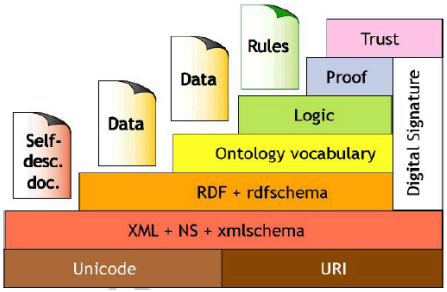
SWT TT1

QB

1. Explain in detail about the layered architecture of semantic web stack.

- The development of the Semantic Web proceeds in steps
- Each step building a layer on top of another
- Principles: Downward compatibility and Upward partial understanding



Semantic Web layers:

- XML layer
  - Syntactic basis
- RDF layer
  - RDF basic data model for facts
  - RDF Schema simple ontology language
- Ontology layer
  - More expressive languages than RDF Schema
  - Current Web standard: OWL
- Logic layer
  - enhance ontology languages further
  - application-specific declarative knowledge
- Proof layer
  - Proof generation, exchange, validation
- Trust layer
  - Digital signatures
  - recommendations, rating agencies ….

2. Discuss the problems associated with keyword-based search engines.

Problems:

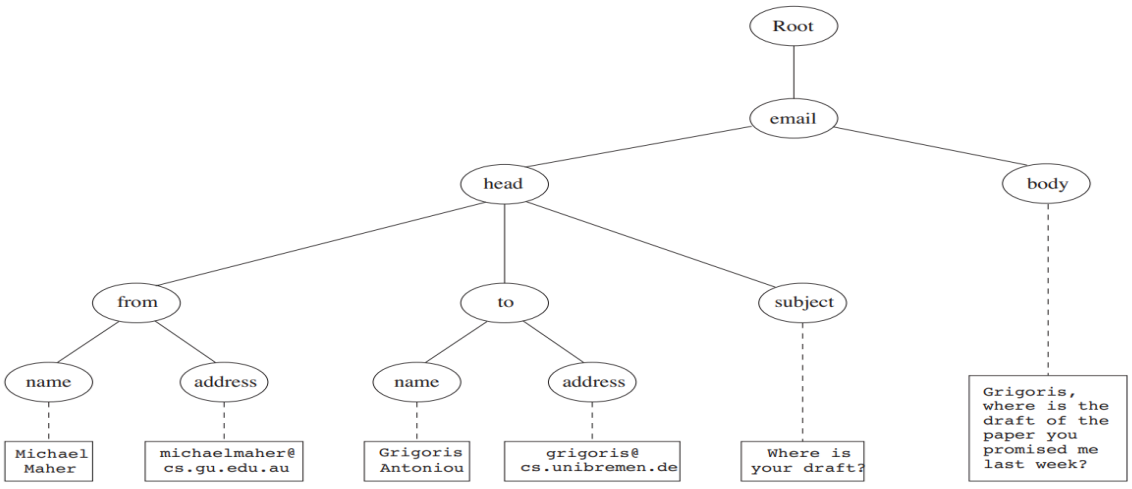
- High recall, low precision.

- Low or no recall
- Results are highly sensitive to vocabulary
- Results are single Web pages
- Human involvement is necessary to interpret and combine results
- Results of Web searches are not readily accessible by other software tools

3. Explain with example tree model in XML.

The tree representation of an XML document is an ordered labeled tree:

- There is exactly one root
- There are no cycles
- Each non-root node has exactly one parent
- Each node has a label.
- The order of elements is important
- But the order of attributes is not important




```
<email>
        <head>
                <from name="Michael Maher"
                        address="michaelmaher@cs.gu.edu.au"/>
                <to name="Grigoris Antoniou"
                        address="grigoris@cs.unibremen.de"/>
                <subject>Where is your draft?</subject>
        </head>
        <body>
                Grigoris, where is the draft of the paper you promised me
                last week?
```

```
    </body>
</email>
```

5. Explain the following RDF(S) core classes or properties: i. rdfs:Resource ii.rdfs:domain iii. rdfs:range iv. rdf:Property v. rdf:type vi. rdfs:label
Assume XSD properties take values of type integer.

