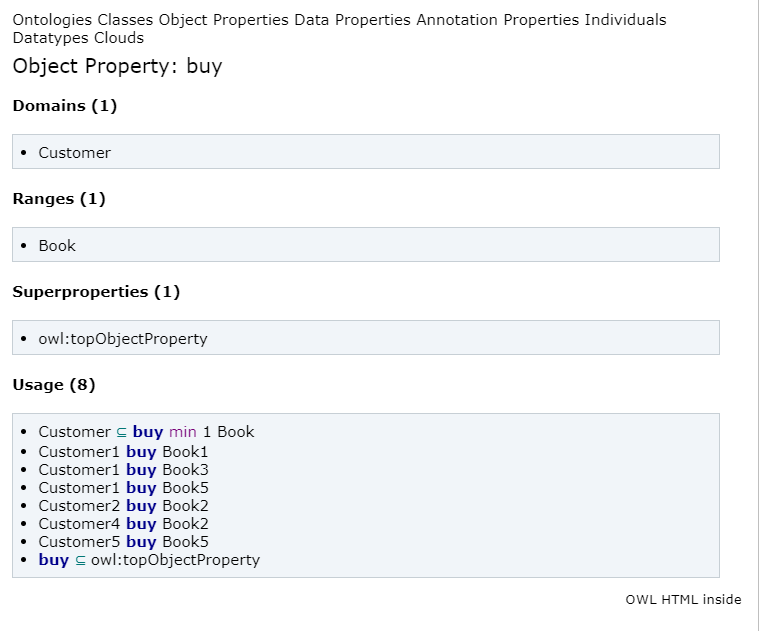
### isWritten:



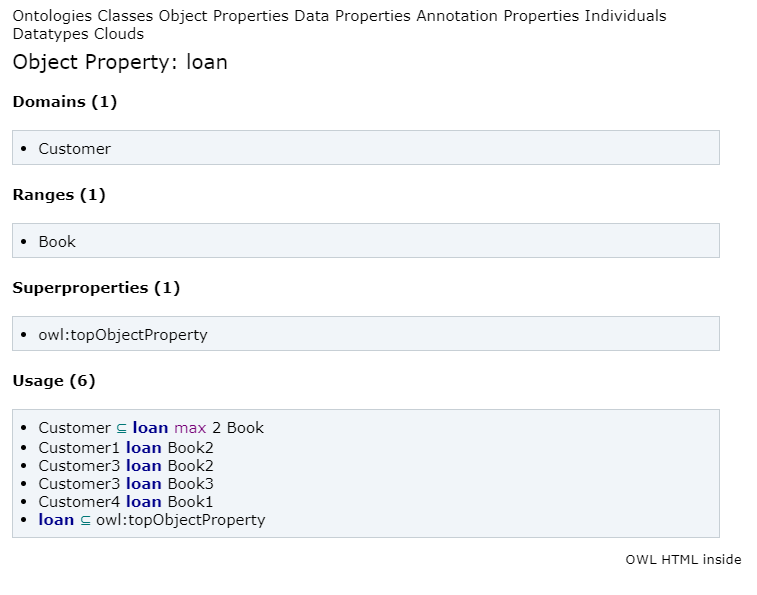
### isPublished:



### buy:



### loan:



### hasBook:

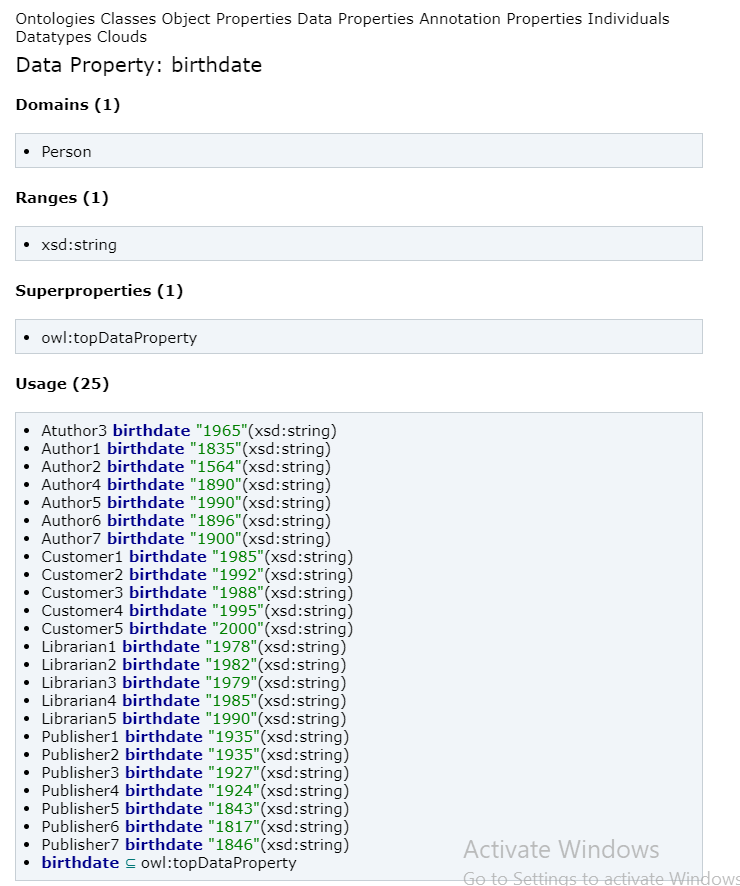


## Data Properties:

### Constraints:

* Book genre has values Drama, Romance, Crime, Science Fiction, Comedy, Poetry.
* Book 10 < Price > 1000.

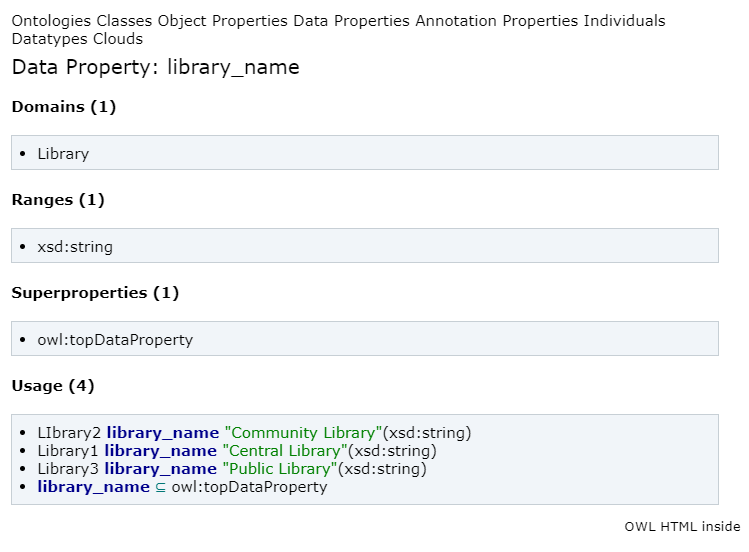
### [birthdate](file:///F:\Last%20one\dataproperties\birthdate___-766221354.html):



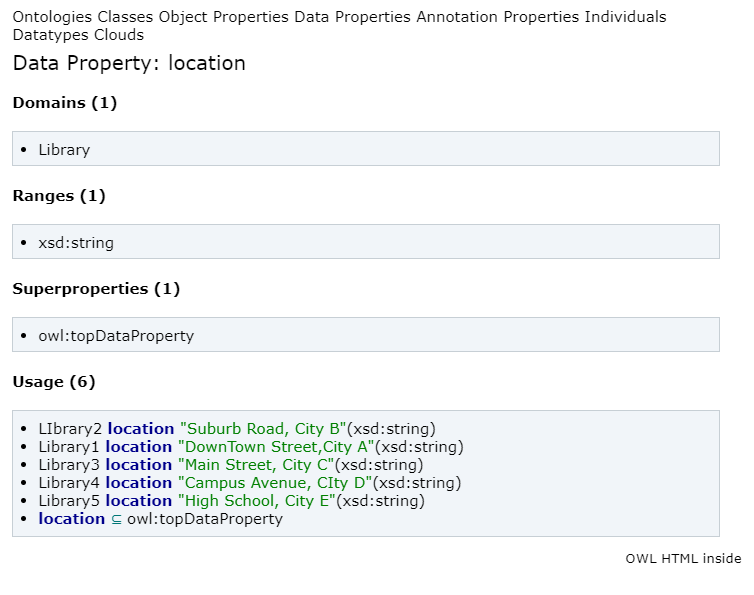
### [genre](file:///F:\Last%20one\dataproperties\genre___541098092.html):



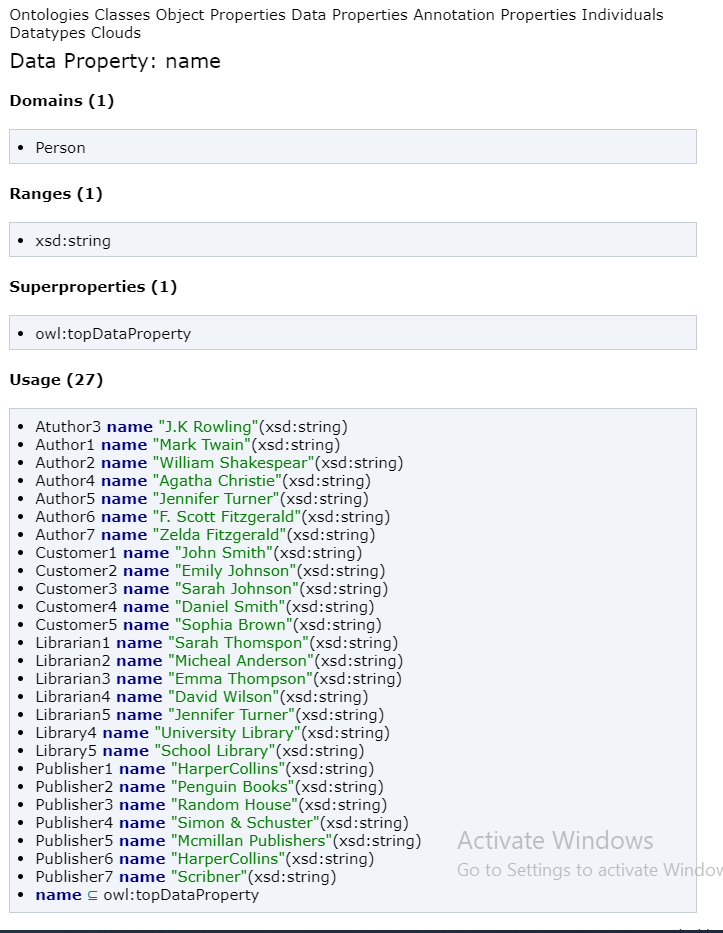
### [library\_name](file:///F:\Last%20one\dataproperties\library_name___-1266924936.html):



