Knowledge Representation & OWL

Lectures 21-22: Complete Exam Guide

Simplified for Beginners with Practical Examples

February 1, 2026

Contents

1 Introduction: From Documents to Data

The Big Picture

Imagine the internet today: when you search for "Paris", Google finds web pages that **contain the word** "Paris". But what if we want machines to **understand** that Paris is a city, the capital of France, and has a population?
This is the difference between:

- **Web of Documents** (current): Humans read and understand
- **Web of Data** (goal): Machines understand relationships and meaning

1.1 Why Do We Need Semantic Web?

Real-World Scenario

You want to find: "Lecturers who teach Big Data at universities in Edinburgh"
Old Way (Documents):

- Search: "lecturer big data Edinburgh"
- Get: Random web pages containing these words
- You must read each page to find the answer

New Way (Semantic Web):

- Machine understands: John → is_a → Lecturer
- John → teaches → Big Data Course
- Big Data Course → taught_at → Edinburgh
- Direct answer: "John"

2 Knowledge Graphs: The Foundation

2.1 What is a Knowledge Graph?

Simple Definition

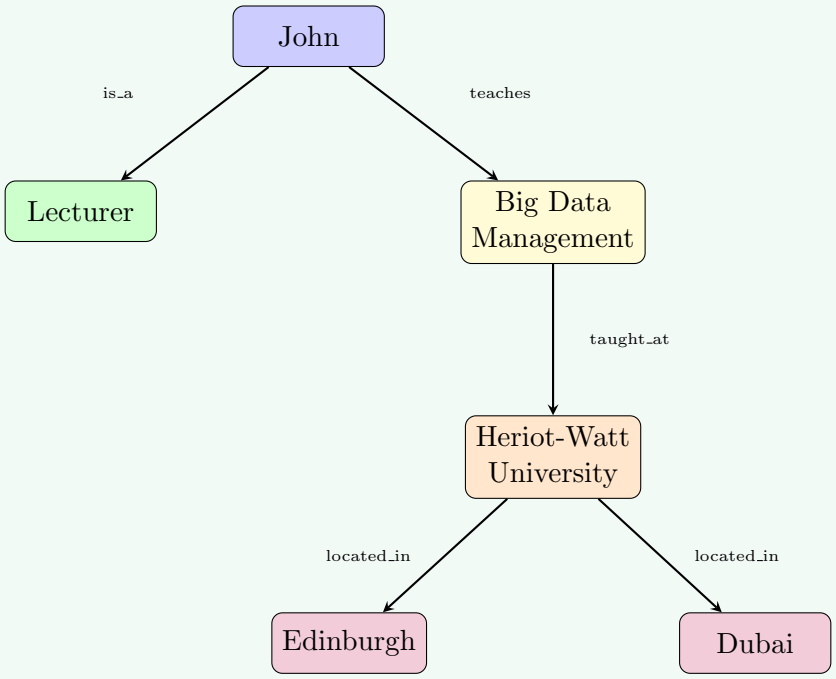
A **knowledge graph** is like a map where:

- **Circles (nodes)** = Things (people, places, courses)
- **Arrows (edges)** = Relationships between things

2.2 Example: Building a Knowledge Graph

Sentence to Graph

Sentence: "John is a lecturer who teaches Big Data Management course at Heriot-Watt University taught in Edinburgh and Dubai."



2.3 5 Key Characteristics of Knowledge Graphs

lightgray **Characteristic**	**Simple Explanation**
Scaling	One graph can store many types of knowledge simultaneously (geography, people, jobs, etc.)
Agreement	Everyone must use the same relationship names (standardization)
Structure	We must define which things can be connected by which relationships
Plurality	Same relationship can appear multiple times (e.g., many "is_a" connections)
Asymmetry	Relationships have direction: A→B doesn't mean B→A

Table 1: Knowledge Graph Characteristics - Learn These for Exams!

Exam Question: "Explain why knowledge graphs use directed relationships."
Answer: Relationships are asymmetric. For example, "John teaches Course" does NOT mean "Course teaches John". Direction matters!