- RDF is the basic building block for supporting the Semantic Web.
- RDF is to the Semantic Web what HTML has been to the Web.
- RDF is an XML-based language for describing information contained in a Web resource.
- A Web resource can be a Web page, an entire Web site, or any item on the Web that contains information in some form.

RDF Core Classes -

i. rdfs:Resource:
This is the most general class in RDF(S). It serves as the superclass of all other RDF classes. Every RDF instance is an instance of rdfs:Resource. It's used to denote anything that can be identified by a URI (Uniform Resource Identifier).

ii. rdfs:domain:
This property is used to specify the domain of a property. In other words, it indicates the class of resources to which a property can be applied. For example, if you have a property "hasFather" and you specify its domain as "Person", it means that "hasFather" can only be applied to instances of the class "Person".

iii. rdfs:range:
This property is used to specify the range of a property. It indicates the class of resources that can be the object of a triple whose predicate is the specified property. For example, if you have a property "hasAge" and you specify its range as "xsd:integer", it means that the object of the triple using "hasAge" must be an integer.

iv. rdf:Property:
This is the class of properties in RDF. It is used to represent properties that can be used to relate resources to other resources. Properties such as "hasFather", "hasAge", etc., are instances of rdf:Property.

v. rdf:type:

This property is used to assert that a resource is an instance of a particular class. For example, if you have a resource representing a person and you assert that it is of type "Person" using rdf:type, it means that the resource belongs to the class "Person".

vi. rdfs:label:
This property is used to provide a human-readable version of the name of a resource. It's often used for displaying purposes or to provide additional information about the resource. For example, you might have a resource representing a book and you can use rdfs:label to provide the title of the book in a human-readable format.

6. Explain semantic web technologies and how Semantic Web agents will make use of all the technologies [Imp]

Semantic Web technologies refer to a set of standards and protocols designed to enable machines to understand the meaning (semantics) of information on the World Wide Web.

- Resource Description Framework (RDF):
    - RDF provides a standard model for data interchange on the web. It represents information in the form of subject-predicate-object triples.
    - Semantic Web agents will use RDF to describe resources and their relationships in a machine-understandable format.

- RDF Schema (RDFS):
    - RDFS provides a basic vocabulary for describing classes and properties of RDF resources.
    - Semantic Web agents will use RDFS to define the structure of RDF vocabularies, specifying classes, properties, and relationships between them.

- Web Ontology Language (OWL):
    - OWL is a more expressive language for defining ontologies, which represent the formal semantics of a domain.
    - Semantic Web agents will use OWL to define rich, complex vocabularies and ontologies, enabling more precise and detailed modeling of domains.

- SPARQL Protocol and RDF Query Language (SPARQL):
    - SPARQL is used to query RDF data, providing a standardized query language and protocol for accessing semantic data.

    - Semantic Web agents will use SPARQL to retrieve information from RDF datasets, enabling them to answer complex queries and extract knowledge from distributed sources.

- Linked Data:
    - Linked Data principles advocate for publishing structured data on the web using standardized formats and linking it to other related data.
    - Semantic Web agents will utilize linked data to navigate and discover relevant information across distributed datasets, leveraging interconnectedness to enhance their knowledge.

- Ontology Reasoning:
    - Semantic Web agents can employ reasoning engines to infer new knowledge based on the axioms and rules defined in ontologies.
    - Through ontology reasoning, agents can deduce implicit relationships, classify resources, and detect inconsistencies within semantic data.

- Semantic Web Services:
    - Semantic Web services use semantic annotations to describe the capabilities and requirements of web services, enabling automated discovery, composition, and invocation.
    - Semantic Web agents can utilize semantic web services to dynamically discover and integrate services based on their semantic descriptions.