### [location](file:///F:\Last%20one\dataproperties\location___-1951066466.html)



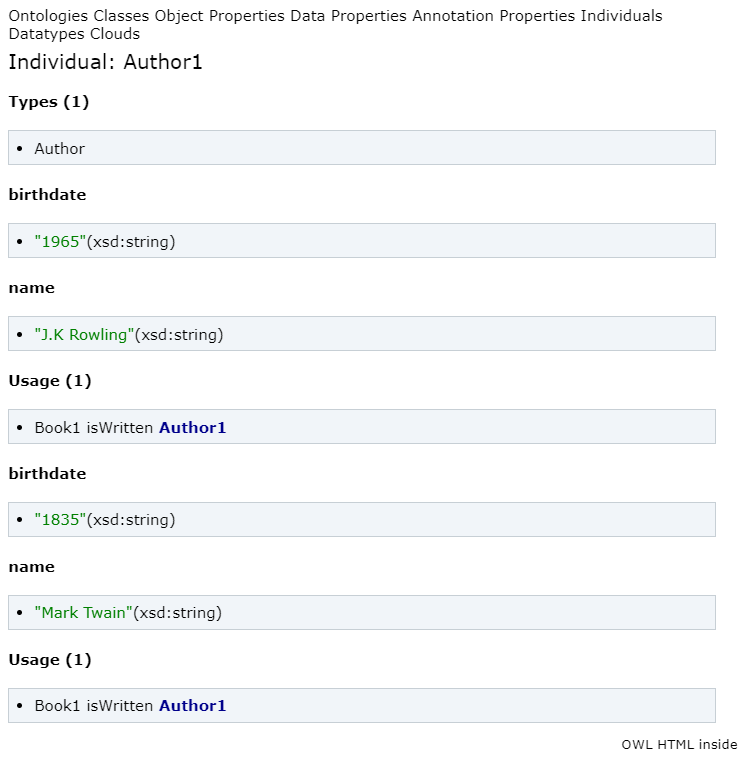
### [name](file:///F:\Last%20one\dataproperties\name___446230900.html)



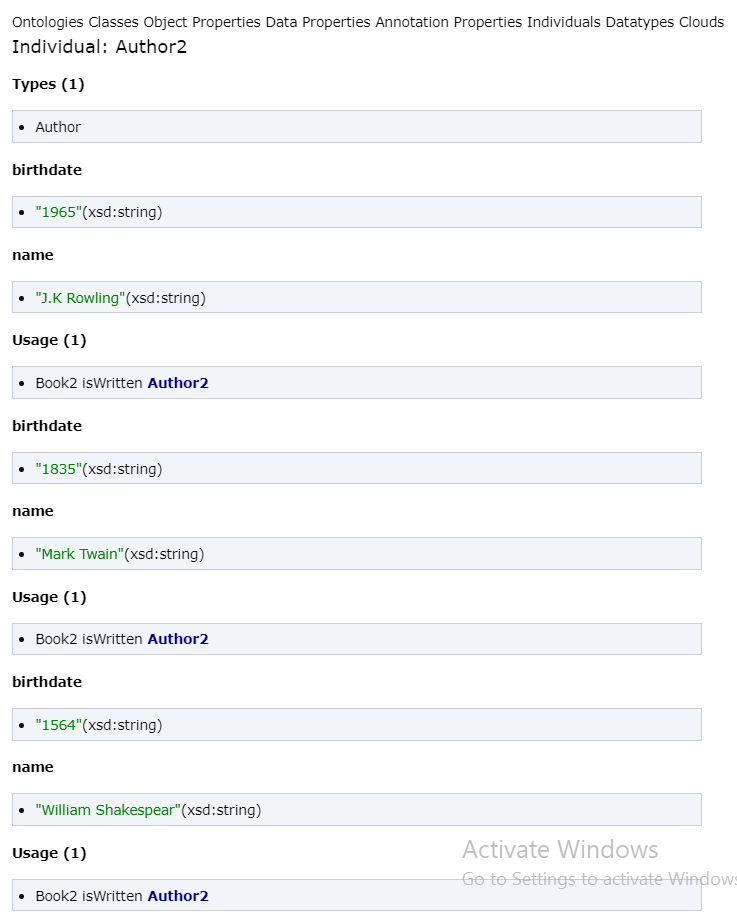
### Individuals:



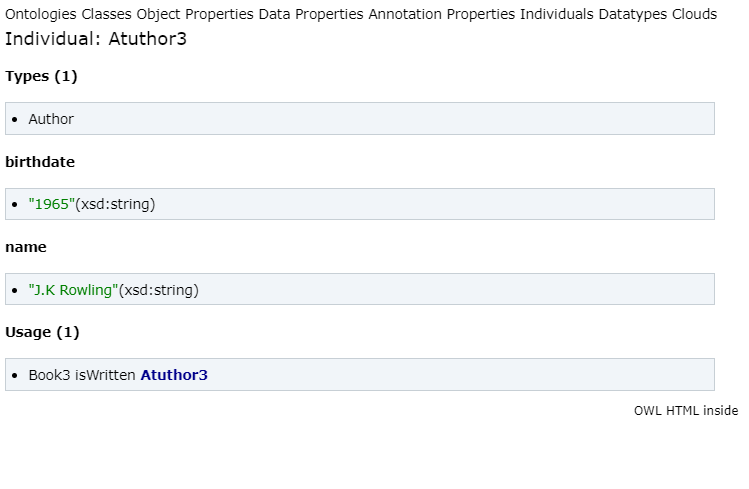
### Author 1:



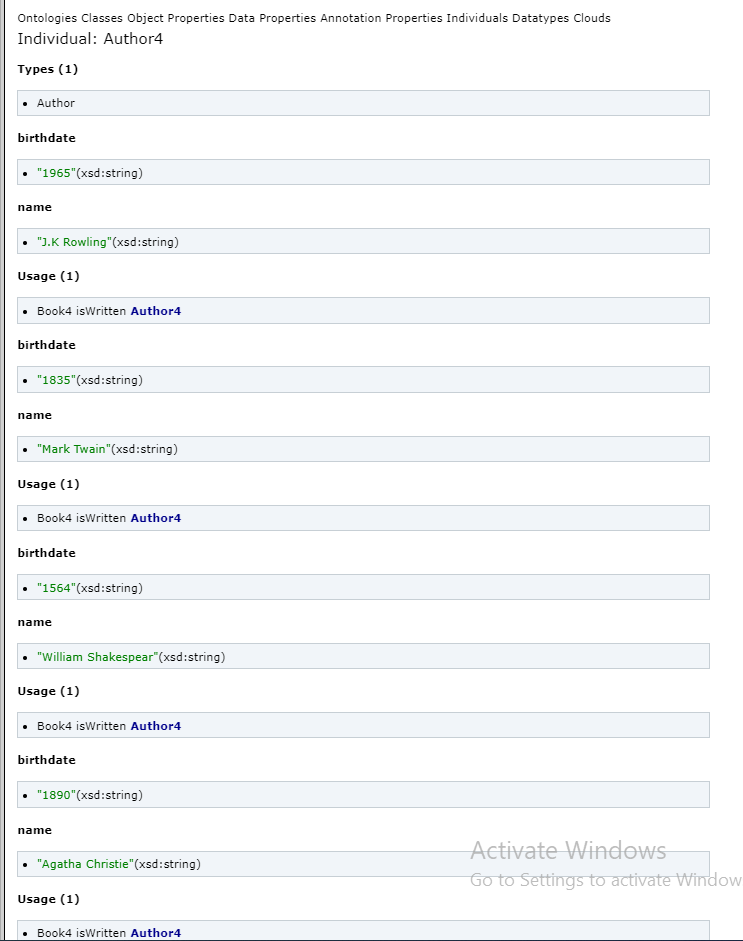
### Author 2:



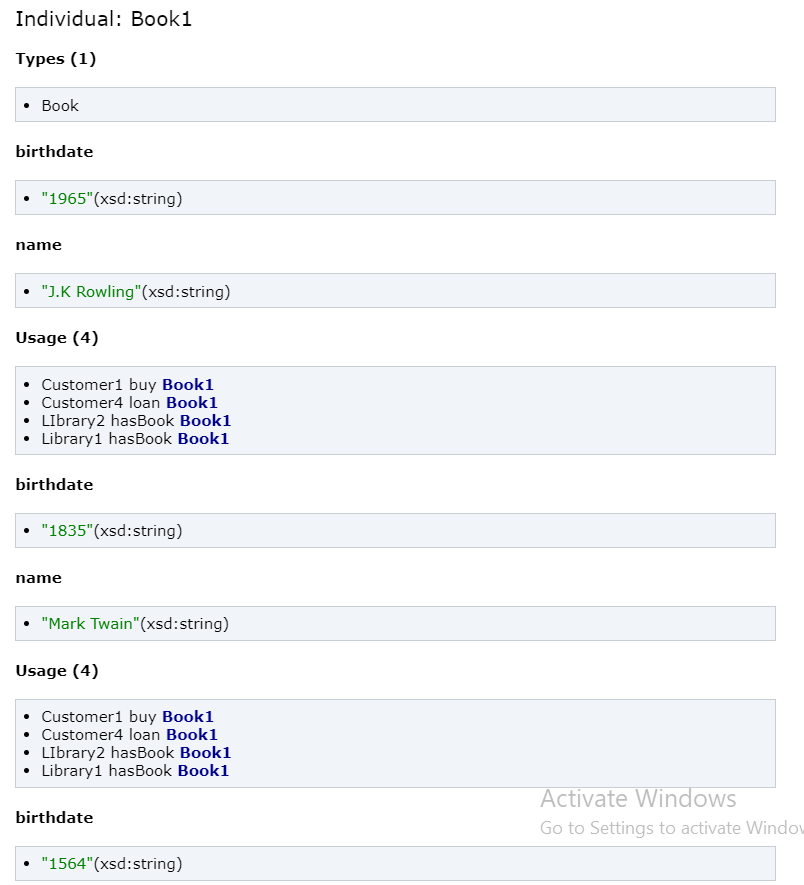
### Author 3:

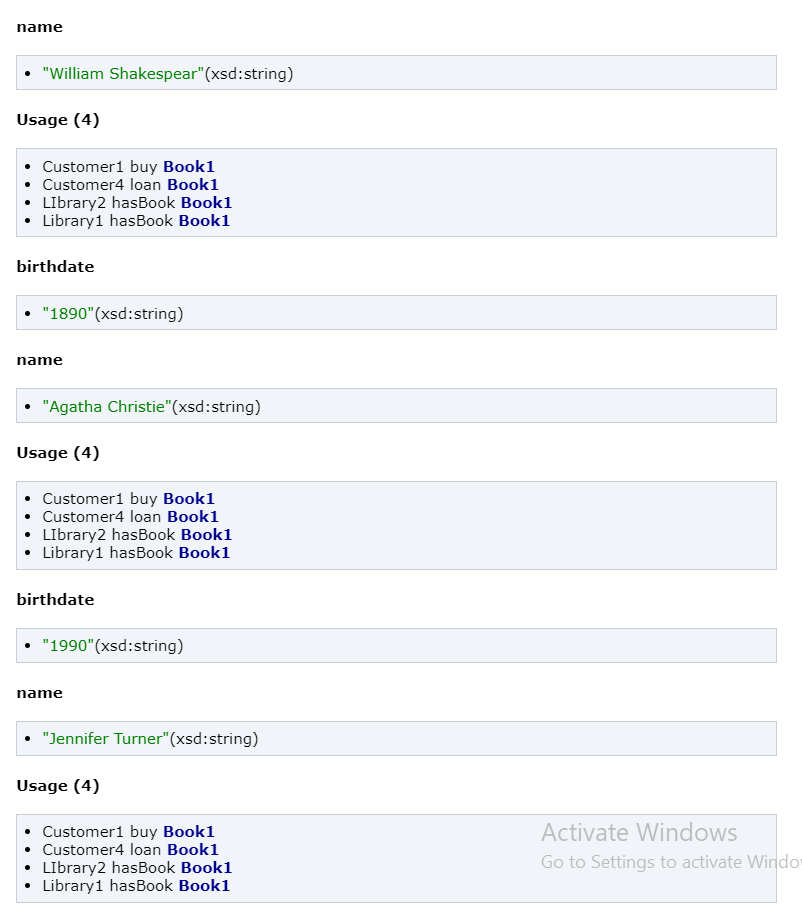


### Author 4:

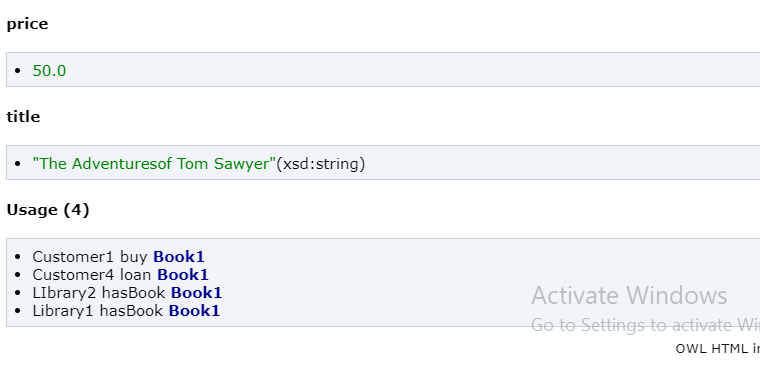


### Book1:

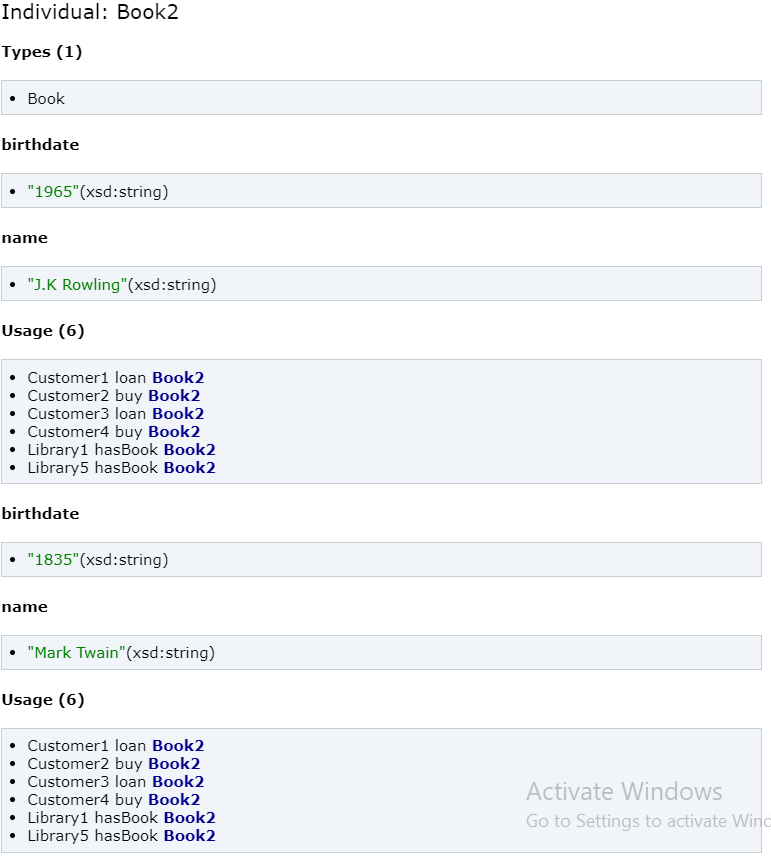


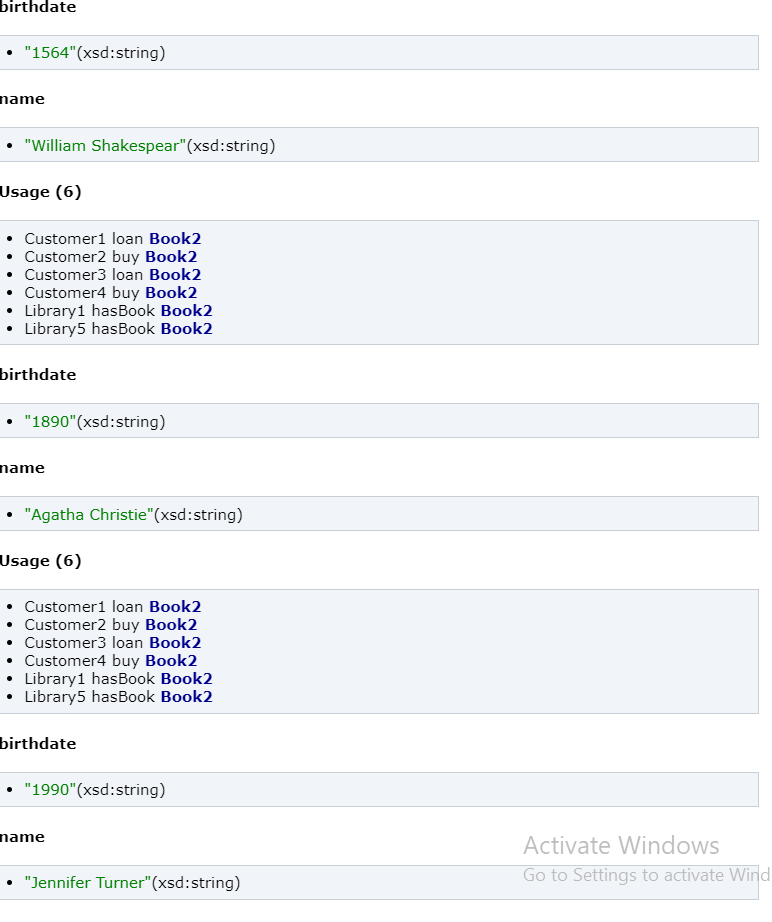




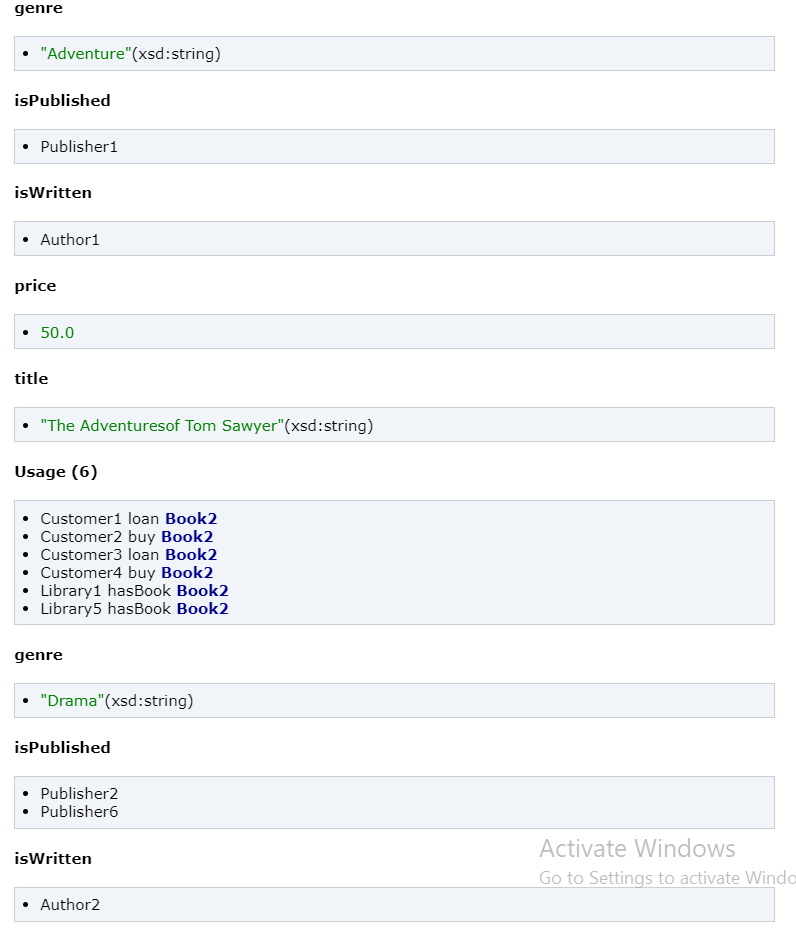