3 Reading Knowledge Graphs: Triples

Triple

Every arrow in a knowledge graph can be written as:

$$(\text{Subject}, \text{Predicate}, \text{Object})$$

- **Subject** = Starting point (who/what)
- **Predicate** = Relationship (action/connection)
- **Object** = Ending point (who/what/value)

3.1 Converting Graph to Triples

Graph → Triples Conversion

From our previous graph:

1. (John, is_a, Lecturer)
2. (John, teaches, Big Data Management)
3. (Big Data Management, taught_at, Heriot-Watt University)
4. (Heriot-Watt University, located_in, Edinburgh)
5. (Heriot-Watt University, located_in, Dubai)

Key Point: A knowledge graph = Collection of triples

4 RDF: The Language of Semantic Web

4.1 What is RDF?

RDF

RDF is the **standard language** for writing knowledge graphs.
Think of it like:

- HTML is for web pages
- RDF is for knowledge graphs

4.2 RDF Structure

lightgray **Part**	**What It Is**	**Example**
Subject	Resource with IRI	`http://example.org/John`
Predicate	Property with IRI	`http://example.org/teaches`
Object	Resource (IRI) OR Literal (value)	IRI: `http://example.org/Course1` Literal: "John Smith"

Table 2: RDF Triple Components

Common Mistake

IRI vs Literal:

- **IRI** = Identifier for a thing (use when thing can have properties)
- **Literal** = Actual value (string, number, date)

Example:

- `(John, hasName, "John Smith")` ← Literal (just text)
- `(John, knows, Bob)` ← IRI (Bob is a person with properties)

4.3 RDF Serialization Formats

RDF can be written in different formats (like saving a Word doc as .docx or .pdf):

lightgray **Format**	**Description**	**File Extension**
RDF/XML	Original format (verbose, hard to read)	.rdf, .xml
Turtle	Easy to read for humans (**most common**)	.ttl
N-Triples	One triple per line (simple)	.nt
JSON-LD	JSON format (for web developers)	.jsonld

Table 3: RDF Formats - Turtle is most exam-relevant!

5 RDF Containers: Grouping Data

5.1 Why Containers?

Sometimes you need to group multiple things together. RDF has 3 types of containers:

lightgray **Container**	**Use When**	**Example**
rdf:Bag	Order doesn't matter	Beatles members: John, Paul, George, Ringo (any order)
rdf:Seq	Order matters	Song rankings: 1st, 2nd, 3rd
rdf:Alt	Choose one option	Album formats: CD OR Record OR Tape (pick one)

Table 4: RDF Container Types

5.2 Example: Beatles Band Members

Bag Example (Unordered)

Turtle Format:

```
@prefix cd: <http://www.recshop.fake/cd#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<http://www.recshop.fake/cd/Beatles>
   cd:artist [
     a rdf:Bag ;
     rdf:_1 "John" ;
     rdf:_2 "Paul" ;
     rdf:_3 "George" ;
     rdf:_4 "Ringo"
   ] .
```

Explanation:

- `rdf:Bag` = Unordered list
- `rdf:_1`, `rdf:_2`, etc. = Items in the bag
- Order doesn't matter (John could be _2 and Paul _1)

Exam Question: "When would you use rdf:Seq vs rdf:Bag?"
Answer: Use **rdf:Seq** when order is important (e.g., rankings, steps). Use **rdf:Bag** when order doesn't matter (e.g., group members).

6 Blank Nodes: Anonymous Resources

What is a Blank Node?

A **blank node** is a resource without a name (IRI). Use it when:

- The resource doesn't need to be referenced from outside
- You don't want to create an IRI for something temporary

Symbol: Empty circle or `[ ]` in Turtle

Blank Node Example

Scenario: John knows someone named Paul Smith (email: paul.smith@example.com), but we don't need to reference Paul elsewhere.
Turtle Format:

```
@prefix pep: <http://example.com/people#> .

<pep:john>
    pep:knows [
        pep:surname "Smith" ;
        pep:givenName "Paul" ;
        pep:email "paul.smith@example.com"
    ] .
```

Explanation:

- `[ ... ]` = Blank node (no IRI)
- All properties inside describe the blank node
- System generates internal ID automatically

7 RDFS: Adding Structure to RDF

7.1 Problem with Pure RDF

RDF Limitation

In pure RDF, nothing stops you from writing nonsense like:

- `(MyCar, lastName, "Honda")`
- `(MyDog, teaches, "Mathematics")`

Cars don't have last names! Dogs don't teach!