- Semantic Web Rules:
    - Semantic web rules languages such as Rule Interchange Format (RIF) enable the formulation of logical rules for making deductions and drawing conclusions.
    - Semantic Web agents can utilize rules to implement complex inferencing and decision-making capabilities, enhancing their ability to process and interpret semantic data.

Limitations of SWT:

- Semantic web technologies have made significant strides in enabling the structuring and linking of data on the web, but they also have several limitations:

- Complexity: Implementing and using semantic web technologies can be complex. Knowledge of RDF, OWL, SPARQL, and related standards is required, which can be a barrier to entry for some developers and organizations.

- Scalability: While semantic web technologies are effective for representing and querying small to medium-sized datasets, they may face scalability issues with very large datasets. Querying large RDF graphs can be computationally intensive and may require specialized infrastructure.

- Performance: Querying RDF data can be slower compared to traditional databases, especially when dealing with complex queries involving multiple RDF graphs or large datasets. This can impact the real-time responsiveness of applications.

- Interoperability: Achieving interoperability between different semantic web systems and datasets can be challenging. Despite standardized formats and protocols, variations in how ontologies are defined and data is represented can lead to compatibility issues.

- Quality and Trustworthiness: Ensuring the quality and trustworthiness of semantic web data can be challenging. Unlike traditional databases where data validation and integrity constraints are enforced, RDF data is often decentralized and sourced from various providers, raising concerns about data accuracy and reliability.

- Maintenance Overhead: Managing and maintaining semantic web data can be resource-intensive. Ontologies and vocabularies may need to be updated regularly to reflect changes in domain knowledge or requirements. Additionally, ensuring data consistency and alignment with evolving standards and best practices can require ongoing effort and investment in governance processes and infrastructure.

7. Explain how search process of semantic web differ from traditional web.

Aspect	Traditional Web	Semantic Web
Data Representation	HTML pages with unstructured content	RDF triples with structured data
Search Mechanism	Keyword-based search engines (e.g., Google)	Semantic search engines (e.g., SWSE, Swoogle)

Query Interpretation	Based on keyword matching	Utilizes semantics and ontology reasoning
Precision and Recall	Relies on keyword relevance	Improved precision through semantic context
Search Results	Ranked based on keyword frequency	Enhanced relevance based on semantic meaning
Data Integration	Limited ability to integrate disparate sources	Facilitates integration via linked data principles
Query Complexity	Relatively straightforward queries	Supports complex queries involving relationships and semantics

Queries

Example 1:

Consider a dataset of books with information about their titles, authors, genres, and publication years.

turtleCopy code

```
@prefix ex: <http://example.org/books#> .

ex:Book1    ex:title "The Great Gatsby" ;
            ex:author "F. Scott Fitzgerald" ; ex:genre "Fiction" ;
            ex:publicationYear 1925 .

ex:Book2    ex:title "To Kill a Mockingbird" ;
            ex:author "Harper Lee" ; ex:genre "Fiction" ;
            ex:publicationYear 1960 .

ex:Book3    ex:title "1984" ;
            ex:author "George Orwell" ;
            ex:genre "Dystopian Fiction" ;
            ex:publicationYear 1949 .

ex:Book4    ex:title "Pride and Prejudice" ;
            ex:author "Jane Austen" ;
            ex:genre "Romance" ;
            ex:publicationYear 1813 .
```

SPARQL Queries:

1. **Retrieve the titles and authors of all books:**

PREFIX ex: <http://example.org/books#> SELECT ?title ?author WHERE { ?book ex:title ?title ; ex:author ?author . }

Solution:

title	author
The Great Gatsby	F. Scott Fitzgerald
To Kill a Mockingbird	Harper Lee
1984	George Orwell
Pride and Prejudice	Jane Austen

