

# RDF stores and data persistence

**Petr Křemen**

Ontologies and Semantic Web  
Winter 2025

# Outline

- RDF stores (overview)
- Storing/Indexing approaches
- Inferencing
- Access Control
- Programmatic Access to Ontologies



# RDF Store Overview

# RDF stores

- RDF4J
- GraphDB
- Virtuoso
- Fuseki
- Stardog
- AllegroGraph
- Amazon Neptune
- BlazeGraph
- Fluree

# RDF stores

RDF Store	Communication protocols	Inferencing
<a href="#">AllegroGraph</a>	SPARQL, custom API	- query-time (rule-based) RDFS++ reasoner
<a href="#">Amazon Neptune</a>	SPARQL, Gremlin, GraphQL	- no symbolic inference support (can be achieved by integrating with RDFox, e.g.)
<a href="#">AnzoGraph DB</a>	SPARQL, OpenCypher, REST	- materialized RDFS+ and OWL2-RL reasoner
<a href="#">Fuseki</a>	SPARQL	- materialized reasoning through custom rules
<a href="#">GraphDB</a>	SPARQL, (SQL), (REST)	- materialized custom rulesets
<a href="#">RDF4J Server/Workbench</a>	SPARQL	- materialized custom rulesets
<a href="#">RDFox</a>	SPARQL, REST	- materialized datalog reasoning
<a href="#">Stardog</a>	SPARQL, SQL, GraphQL	- query-time reasoning using predefined and custom rules
<a href="#">Virtuoso</a>	SPARQL, SQL	- v7 open-source: materializing just rdfs:subClassOf and rdfs:subPropertyOf transitivity - v8 commercial: custom rules

# Storing / Indexing Approaches

# RDF store

- SPARQL API
- often REST API
- indexing crucial, e.g.
  - SPOC
  - POSC
- more indexes
  - faster queries,
  - slower updates,
  - bigger disk footprint

## Triple store

subject	predicate	object
:John	:loves	:Peggy
:Peggy	rdf:type	:Person
...	...	...

## Quad store

subject	predicate	object	context
:John	:loves	:Peggy	:people
:Peggy	rdf:type	:Person	:people
...	...	...	...

# Triple Table

subject	predicate	object
:John	:loves	:Peggy
:Peggy	rdf:type	:Person
:Mary	:loves	:George
:John	rdf:type	:Man
...	...	...

- + simple implementation
- - eliminates self-joins



# Property Table

subject	:loves	rdf:type
:John	:Peggy	:Man
:Peggy		:Person
:Mary	:George	
...	...	...

- + eliminates self-joins
- - null values
- - single-valued properties

# Vertical partitioning table

subject	object
:John	:Peggy
:Mary	:George
...	...

:loves

subject	object
:Peggy	:Person
:John	:Man
...	...

rdf:type

- + eliminates self-joins
- - null values
- - single-valued properties

# Mapping dictionary

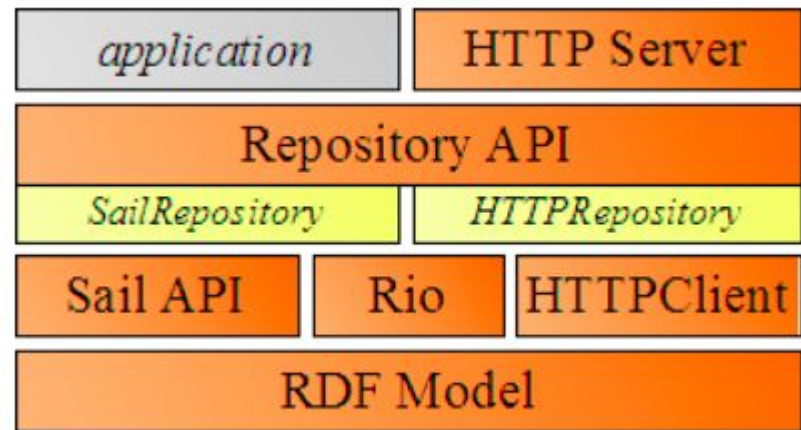
subject	predicate	object
3	1	4
4	2	5
6	1	7
3	2	8
...	...	...

id	node
:1	:loves
:2	rdf:type
:3	:John
:4	:Peggy
...	...