7.2 RDFS = RDF Schema

RDFS Solution

RDFS adds **types** and **rules** to RDF:

- Define what **classes** (types of things) exist
- Define what **properties** (relationships) are allowed
- Define which properties work with which classes

Think of it like defining rules for a database before entering data.

7.3 RDFS Classes (Meta-Classes)

lightgray **RDFS Class**	**Simple Explanation**
rdfs:Class	The "mother of all classes" - defines what a class is
rdfs:Resource	Everything is a resource (most general thing)
rdfs:Literal	Values like numbers, strings, dates
rdfs:Property	Defines what a property is
rdfs:Datatype	Types of data (integer, string, boolean, etc.)

Table 5: RDFS Built-in Classes

7.4 RDFS Properties (Key Relationships)

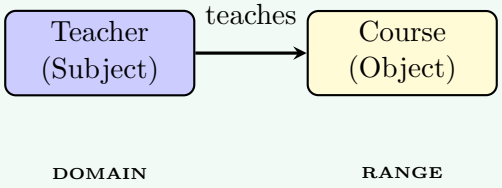
lightgray **Property**	**Meaning**	**Example**
rdfs:subClassOf	A is a type of B	Horse subClassOf Animal
rdfs:subPropertyOf	Property A is a special case of B	hasDaughter subPropertyOf hasChild
rdfs:domain	Which class can use this property (Subject)	teaches domain = Teacher
rdfs:range	What type the property points to (Object)	teaches range = Course

Table 6: RDFS Properties - VERY IMPORTANT FOR EXAMS

7.5 Domain and Range Explained

Domain & Range

Define: Property "teaches"



RDFS Code:

```
:teaches rdf:type rdf:Property ;
         rdfs:domain :Teacher ;
         rdfs:range :Course .
```

What this means:

- Only Teachers can be subjects of "teaches"
- Only Courses can be objects of "teaches"
- `(Student, teaches, Course)` would be invalid!

Common Exam Question: "What is the domain and range of property X?"
Remember:

- **Domain** = Type of Subject (left side)
- **Range** = Type of Object (right side)

8 RDFS Example: Animal Ontology

Complete RDFS Example

Goal: Create a simple animal classification
Step 1: Define Classes (Hierarchy)

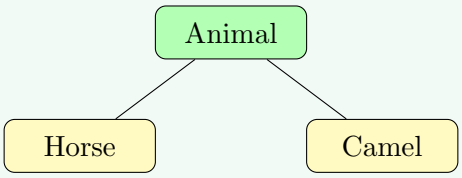
```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix aml: <http://www.animals.fake/animals#> .

aml:Animal a rdfs:Class .

aml:Horse a rdfs:Class ;
    rdfs:subClassOf aml:Animal .

aml:Camel a rdfs:Class ;
    rdfs:subClassOf aml:Animal .
```

Hierarchy Visualization:



9 RDFS Inference: Discovering New Facts

What is Inference?

Inference = Automatically discovering facts that aren't explicitly stated by applying logical rules.
Think: If A, and A→B, then B

9.1 Type 1: Inheritance Inference

Type Inheritance

Given Facts:

1. Alice is a Person
2. Person is subClassOf Agent

Inference Rule:

$$\frac{\text{alice } rdf : type \text{ Person} \quad \text{Person } rdfs : subClassOf \text{ Agent}}{\text{alice } rdf : type \text{ Agent}}$$

Conclusion: Alice is an Agent (even though not explicitly stated)

9.2 Type 2: Property Constraint Inference

Domain/Range Inference

Given Facts:

1. Property "knows" has domain Person
2. Alice knows Bob

Inference Rule:

$$\frac{\text{knows } rdfs : domain \text{ Person} \quad \text{Alice } knows \text{ Bob}}{\text{Alice } rdf : type \text{ Person} \quad \text{Bob } rdf : type \text{ Person}}$$

Conclusion: Both Alice and Bob are Persons (inferred from relationship)

Exam Question: "Given the following triples, what can be inferred?"
Strategy:

1. Look for subClassOf chains → apply transitivity
2. Look for property constraints (domain/range) → infer types
3. Look for subPropertyOf → generalize relationships

10 Popular Vocabularies

10.1 Why Use Standard Vocabularies?

Don't Reinvent the Wheel

Instead of creating your own terms, use established vocabularies:

- **Interoperability:** Others understand your data
- **Tools:** Software already supports them
- **Best Practices:** Already well-designed

10.2 Common Vocabularies

lightgray **Vocabulary**	**Purpose**	**Example Terms**
Dublin Core	Document metadata	dc:title, dc:creator, dc:date
FOAF	People & social networks	foaf:Person, foaf:knows, foaf:name
SKOS	Taxonomies & thesauri	skos:Concept, skos:broader, skos:narrower
Schema.org	General web content	schema:Product, schema:price, schema:rating

Table 7: Standard Vocabularies - Use These in Projects!

11 SKOS: Simple Knowledge Organization System

11.1 What is SKOS?

SKOS in Simple Terms

SKOS is for creating **structured vocabularies**:

- **Taxonomy:** Hierarchical classification (e.g., Animal → Mammal → Cat)
- **Thesaurus:** Related terms and synonyms
- **Controlled Vocabulary:** Standard terms for a domain

11.2 SKOS Core Elements

lightgray **SKOS Term**	**Meaning**	**Example**
skos:Concept	A unit of thought/idea	”Cat”, ”Mammal”, ”Animal”
skos:prefLabel	Preferred name	”Cat”@en
skos:altLabel	Alternative name (synonym)	”Feline”@en
skos:broader	More general concept	Cat broader Mammal
skos:narrower	More specific concept	Mammal narrower Cat
skos:related	Associated concept	Cat related Dog

Table 8: SKOS Key Properties

11.3 SKOS Hierarchy Example

Animal Taxonomy with SKOS

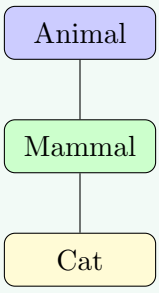
```
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .

:Animal rdf:type skos:Concept ;
    skos:prefLabel "Animal"@en .

:Mammal rdf:type skos:Concept ;
    skos:prefLabel "Mammal"@en ;
    skos:broader :Animal .

:Cat rdf:type skos:Concept ;
    skos:prefLabel "Cat"@en ;
    skos:altLabel "Feline"@en ;
    skos:broader :Mammal ;
    skos:related :Dog .
```

Hierarchy:



11.4 SKOS vs RDFS: Key Difference

Common Confusion

DON'T mix these up:

lightgray **Aspect**	**RDFS**	**SKOS**
Purpose	Define class structure	Organize concepts
Relationship	rdfs:subClassOf	skos:broader
Use for	Ontology structure	Controlled vocabularies
Example	Country subClassOf Location	UK broader Europe

Wrong: UK rdfs:subClassOf Europe
Right: UK skos:broader Europe

12 Limitations of RDFS

Why RDFS is Not Enough

RDFS cannot express:

1. **Cardinality:** "A person has exactly 2 parents"
2. **Property characteristics:** "touches is symmetric"
3. **Conditional constraints:** "hasChild range is Person ONLY when subject is Person"
4. **Complex reasoning:** Disjoint classes, equivalence, etc.

Solution: We need OWL!