2. **Retrieve the titles and publication years of books published after 1950:**

PREFIX ex: <http://example.org/books#> SELECT ?title ?publicationYear WHERE { ?book ex:title ?title ; ex:publicationYear ?publicationYear . FILTER (?publicationYear > 1950) }

Solution:

title	publicationYear
To Kill a Mockingbird	1960
1984	1949

3. **Retrieve the titles of books written by George Orwell:**

PREFIX ex: <http://example.org/books#> SELECT ?title WHERE { ?book ex:title ?title ; ex:author "George Orwell" . }

Solution:

title
1984

4. **Retrieve the titles and genres of books published before 1900:**

PREFIX ex: <http://example.org/books#> SELECT ?title ?genre WHERE { ?book ex:title ?title ; ex:genre ?genre ; ex:publicationYear ?publicationYear . FILTER (?publicationYear < 1900) }

Solution:

title	genre
Pride and Prejudice	Romance

5. Count the number of books in the dataset:

PREFIX ex: <http://example.org/books#> SELECT (COUNT(?book) as ?numBooks) WHERE { ?book a ex:Book . }

Solution:

numBooks
4

6. Retrieve the titles of books sorted by publication year in ascending order:

PREFIX ex: <http://example.org/books#> SELECT ?title WHERE { ?book ex:title ?title ; ex:publicationYear ?publicationYear . } ORDER BY ASC(?publicationYear)

Solution:

title
Pride and Prejudice
1984
The Great Gatsby
To Kill a Mockingbird

7. Retrieve the unique genres of books in the dataset:

PREFIX ex: <http://example.org/books#> SELECT DISTINCT ?genre WHERE { ?book ex:genre ?genre . }

Solution:

genre
Fiction
Dystopian Fiction
Romance

8. Retrieve the titles of books authored by Harper Lee or George Orwell:

?author . FILTER (?author = "Harper Lee" || ?author = "George Orwell") }

Solution:

title
To Kill a Mockingbird
1984

9. Retrieve the authors and publication years of books published between 1900 and 1950 (inclusive):

Solution:

author	publicationYear
F. Scott Fitzgerald	1925
George Orwell	1949

10. Retrieve the titles of books along with their authors and genres:

ex:author ?author ; ex:genre ?genre . }

Solution:

title	author	genre
The Great Gatsby	F. Scott Fitzgerald	Fiction
To Kill a Mockingbird	Harper Lee	Fiction
1984	George Orwell	Dystopian Fiction
Pride and Prejudice	Jane Austen	Romance

Example 2:

Let's assume we have a university database that contains information about students, courses, and professors. Each student has a unique identifier (studentID), a name, and a

major. Each course has a unique identifier (courseID), a name, and is taught by a professor. Each professor has a unique identifier (professorID) and a name.

Here's an example of a SPARQL query that retrieves the names of all students and their majors from the university database:

```
PREFIX ex: <http://www.example.com/university#>

SELECT ?studentName ?major
WHERE {
 ?student ex:hasName ?studentName ;
       ex:hasMajor ?major .
}
```

In this query:

- **PREFIX ex: <http://www.example.com/university#>** defines a namespace prefix for the university ontology.
- **SELECT ?studentName ?major** specifies the variables we want to retrieve.
- **WHERE { ... }** is the WHERE clause, which contains the triple patterns that define the query pattern.
- **?student ex:hasName ?studentName ; ex:hasMajor ?major .** is a triple pattern that matches students with their names and majors.