### Book2:



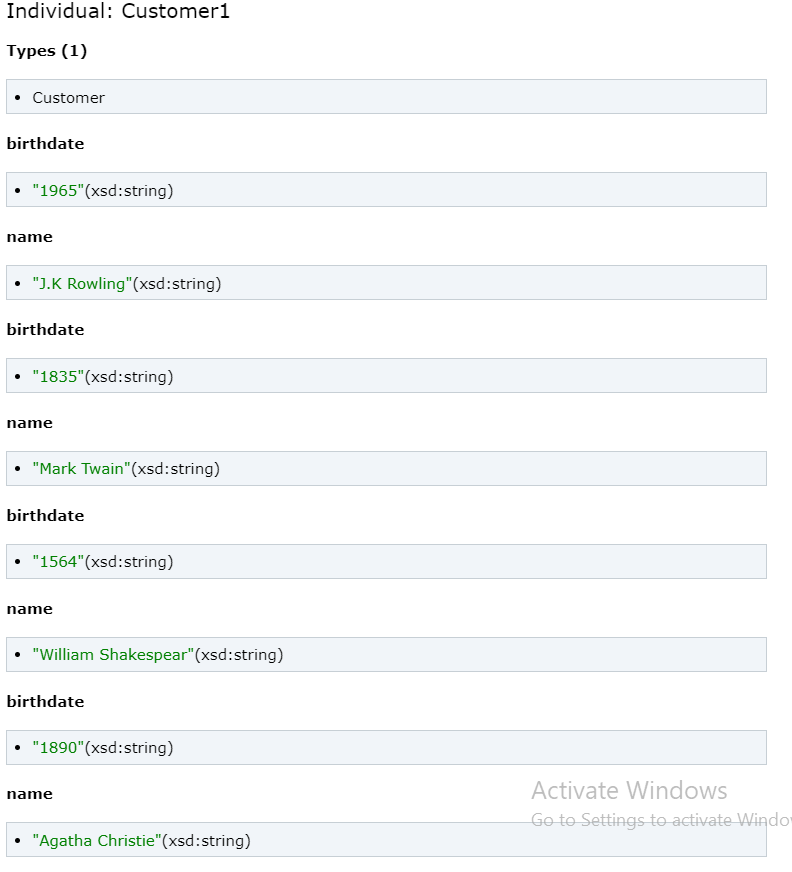


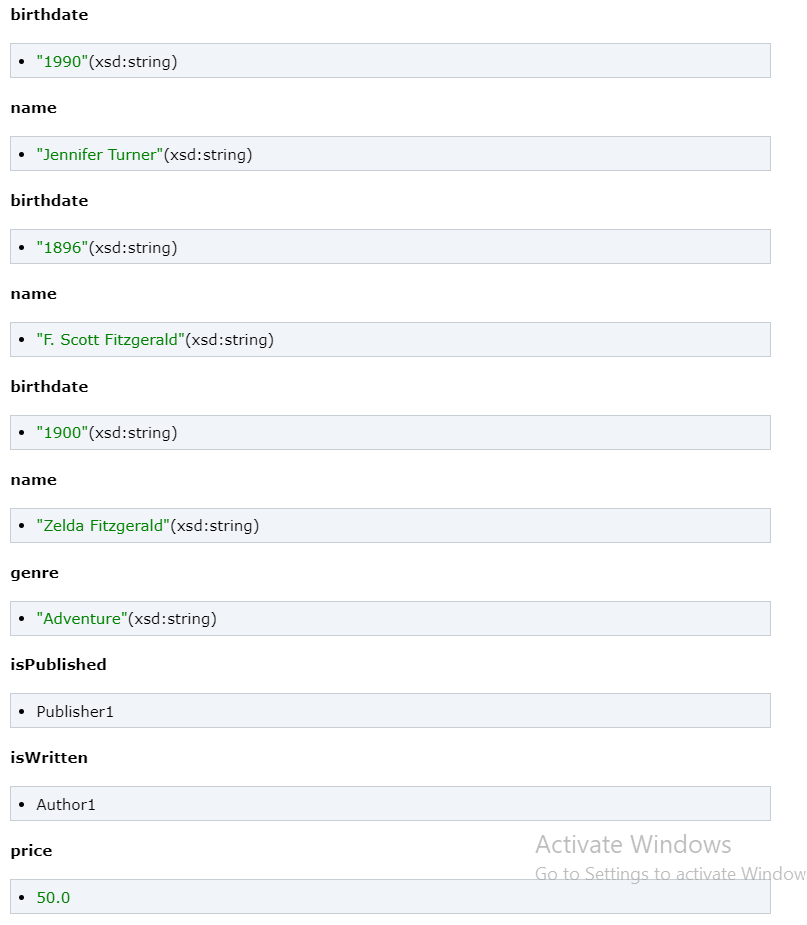


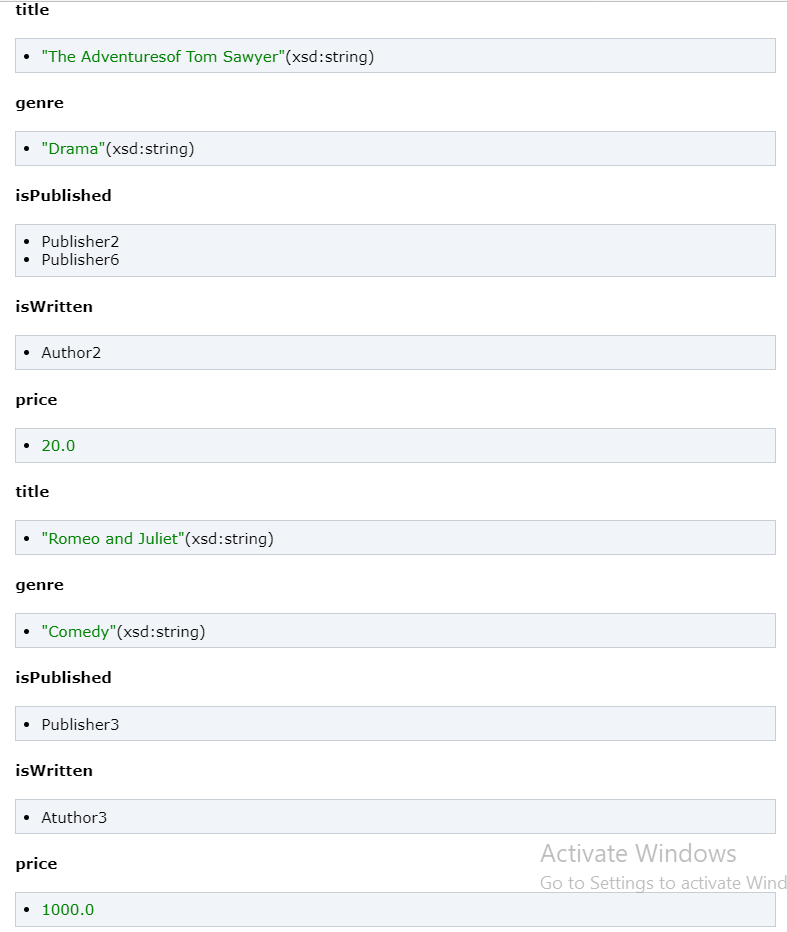


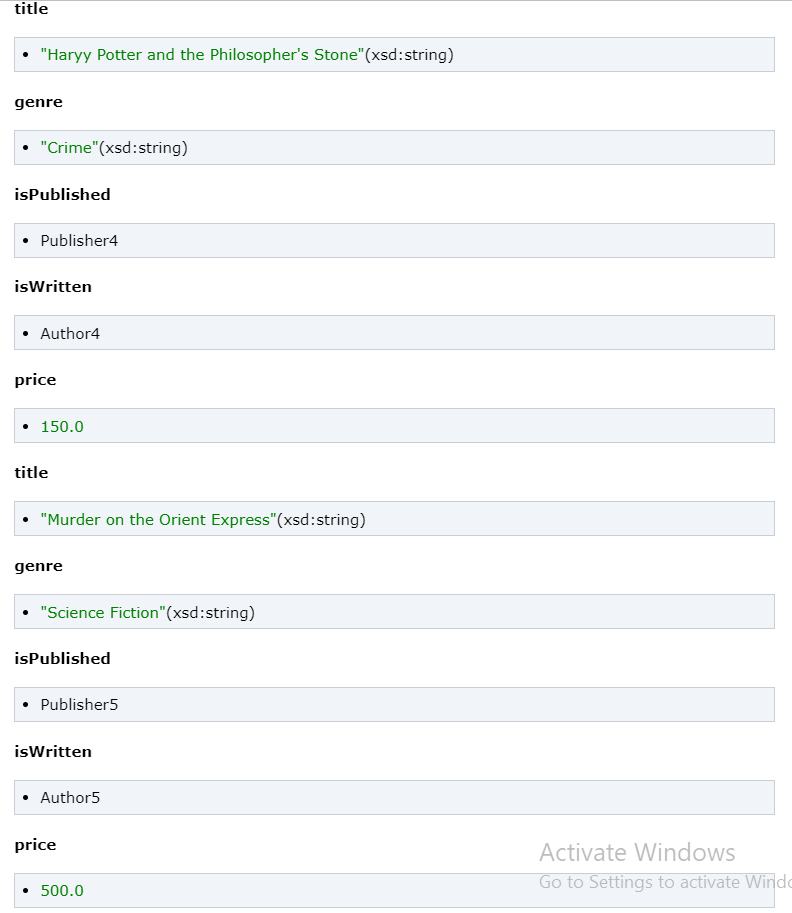


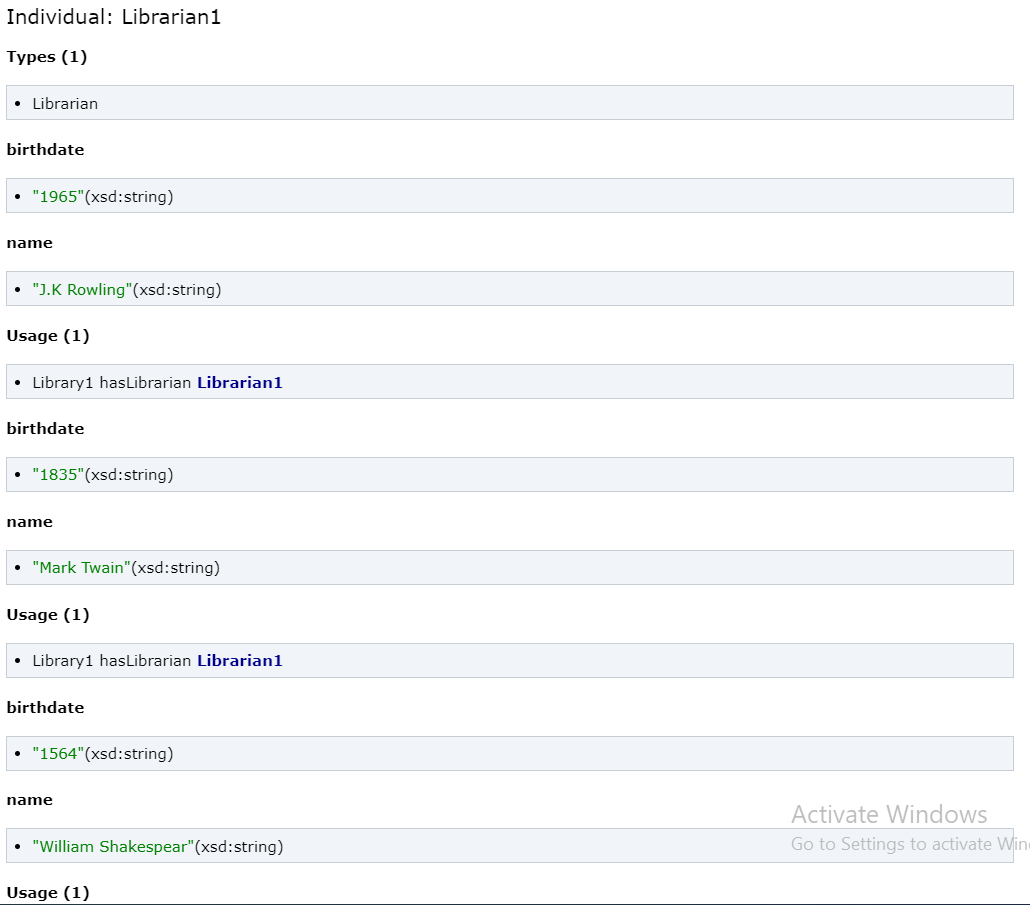
### Customer 1:

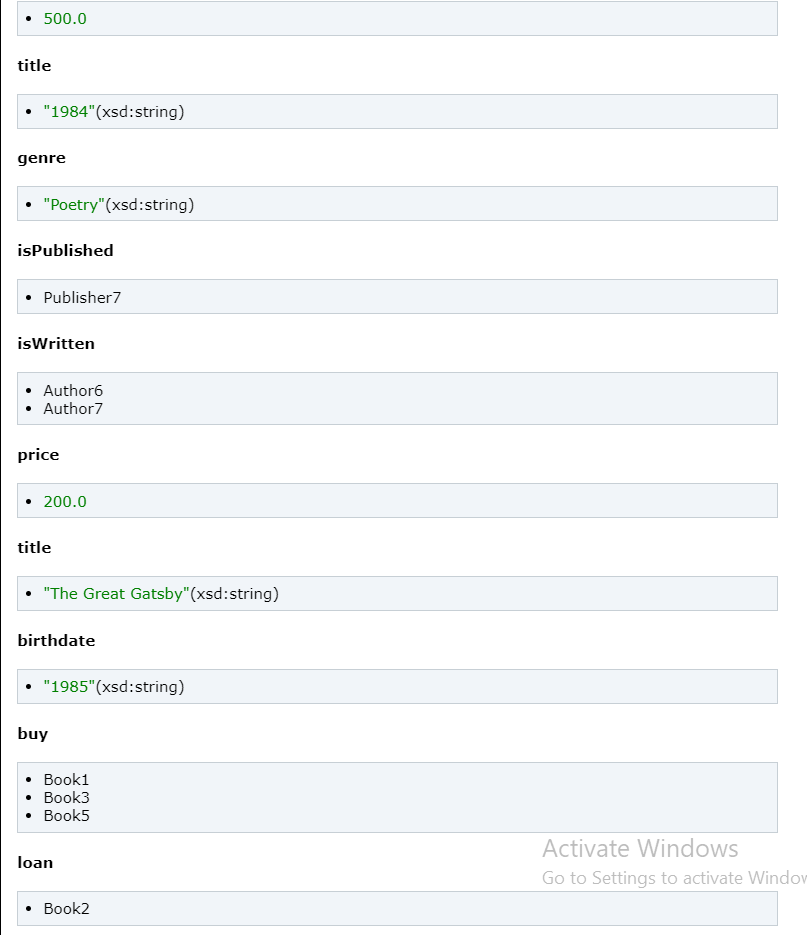




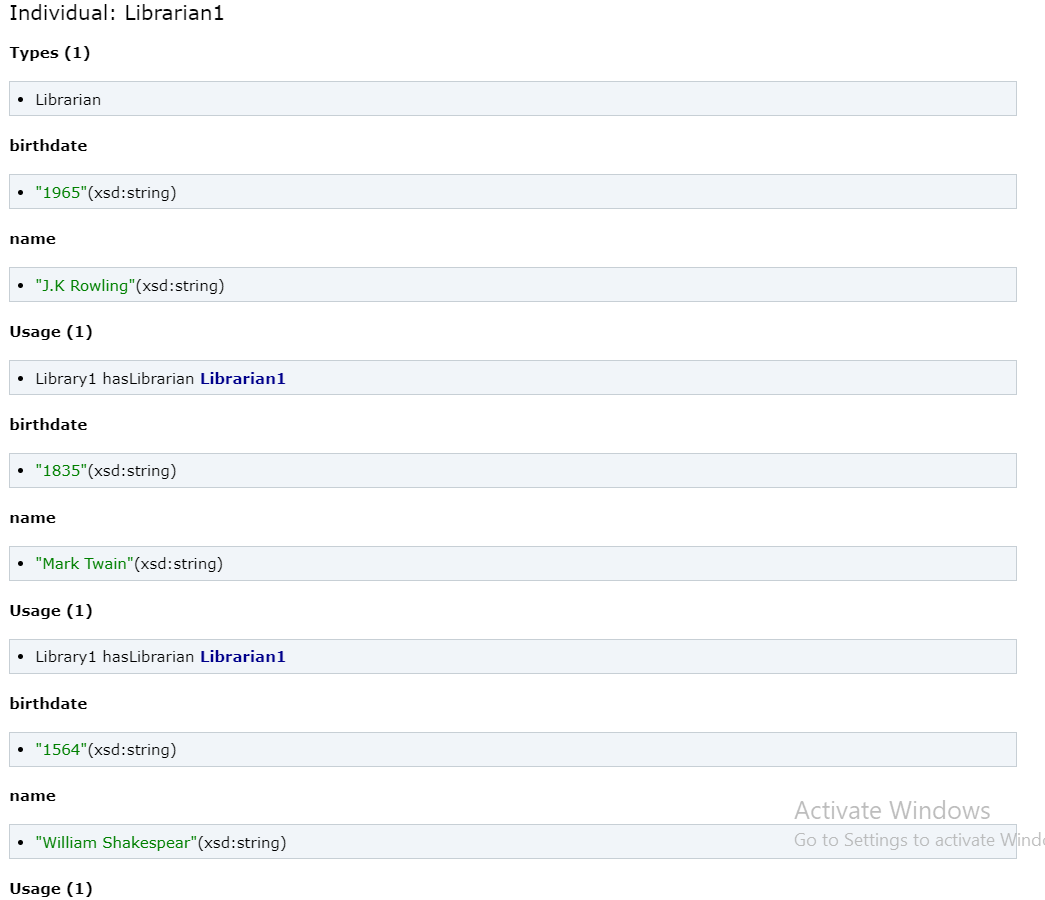


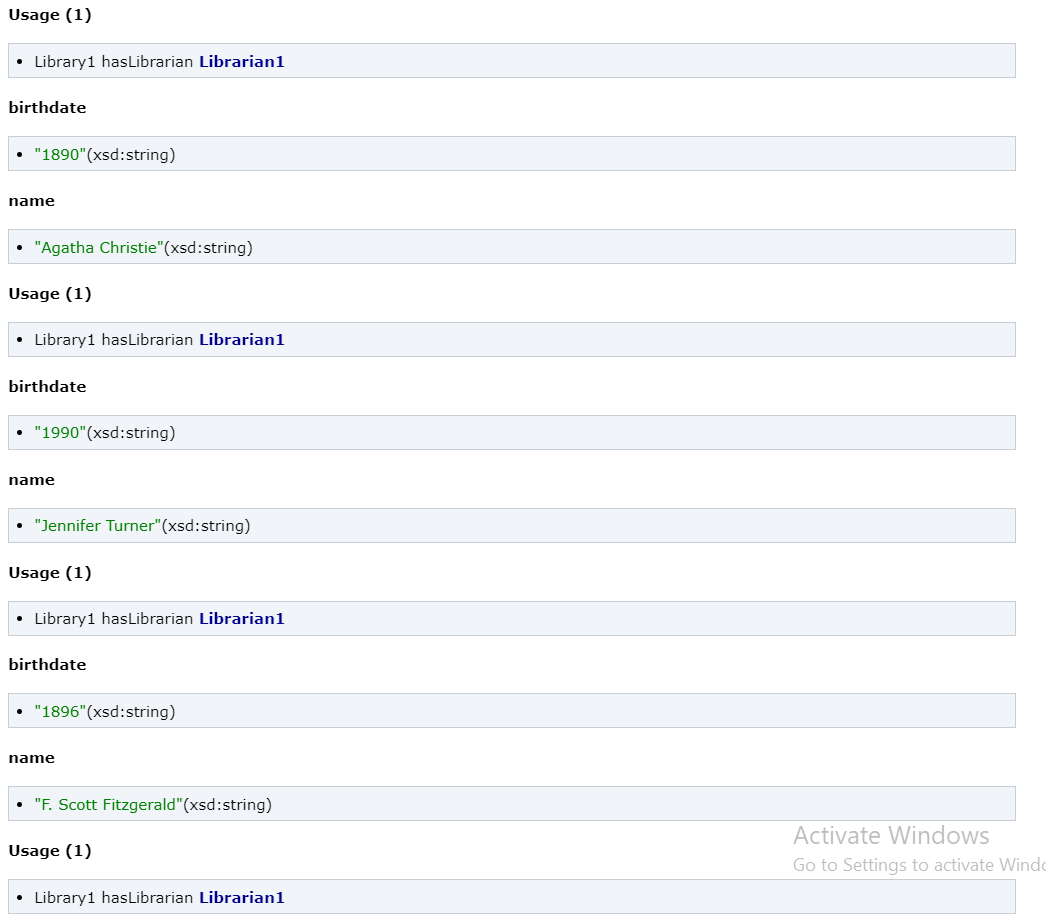