13 OWL: Web Ontology Language

13.1 What is OWL?

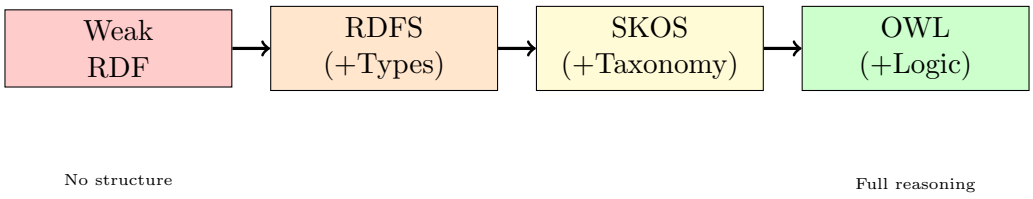
OWL Definition

OWL (Web Ontology Language) extends RDFS with:

- Rich expressiveness (say complex things)
- Formal logic (precise meaning)
- Automated reasoning (discover new facts)
- Complex constraints (strict rules)

Think: RDFS = Basic rules, OWL = Advanced rules + Logic

13.2 Semantic Spectrum



13.3 Ontology vs Database Schema

lightgray **Aspect**	**Database Schema**	**Ontology (OWL)**
Purpose	Organize data storage	Share knowledge with semantics
Semantics	In code or human mind	Formally defined in ontology
Relationships	Implicit (foreign keys)	Explicit (named properties)
Reasoning	No automatic reasoning	Automatic inference
Flexibility	Fixed structure	Open world (extensible)

Table 9: Ontology vs Database - Important Distinction!

14 OWL Key Components

14.1 TBox vs ABox

Two Parts of Knowledge

- **TBox (Terminology Box):** The rules/structure (Classes, Properties, Constraints)
- **ABox (Assertion Box):** The actual data (Individuals/Instances)

Analogy:

- TBox = Recipe (what ingredients, steps)
- ABox = Actual cake you made (the data)

TBox vs ABox

TBox (Structure):

- Class: Person
- Class: Course
- Property: teaches (Teacher → Course)

ABox (Data):

- John is a Person
- F21BD is a Course
- John teaches F21BD

14.2 OWL Class Characteristics

lightgray **Characteristic**	**Meaning**	**Example**
owl:equivalentClass	Two classes refer to same things	Human Person
owl:disjointWith	No overlap (can't be both)	Man Woman
rdfs:subClassOf	A is type of B	Dog Animal

Table 10: OWL Class Relationships

Disjoint Classes

Important: If classes are disjoint, an individual CANNOT be instance of both.
Example:

- Man disjointWith Woman
- Alice type Woman
- Alice type Man ← INCONSISTENT!

15 OWL Properties

15.1 Object Properties vs Data Properties

lightgray **Property Type**	**Connects**	**Example**
Object Property	Individual → Individual	John teaches Course1
Data Property	Individual → Literal (value)	John hasAge 35

Table 11: Property Types

15.2 Object Property Characteristics

lightgray **Characteristic**	**Meaning**	**Example**
Functional	Each subject has at most 1 object	hasBirthDate (only one birthday)
InverseFunctional	Each object has at most 1 subject	hasSSN (unique ID)
Transitive	If A→B and B→C, then A→C	ancestor (if A ancestor B, B ancestor C, then A ancestor C)
Symmetric	If A→B, then B→A	sibling (if A sibling B, then B sibling A)
Asymmetric	If A→B, then NOT B→A	parentOf (if A parent B, B cannot be parent of A)
Reflexive	Everything related to itself	knows (person knows themselves)
Irreflexive	Nothing related to itself	parentOf (can't be your own parent)

Table 12: Property Characteristics - MEMORIZE FOR EXAMS

Inverse Properties

Definition:

```
:hasWife owl:inverseOf :hasHusband .
```

Given:

```
:John :hasWife :Mary .
```

Reasoner infers:

```
:Mary :hasHusband :John .
```

Key Point: You only state one direction; reasoner discovers the other!

16 OWL Reasoning

What is Reasoning?