SPARQL Queries:

1. **Retrieve all courses and their instructors:**

```
PREFIX ex: <http://www.example.com/university#>

SELECT ?courseName ?instructorName
WHERE {
 ?course ex:hasName ?courseName ;
  ex:taughtBy ?instructor .
 ?instructor ex:hasName ?instructorName .
}
```

2. **Retrieve the number of students enrolled in each course:**

```
PREFIX ex: <http://www.example.com/university#>

SELECT ?courseName (COUNT(?student) AS ?numStudents)
WHERE {
```

```
 ?course ex:hasName ?courseName ;
      ex:hasStudent ?student .
}
GROUP BY ?courseName
```

3. **Retrieve all professors who teach courses in a specific department:**

```
PREFIX ex: <http://www.example.com/university#>

SELECT ?professorName
WHERE {
 ?course ex:taughtBy ?professor ;
      ex:belongsToDepartment ex:ComputerScienceDepartment .
 ?professor ex:hasName ?professorName .
}
```

4. **Retrieve all students who are enrolled in a course taught by a specific professor:**

```
PREFIX ex: <http://www.example.com/university#>

SELECT ?studentName
WHERE {
 ?course ex:taughtBy ex:ProfessorSmith ;
      ex:hasStudent ?student .
 ?student ex:hasName ?studentName .
}
```

5. **Retrieve all courses that have more than 20 students enrolled:**

```
PREFIX ex: <http://www.example.com/university#>

SELECT ?courseName
WHERE {
 ?course ex:hasName ?courseName ;
      ex:hasStudent ?student .
}
GROUP BY ?courseName
HAVING (COUNT(?student) > 20)
```

6. **Retrieve the names of all professors along with the courses they teach:**

```
PREFIX ex: <http://www.example.com/university#>

SELECT ?professorName ?courseName
WHERE {
 ?course ex:taughtBy ?professor ;
       ex:hasName ?courseName .
 ?professor ex:hasName ?professorName .
}
```

7. **Retrieve all students along with the courses they are enrolled in and the grades they received:**

```
PREFIX ex: <http://www.example.com/university#>

SELECT ?studentName ?courseName ?grade
WHERE {
 ?student ex:hasName ?studentName ;
       ex:enrolledIn ?course .
 ?course ex:hasName ?courseName ;
       ex:hasGrade ?grade .
}
```

8. **Retrieve the total number of courses taught by each professor:**

```
PREFIX ex: <http://www.example.com/university#>

SELECT ?professorName (COUNT(?course) AS ?numCourses)
WHERE {
 ?course ex:taughtBy ?professor .
 ?professor ex:hasName ?professorName .
}
GROUP BY ?professorName
```

9. **Retrieve all courses offered by a specific department along with their instructors:**

```
PREFIX ex: <http://www.example.com/university#>

SELECT ?courseName ?instructorName
```

```
WHERE {
 ?course ex:belongsToDepartment ex:ComputerScienceDepartment ;
      ex:taughtBy ?instructor .
 ?course ex:hasName ?courseName .
 ?instructor ex:hasName ?instructorName .
}
```

10. Retrieve the average grade for each course:

```
PREFIX ex: <http://www.example.com/university#>

SELECT ?courseName (AVG(?grade) AS ?avgGrade)
WHERE {
 ?course ex:hasName ?courseName ;
      ex:hasGrade ?grade .
}
GROUP BY ?courseName
```

11. Retrieve the names of all courses along with the total number of enrolled students:

```
PREFIX ex: <http://www.example.com/university#>

SELECT ?courseName (COUNT(?student) AS ?numStudents)
WHERE {
 ?course ex:hasName ?courseName ;
      ex:hasStudent ?student .
}
GROUP BY ?courseName
```

12. Retrieve all students who are enrolled in multiple courses:

```
PREFIX ex: <http://www.example.com/university#>

SELECT ?studentName
WHERE {
 ?student ex:hasName ?studentName ;
       ex:enrolledIn ?course1, ?course2 .
 FILTER (?course1 != ?course2)
}
```

13. Retrieve the names of all courses that have not been assigned a professor:

```
PREFIX ex: <http://www.example.com/university#>

SELECT ?courseName
WHERE {
 ?course ex:hasName ?courseName ;
  ex:taughtBy ?professor .
 FILTER NOT EXISTS { ?course ex:taughtBy ?professor }
}
```