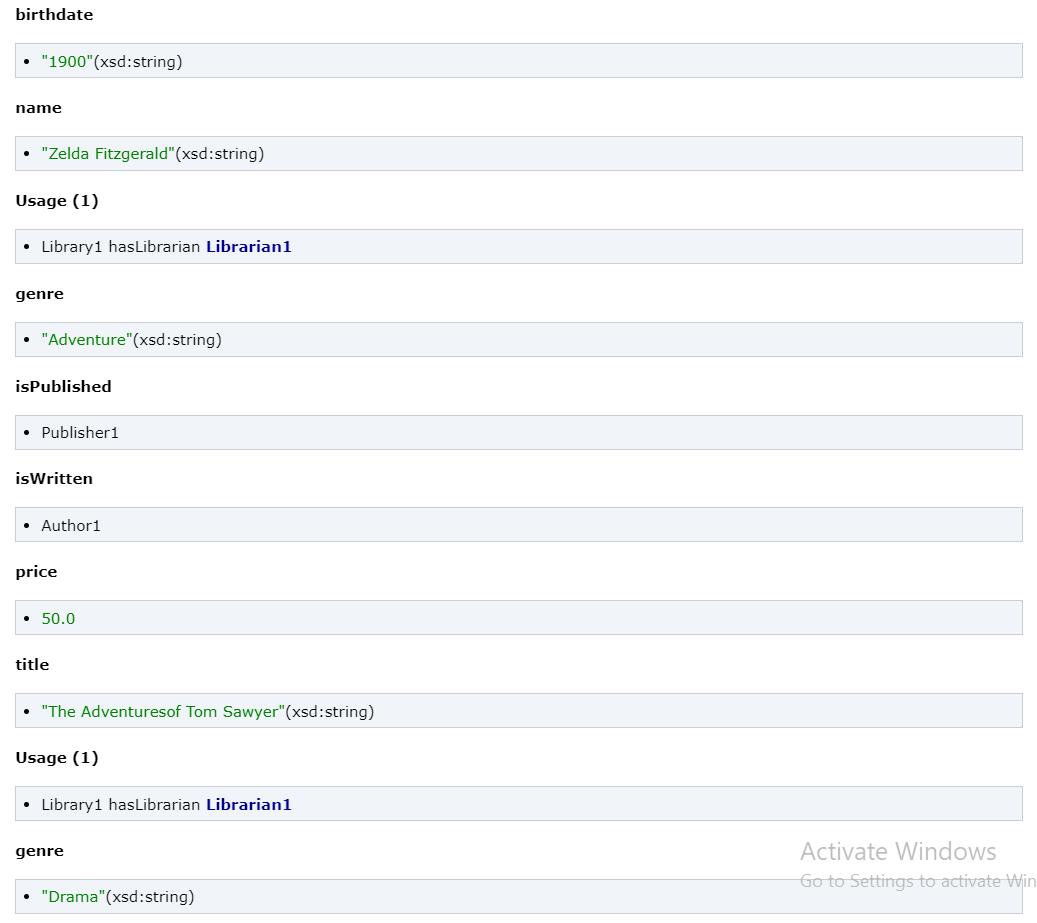


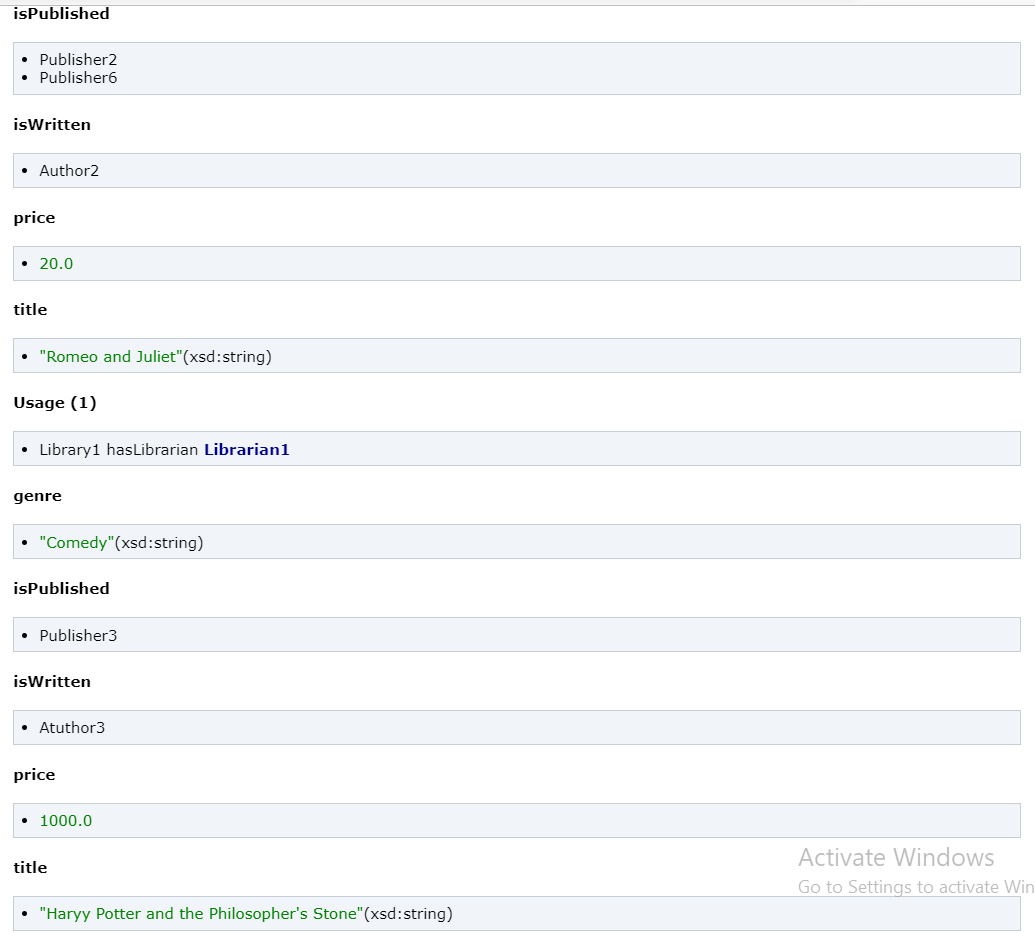


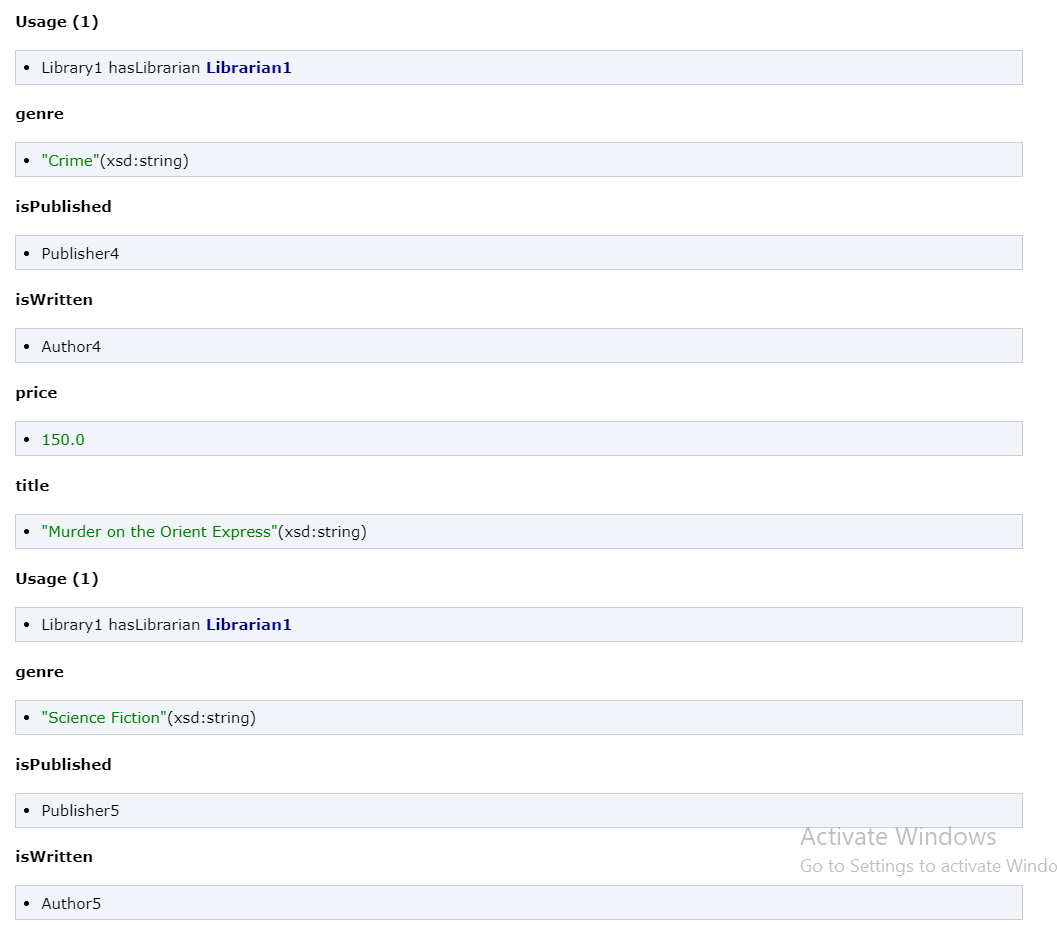
### Librarian1:

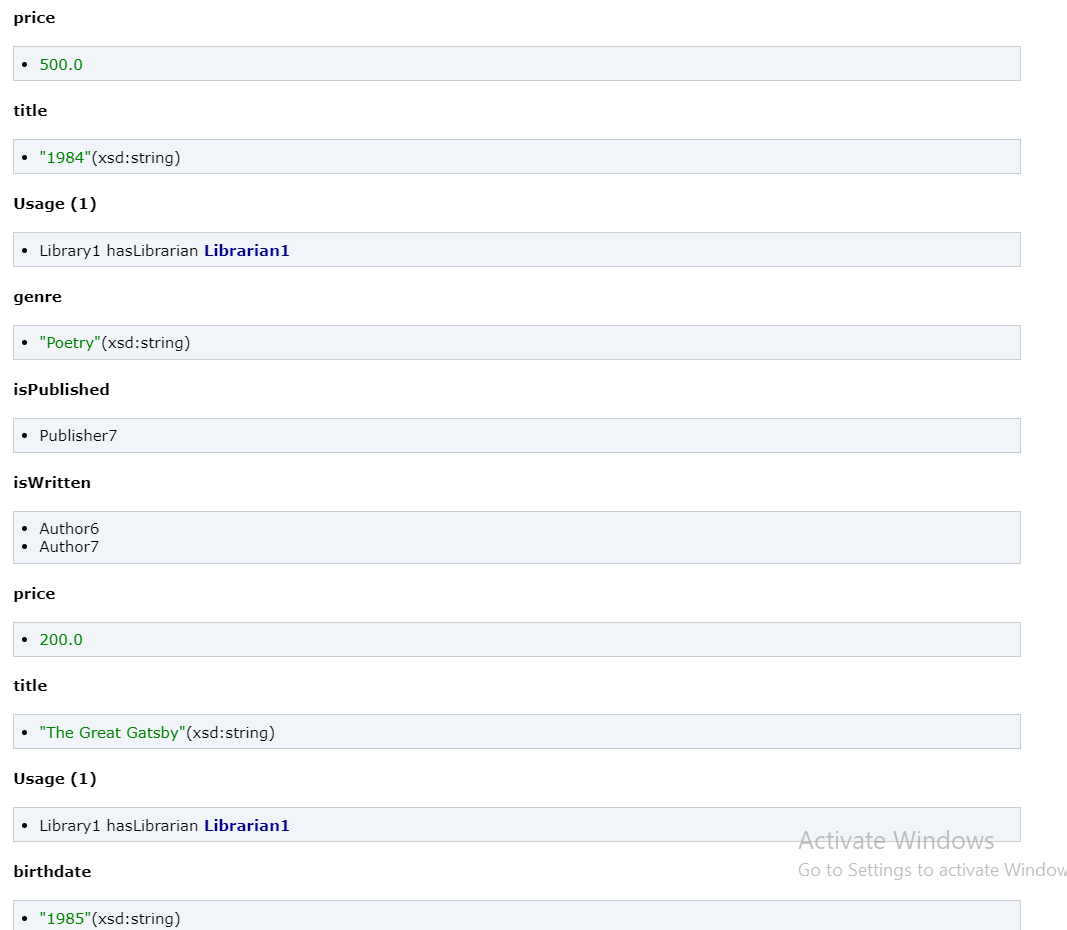




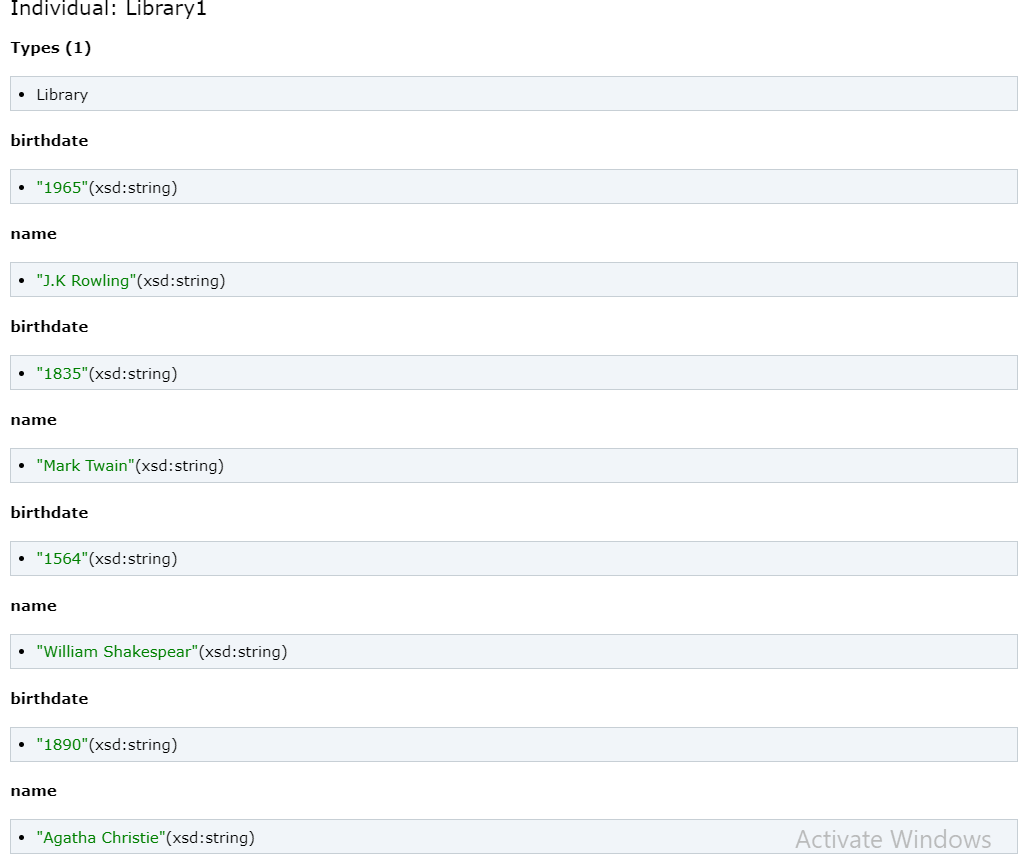




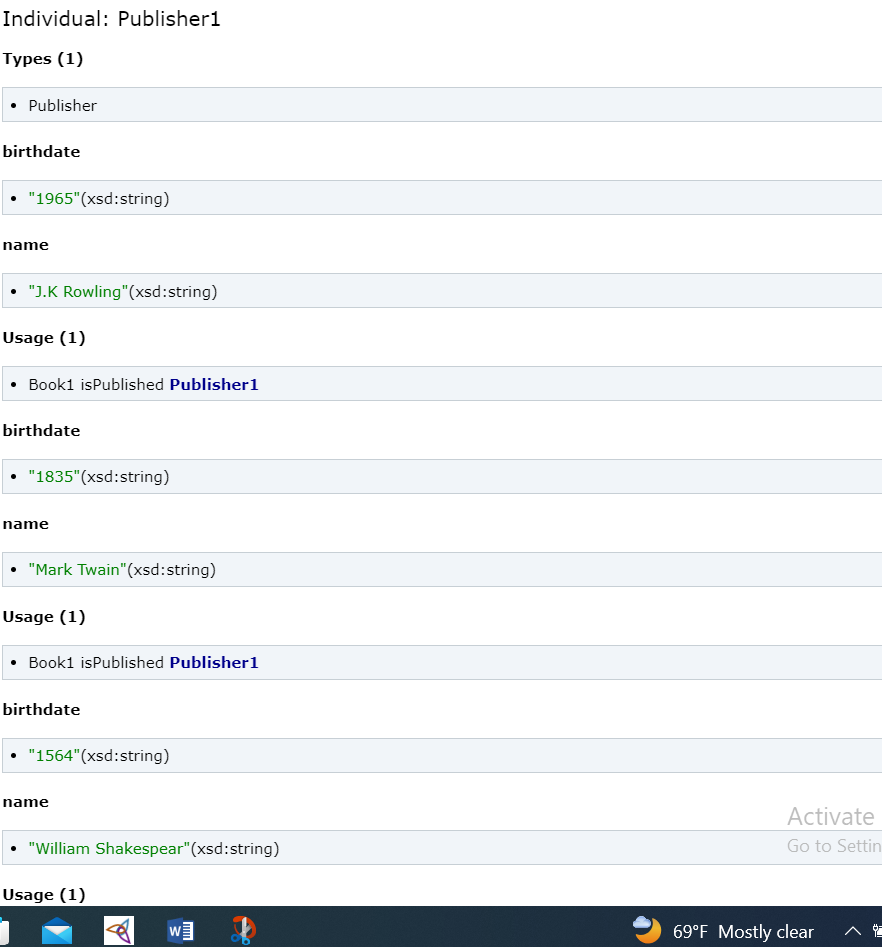




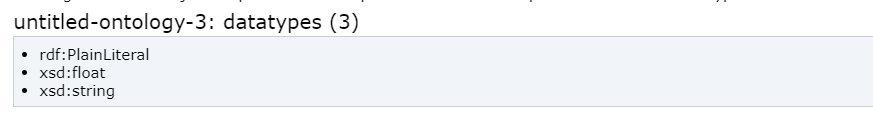
### Library 1:

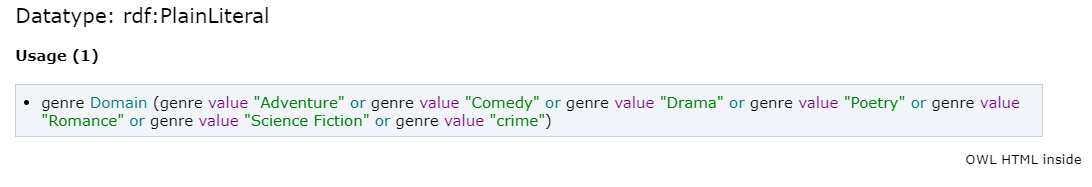


### Publisher 1:



## Datatypes:



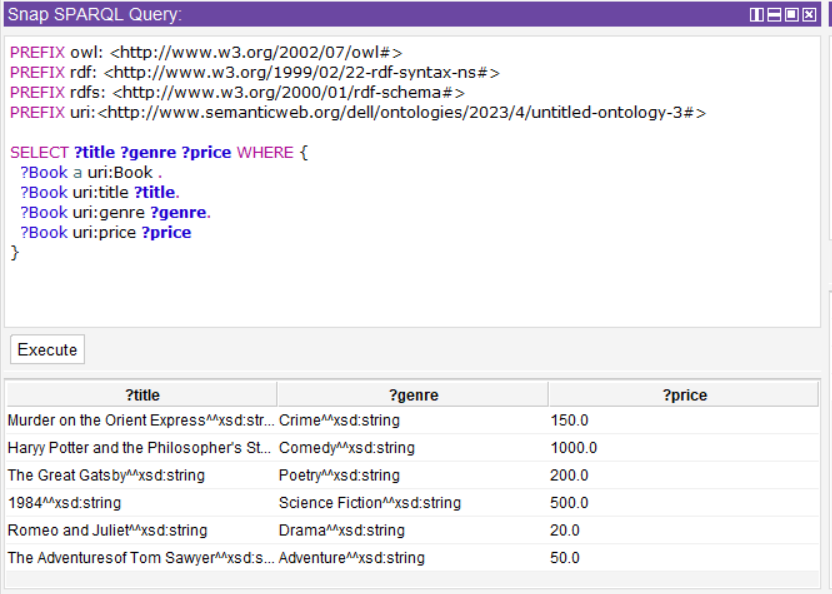




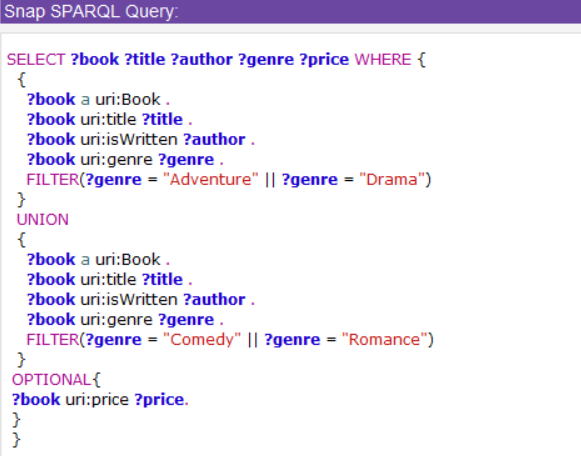


## Sparql queries:

Get the title, genre and price of all books.

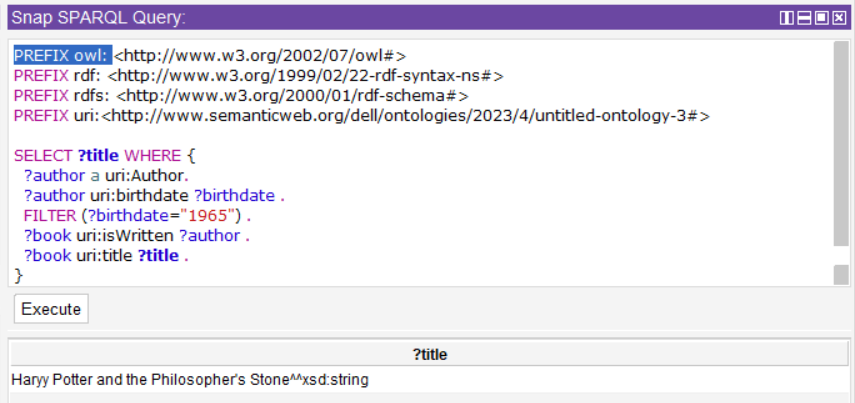


Query that includes two alternatives (using the UNION operator) and two conjunctions (using the FILTER and OPTIONAL clauses). This query retrieves books along with their titles, authors, and genres. It includes two alternatives where the genre can be either "Adventure" or "Drama" in one group, and "Comedy" or "Romance" in the other group. It also uses the FILTER clause to apply the genre constraints. Additionally, it uses the OPTIONAL clause to retrieve the optional property "cost" if available for each book.

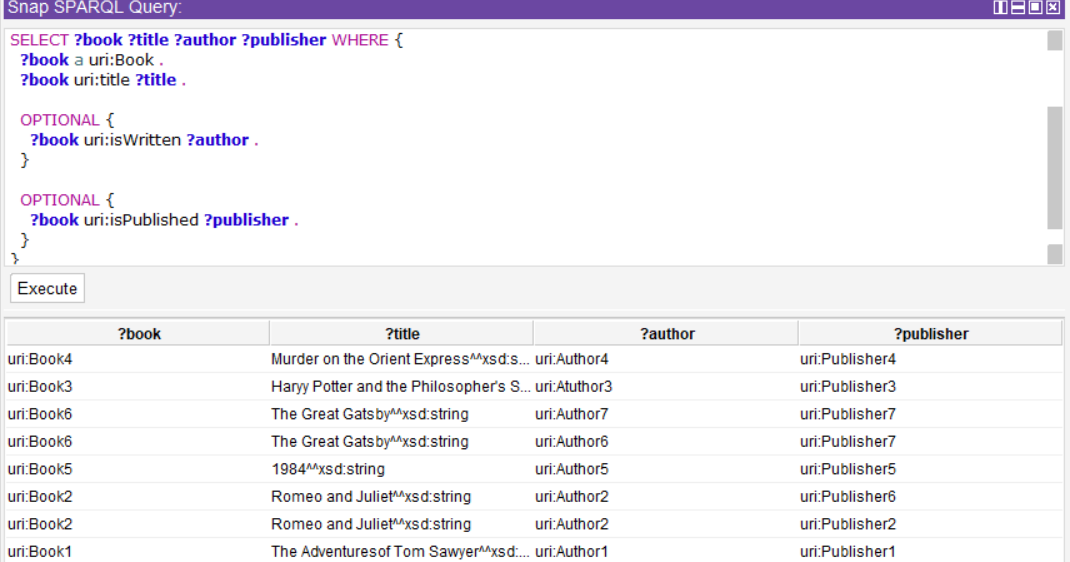




Get the title of the book for which the author's birthdate is 1965.



This query includes 2 Optional. Get books along with their titles.



## User Interface:

* The code begins by importing the necessary modules: Flask for creating the web application, render\_template for rendering HTML templates, request for handling HTTP requests, and Graph from rdflib for working with RDF graphs.
* An instance of the Flask application is created.
* The index function is defined and decorated with @app.route('/'), which means it is the handler for the root URL of the application.
* Inside the index function, the code checks if the request method is POST. If it is, it retrieves the SPARQL query entered by the user from the form data and calls the execute\_sparql\_query function to execute the query on the OWL file. The results are then passed to the 'index.html' template for rendering.
* If the request method is not POST (i.e., a GET request), the 'index.html' template is rendered without any results.
* The execute\_sparql\_query function is defined to handle the execution of SPARQL queries. It loads the OWL file using the rdflib Graph class and parses it in XML format.
* The SPARQL query passed as an argument to the function is executed on the loaded graph using the query method, and the results are stored in the results variable.
* The results are then converted to a list of dictionaries, where each dictionary represents a row of the query results. The bindings attribute of the results object contains the individual query results, and the vars attribute contains the variables of the query.
* Finally, the rows list is returned, containing the query results in a format suitable for rendering in the template

from flask import Flask, render\_template, request

from rdflib import Graph

app = Flask(\_\_name\_\_)

@app.route('/', methods=['GET', 'POST'])

def index():

    if request.method == 'POST':

        query = request.form['query']

        results = execute\_sparql\_query(query)

        return render\_template('index.html', results=results)

    return render\_template('index.html')

def execute\_sparql\_query(query):

    # Load the OWL file

    g = Graph()

    g.parse('LibrarySystemProject.owl', format='xml')

    # Execute the SPARQL query

    results = g.query(query)

    print(results)

    # Convert the results to a list of dictionaries

    bindings = results.bindings

    rows = []

    for binding in bindings:

        row = {}

        for var in results.vars:

            row[var] = binding[var].toPython()

        rows.append(row)

    return rows

if \_\_name\_\_ == '\_\_main\_\_':

    app.run()

<!DOCTYPE html>

<html>

<head>

    <title>SPARQL Query</title>

</head>

<body>

    <h1>SPARQL Query</h1>

    <form action="/" method="POST">

        <label for="query">Enter SPARQL query:</label>

        <input type="text" width = "50dp" name="query" id="query" required>

        <button type="submit">Execute</button>

    </form>

    {% if results %}

    <h2>Results:</h2>

    <table>

        <tr>

            {% for var in results[0].keys() %}

            <th>{{ var }}</th>

            {% endfor %}

        </tr>

        {% for row in results %}

        <tr>

            {% for value in row.values() %}

            <td>{{ value }}</td>

            {% endfor %}

        </tr>

        {% endfor %}

    </table>

    {% endif %}

</body>

</html>

## Screenshots:

the SPARQL query retrieves the count of instances of the ont:Book class

A screenshot of a computer

Description automatically generated with low confidence

Retrieve all instances of the class "Library" that has books and a librarian

SELECT ?library

WHERE {

?library rdf:type <http://www.semanticweb.org/dell/ontologies/2023/4/untitled-ontology-3#Library>.

?library <http://www.semanticweb.org/dell/ontologies/2023/4/untitled-ontology-3#hasBook> ?book.

?library <http://www.semanticweb.org/dell/ontologies/2023/4/untitled-ontology-3#hasLibrarian> ?librarian.

}A screenshot of a computer

Description automatically generated with low confidence

A picture containing text, screenshot, font, receipt

Description automatically generatedA screenshot of a computer

Description automatically generated with low confidenceA picture containing text, font, letter, document

Description automatically generatedA screenshot of a computer

Description automatically generated with low confidence

## Github:

<https://github.com/IlariaRefaat/Project_Ontologies_And_Semantic_web>