Reasoning = Automatic discovery of facts using logic rules.
A **reasoner** is software that:

1. Checks consistency (no contradictions)
2. Infers class membership (classification)
3. Discovers new relationships (realization)

16.1 4 Types of Reasoning

lightgray **Reasoning Type**	**Question**	**Example**
Consistency	Is ontology contradiction-free?	Check: Can Alice be both Man and Woman (if disjoint)?
Satisfiability	Can a class have instances?	Can class "MarriedBachelor" have individuals?
Classification	What is class hierarchy?	Build full subClassOf tree
Realization	What class does individual belong to?	John teaches Course → John is Teacher

Table 13: OWL Reasoning Tasks

Reasoning Example

Ontology (TBox):

- Property "studies" domain: Student, range: Course
- Property "hasStudent" inverseOf "studies"

Data (ABox):

- Alice studies F21BD

Reasoner Infers:

1. Alice type Student (from domain constraint)
2. F21BD type Course (from range constraint)
3. F21BD hasStudent Alice (from inverse property)

17 Lab Exercise: Building Ontology in Protégé

17.1 What is Protégé?

Protégé Tool

Protégé is a free, open-source ontology editor:

- Graphical interface (no coding needed)
- Built-in reasoners (HermiT, Pellet)
- Exports to OWL formats (Turtle, RDF/XML)

Download: `https://protege.stanford.edu`

17.2 Step-by-Step: Create University Ontology

Lab Exercise Walkthrough

Step 1: Set Up Ontology

1. Open Protégé
2. File → New
3. Set IRI: `http://example.org/university`
4. Add metadata:
   - dcterms:title = "University Ontology"
   - dcterms:creator = "Your Name"
   - dcterms:description = "Ontology for university domain"
5. Save as Turtle format (.ttl)

Step 2: Create Class Hierarchy

1. Entities tab → Classes
2. Select `owl:Thing`
3. Click "Add subclass" button
4. Create: `Person`, `Course`
5. Make them **disjoint**:
   - Select Person
   - Disjoint With → Add → Course
6. Under Person, create:
   - Staff (subclass of Person)
     - Academic (subclass of Staff)
     - Administrative (subclass of Staff)
   - Student (subclass of Person)
     - Undergraduate (subclass of Student)
     - Postgraduate (subclass of Student)

Step 3: Create Object Properties

1. Entities tab → Object Properties
2. Create properties:
   - `teaches` (Academic → Course)
   - `taughtBy` (Course → Academic) [inverse of teaches]
   - `studies` (Student → Course)
   - `hasStudent` (Course → Student) [inverse of studies]
3. Set characteristics:
   - Select `studies`

17.3 Testing Inconsistency

Deliberate Error

Create Inconsistency:

1. Select alice
2. Add type: Course
3. Synchronize reasoner
4. ERROR: "Inconsistent ontology"

Why?

- Person disjoint with Course
- Alice is Person (from Student subClassOf Person)
- Alice is Course (you just added)
- Contradiction!

Fix: Remove "alice type Course"

18 Exam Preparation Guide

18.1 Key Concepts Checklist

Master These Topics:

- ☐ Knowledge graphs: nodes, edges, triples
- ☐ RDF: Subject-Predicate-Object structure
- ☐ RDF serialization: Turtle format (most important)
- ☐ RDF containers: Bag, Seq, Alt
- ☐ Blank nodes: when and why to use
- ☐ RDFS: Classes, Properties, domain, range
- ☐ RDFS: subClassOf, subPropertyOf
- ☐ RDFS inference: type inheritance, constraint inference
- ☐ SKOS: Concept, broader, narrower, related
- ☐ SKOS vs RDFS: when to use each
- ☐ OWL: equivalentClass, disjointWith
- ☐ OWL properties: characteristics (functional, symmetric, etc.)
- ☐ OWL reasoning: consistency, classification, realization
- ☐ Protégé: creating classes, properties, individuals

18.2 Common Exam Questions

Question Type 1: Convert Sentence to Triples
Example: "Alice, a student, studies Math course taught by Professor John at MIT."
Answer:

1. (Alice, rdf:type, Student)
2. (Alice, studies, Math)
3. (Math, rdf:type, Course)
4. (Math, taughtBy, John)
5. (John, rdf:type, Professor)
6. (Math, taughtAt, MIT)

Question Type 2: Domain/Range Identification
Given: Property "writes" connects Authors to Books
Questions:

1. What is domain of "writes"? → Author
2. What is range of "writes"? → Book

Question Type 3: Inference
Given:

- Cat subClassOf Mammal
- Mammal subClassOf Animal
- Fluffy type Cat

What can be inferred?