14. Retrieve the names of all students who have not enrolled in any course:

```
PREFIX ex: <http://www.example.com/university#>

SELECT ?studentName
WHERE {
 ?student ex:hasName ?studentName .
 FILTER NOT EXISTS { ?student ex:enrolledIn ?course }
}
```

15. Retrieve all professors who teach courses in a specific building:

```
PREFIX ex: <http://www.example.com/university#>

SELECT ?professorName
WHERE {
 ?course ex:taughtBy ?professor ;
      ex:heldInBuilding ex:BuildingA .
 ?professor ex:hasName ?professorName .
}
```

Ontology Model Design

1. **Consider a simple ontology about animals.**

   Classes: We have defined two classes - Fruit and Citrus. Citrus is a subclass of Fruits.
   Properties: We have defined two properties - hasColor and hasTaste, both having domain Fruit.

Individuals: We have defined two individual fruits - apple and orange. Both have properties hasColor and hasTaste. The orange fruit is also explicitly asserted as an instance of Citrus.
This RDF-XML represents a simple ontology about fruits, with classes, properties, and individual instances modeled accordingly.

Solution:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:ont="http://example.org/fruits#">

  <!-- Classes -->
  <rdfs:Class rdf:about="http://example.org/fruits#Fruit"/>
  <rdfs:Class rdf:about="http://example.org/fruits#Citrus">
    <rdfs:subClassOf rdf:resource="http://example.org/fruits#Fruit"/>
  </rdfs:Class>

  <!-- Properties -->
  <rdf:Property rdf:about="http://example.org/fruits#hasColor">
    <rdfs:domain rdf:resource="http://example.org/fruits#Fruit"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </rdf:Property>
  <rdf:Property rdf:about="http://example.org/fruits#hasTaste">
    <rdfs:domain rdf:resource="http://example.org/fruits#Fruit"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </rdf:Property>

  <!-- Individuals -->
  <ont:Fruit rdf:about="http://example.org/fruits#apple">
    <ont:hasColor>red</ont:hasColor>
    <ont:hasTaste>sweet</ont:hasTaste>
  </ont:Fruit>
  <ont:Fruit rdf:about="http://example.org/fruits#orange">
    <ont:hasColor>orange</ont:hasColor>
    <ont:hasTaste>tangy</ont:hasTaste>
    <rdf:type rdf:resource="http://example.org/fruits#Citrus"/>
  </ont:Fruit>

</rdf:RDF>
```

2. **Design Generic Ontology Modelling in RDF-XML. Let's consider a simple ontology about animals.**

Classes: Defined classes for Animal, Mammal, and Bird. Mammal and Bird are subclasses of Animal.
Properties: Defined properties such as hasColor (relating Animal to color) and hasNumberOfLegs (relating Animal to the number of legs).
Individuals: Defined individual instances such as dog (a Mammal with color brown and 4 legs) and eagle (a Bird with color brown and 2 legs).

This RDF-XML represents a simple ontology about animals, with classes, properties, and individual instances modeled accordingly.

Solution:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:ont="http://example.org/fruits#">

  <!-- Classes -->
  <rdfs:Class rdf:about="http://example.org/fruits#Fruit"/>
  <rdfs:Class rdf:about="http://example.org/fruits#Citrus">
    <rdfs:subClassOf rdf:resource="http://example.org/fruits#Fruit"/>
  </rdfs:Class>

  <!-- Properties -->
  <rdf:Property rdf:about="http://example.org/fruits#hasColor">
    <rdfs:domain rdf:resource="http://example.org/fruits#Fruit"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </rdf:Property>
  <rdf:Property rdf:about="http://example.org/fruits#hasTaste">
    <rdfs:domain rdf:resource="http://example.org/fruits#Fruit"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </rdf:Property>

  <!-- Individuals -->
```

```
  <ont:Fruit rdf:about="http://example.org/fruits#apple">
    <ont:hasColor>red</ont:hasColor>
    <ont:hasTaste>sweet</ont:hasTaste>
  </ont:Fruit>
  <ont:Fruit rdf:about="http://example.org/fruits#orange">
    <ont:hasColor>orange</ont:hasColor>
    <ont:hasTaste>tangy</ont:hasTaste>
    <rdf:type rdf:resource="http://example.org/fruits#Citrus"/>
  </ont:Fruit>

</rdf:RDF>
```

3. Design the ontology model for following:

a. Classes: We have defined three classes - Person, Employee, and Department. Employee is a subclass of Person.
b. Properties: We have defined two properties - worksFor and hasSalary. worksFor relates individuals of type Person to instances of Department, while hasSalary relates individuals of type Employee to their salary values.

Individuals: We have defined three individual instances - John (a Person), Jane (an Employee), and HR (a Department). Jane works for the HR department and has a salary of 50000. This RDF-XML represents a simple ontology about employees, departments, and their relationships.

Solution:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:ont="http://example.org/ontology#">

  <!-- Classes -->
  <rdfs:Class rdf:about="http://example.org/ontology#Person"/>
```

```
  <rdfs:Class rdf:about="http://example.org/ontology#Employee">
    <rdfs:subClassOf rdf:resource="http://example.org/ontology#Person"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="http://example.org/ontology#Department"/>

  <!-- Properties -->
  <rdf:Property rdf:about="http://example.org/ontology#worksFor">
    <rdfs:domain rdf:resource="http://example.org/ontology#Person"/>
    <rdfs:range rdf:resource="http://example.org/ontology#Department"/>
  </rdf:Property>
  <rdf:Property rdf:about="http://example.org/ontology#hasSalary">
    <rdfs:domain rdf:resource="http://example.org/ontology#Employee"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
  </rdf:Property>

  <!-- Individuals -->
  <ont:Person rdf:about="http://example.org/ontology#John"/>
  <ont:Employee rdf:about="http://example.org/ontology#Jane">
    <ont:worksFor rdf:resource="http://example.org/ontology#HR"/>
    <ont:hasSalary>50000</ont:hasSalary>
  </ont:Employee>
  <ont:Department rdf:about="http://example.org/ontology#HR"/>

</rdf:RDF>
```

4. Model the following music production ontology using RDFS Performance, Recording, Arrangement are Events. SoloPerformance is a type of Performance. Performances have Performers. Performers may be a Group or a Person. Performers have names of type string (In case of a Group, this is the group name). "hasMember" is a property of a Group and the values of the property must be of type Person. AudioRecording and VideoRecording are types of Recording.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:ont="http://example.org/ontology#">

  <!-- Classes -->
  <rdfs:Class rdf:about="http://example.org/ontology#Event"/>
  <rdfs:Class rdf:about="http://example.org/ontology#Performance">
    <rdfs:subClassOf rdf:resource="http://example.org/ontology#Event"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="http://example.org/ontology#Recording">
    <rdfs:subClassOf rdf:resource="http://example.org/ontology#Event"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="http://example.org/ontology#Arrangement">
    <rdfs:subClassOf rdf:resource="http://example.org/ontology#Event"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="http://example.org/ontology#SoloPerformance">
    <rdfs:subClassOf rdf:resource="http://example.org/ontology#Performance"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="http://example.org/ontology#Person"/>
  <rdfs:Class rdf:about="http://example.org/ontology#Group"/>
  <rdfs:Class rdf:about="http://example.org/ontology#AudioRecording">
    <rdfs:subClassOf rdf:resource="http://example.org/ontology#Recording"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="http://example.org/ontology#VideoRecording">
    <rdfs:subClassOf rdf:resource="http://example.org/ontology#Recording"/>
```

```
  </rdfs:Class>

  <!-- Properties -->
  <rdf:Property rdf:about="http://example.org/ontology#hasMember">
    <rdfs:domain rdf:resource="http://example.org/ontology#Group"/>
    <rdfs:range rdf:resource="http://example.org/ontology#Person"/>
  </rdf:Property>

  <!-- Individuals -->
  <ont:Person rdf:about="http://example.org/ontology#John"/>
  <ont:Group rdf:about="http://example.org/ontology#TheBand"/>
  <ont:AudioRecording rdf:about="http://example.org/ontology#SongRecording"/>
  <ont:VideoRecording rdf:about="http://example.org/ontology#MusicVideo"/>

</rdf:RDF>
```