- Fluffy type Mammal (from subClassOf transitivity)
- Fluffy type Animal (from subClassOf transitivity)

Question Type 4: Property Characteristics
Classify these properties:

1. "siblingOf" → **Symmetric** (if A sibling B, then B sibling A)
2. "ancestorOf" → **Transitive** (if A ancestor B, B ancestor C, then A ancestor C)
3. "hasMotherparentOf" → **Functional** (each person has exactly 1 mother)
4. "parentOf" → **Asymmetric** (if A parent B, then NOT B parent A)

18.3 Quick Reference Tables

18.3.1 RDF/RDFS Comparison

lightgray **Feature**	**RDF**	**RDFS**
Purpose	Represent data	Define vocabulary
Structure	Triples only	Classes + Properties
Constraints	None	Domain, Range
Hierarchy	No	Yes (subClassOf)
Reasoning	Minimal	Type inference

Table 14: RDF vs RDFS Quick Comparison

18.3.2 RDFS vs OWL Comparison

lightgray **Feature**	**RDFS**	**OWL**
Expressiveness	Basic	Rich
Disjoint classes	No	Yes
Cardinality	No	Yes
Property characteristics	No	Yes (symmetric, etc.)
Reasoning power	Limited	Advanced
Complexity	Simple	Complex

Table 15: RDFS vs OWL Quick Comparison

18.3.3 Property Characteristics Summary

lightgray **Characteristic**	**Quick Test**
Functional	Each subject → at most 1 object
InverseFunctional	Each object ← at most 1 subject
Transitive	A→B, B→C A→C
Symmetric	A→B B→A
Asymmetric	A→B NOT B→A
Reflexive	Every A→A
Irreflexive	Never A→A

Table 16: Property Characteristics Cheat Sheet

19 Practice Problems

19.1 Problem 1: RDF Triples

Problem: Convert this information to RDF triples:
"Emma is a PhD student studying at Heriot-Watt University. She is supervised by Dr. Smith."

Solution:

1. (:Emma, rdf:type, :PhDStudent)
2. (:PhDStudent, rdfs:subClassOf, :Student)
3. (:Emma, :studiesAt, :HeriotWattUniversity)
4. (:Emma, :supervisedBy, :DrSmith)
5. (:DrSmith, rdf:type, :Professor)

19.2 Problem 2: Domain and Range

Problem: Define domain and range for property "manages"
Constraint: Only Managers can manage, and they manage Departments.

Solution:

```
:manages rdf:type owl:ObjectProperty ;
         rdfs:domain :Manager ;
         rdfs:range :Department .
```

19.3 Problem 3: Property Characteristics

Problem: What characteristics apply to these properties?

1. "marriedTo"
2. "brotherOf"
3. "locatedIn" (for geography)

Solution:

1. "marriedTo": **Symmetric** (if A married B, then B married A)
2. "brotherOf": **Irreflexive** (can't be your own brother), **NOT symmetric** (brother of sister)
3. "locatedIn": **Transitive** (if Edinburgh in Scotland, Scotland in UK, then Edinburgh in UK)

19.4 Problem 4: Inference

Problem: Given:

- Property "studies" domain: Student, range: Course
- hasStudent inverseOf studies
- Tom studies CS101

What can be inferred?

Solution:

1. Tom rdf:type Student (from domain of "studies")
2. CS101 rdf:type Course (from range of "studies")
3. CS101 hasStudent Tom (from inverse property)

20 Final Exam Tips

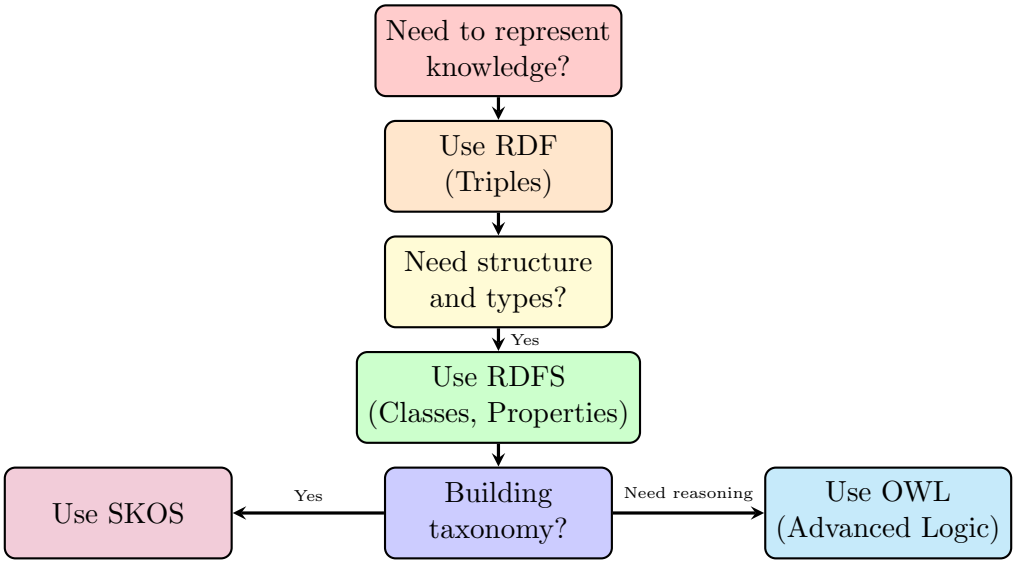
Strategy for Exam Success:

1. **Know the hierarchy:** RDF → RDFS → SKOS/OWL
2. **Master Turtle syntax:** Most practical exam questions use Turtle
3. **Understand inference:** Given facts, what can be concluded?
4. **Memorize property characteristics:** Symmetric, Transitive, etc.
5. **Practice domain/range:** Very common exam question
6. **Know when to use what:**
   - RDF: Just store data
   - RDFS: Add types and basic constraints
   - SKOS: Build taxonomy/thesaurus
   - OWL: Need advanced reasoning
7. **Protégé workflow:** Classes → Properties → Individuals → Reasoner

Common Mistakes to Avoid

- Don't confuse `rdfs:subClassOf` with `rdf:type`
- Don't mix SKOS and RDFS relationships
- Don't forget to run reasoner after changes
- Don't use literals where resources should be used
- Don't forget disjoint classes can't overlap

21 Summary Flowchart



22 Vocabulary Reference

22.1 Must-Know Namespaces

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
```

22.2 Most Common RDF/RDFS Terms

- rdf:type
- rdf:Property
- rdf:Bag
- rdf:Seq
- rdf:Alt
- rdfs:Class
- rdfs:Resource
- rdfs:Literal
- rdfs:subClassOf
- rdfs:subPropertyOf
- rdfs:domain
- rdfs:range
- rdfs:label
- rdfs:comment

22.3 Most Common OWL Terms

- owl:Class
- owl:Thing
- owl:ObjectProperty
- owl:DatatypeProperty

- owl:equivalentClass
- owl:disjointWith
- owl:inverseOf
- owl:FunctionalProperty
- owl:TransitiveProperty
- owl:SymmetricProperty
- owl:AsymmetricProperty
- owl:ReflexiveProperty
- owl:IrreflexiveProperty

23 Conclusion

Key Takeaways

1. **Knowledge graphs** represent data as interconnected nodes and relationships
2. **RDF** provides standard way to encode knowledge graphs as triples
3. **RDFS** adds structure with classes, properties, and basic constraints
4. **SKOS** specialized for building taxonomies and controlled vocabularies
5. **OWL** provides rich expressiveness and automated reasoning
6. **Protégé** is practical tool for building and testing ontologies
7. **Reasoning** automatically discovers facts from rules and data

Good luck on your exam!

Remember: Practice with Protégé and work through examples.