5. Model the following education ontology in RDFS using RDF-XML/Turtle. School, College and University are places of education. International university is a type of university; A School has a school bus. Buses have numbers. University has buildings. Each building has a code number. Cafeteria and library are types of building.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:edu="http://example.org/education#">

  <!-- Classes -->
  <rdfs:Class rdf:about="http://example.org/education#Place"/>
  <rdfs:Class rdf:about="http://example.org/education#EducationFacility">
    <rdfs:subClassOf rdf:resource="http://example.org/education#Place"/>
  </rdfs:Class>
```

```
<rdfs:Class rdf:about="http://example.org/education#School">
  <rdfs:subClassOf rdf:resource="http://example.org/education#EducationFacility"/>
</rdfs:Class>
<rdfs:Class rdf:about="http://example.org/education#College">
  <rdfs:subClassOf rdf:resource="http://example.org/education#EducationFacility"/>
</rdfs:Class>
<rdfs:Class rdf:about="http://example.org/education#University">
  <rdfs:subClassOf rdf:resource="http://example.org/education#EducationFacility"/>
</rdfs:Class>
<rdfs:Class rdf:about="http://example.org/education#InternationalUniversity">
  <rdfs:subClassOf rdf:resource="http://example.org/education#University"/>
</rdfs:Class>
<rdfs:Class rdf:about="http://example.org/education#Building"/>
<rdfs:Class rdf:about="http://example.org/education#Cafeteria">
  <rdfs:subClassOf rdf:resource="http://example.org/education#Building"/>
</rdfs:Class>
<rdfs:Class rdf:about="http://example.org/education#Library">
  <rdfs:subClassOf rdf:resource="http://example.org/education#Building"/>
</rdfs:Class>

<!-- Properties -->
<rdf:Property rdf:about="http://example.org/education#hasBus">
  <rdfs:domain rdf:resource="http://example.org/education#School"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</rdf:Property>
<rdf:Property rdf:about="http://example.org/education#hasNumber">
  <rdfs:domain rdf:resource="http://example.org/education#Bus"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
</rdf:Property>
<rdf:Property rdf:about="http://example.org/education#hasBuilding">
  <rdfs:domain rdf:resource="http://example.org/education#University"/>
```

```
    <rdfs:range rdf:resource="http://example.org/education#Building"/>
  </rdf:Property>
  <rdf:Property rdf:about="http://example.org/education#hasCodeNumber">
    <rdfs:domain rdf:resource="http://example.org/education#Building"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </rdf:Property>

  <!-- Individuals -->
  <edu:School rdf:about="http://example.org/education#ElementarySchool">
    <edu:hasBus>School Bus</edu:hasBus>
  </edu:School>
  <edu:College rdf:about="http://example.org/education#CommunityCollege"/>
  <edu:University rdf:about="http://example.org/education#StateUniversity"/>
  <edu:InternationalUniversity rdf:about="http://example.org/education#GlobalUniversity"/>
  <edu:Building rdf:about="http://example.org/education#MainBuilding">
    <edu:hasCodeNumber>MB01</edu:hasCodeNumber>
  </edu:Building>
  <edu:Cafeteria rdf:about="http://example.org/education#CampusCafeteria"/>
  <edu:Library rdf:about="http://example.org/education#UniversityLibrary"/>
</rdf:RDF>
```