- + removes redundancy
- - saving space

# RDF4J-based triple store (triple table)

- **Memory Store** (speed)
  - transactional RDF database using main memory with optional persistent sync to disk.
- **Native Store** (scalability, consistency)
  - transactional RDF database using direct disk IO for persistence.
  - B-Trees
- **Elasticsearch Store** (fast for read-only scenarios)
  - RDF database that uses Elasticsearch for storage.
  - Elastic indexing



taken from <https://graphdb.ontotext.com/documentation/free/architecture-components.html>

# Inferencing

# RDF4j Inferencing

- **Full materialization**
  - upon save data inference rules are run and new triples inferred which are then stored together with original triples
- non-complete for OWL entailment regimes

subject	predicate	object	context
:John	:loves	:Peggy	:people
:loves	rdf:type	owl:Symmetric Property	:people

Id: prp\_symp

a <rdf:type> <owl:SymmetricProperty>  
b a c

-----  
c a b [Constraint a != <blank:node>]

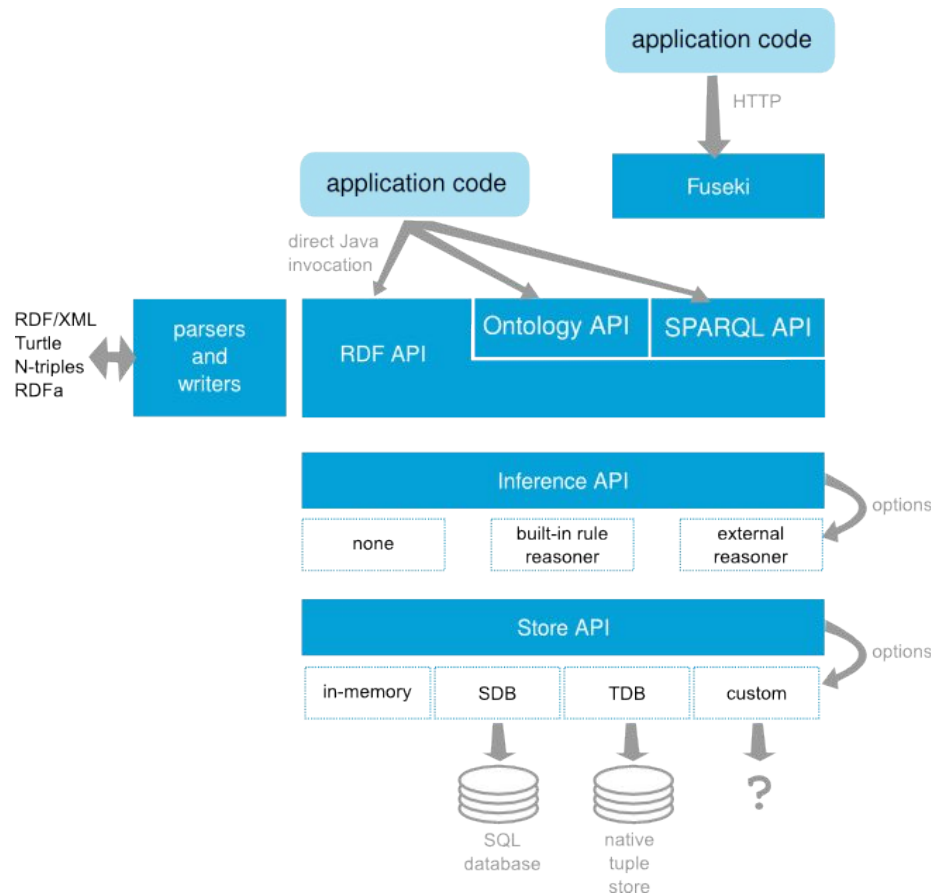
subject	predicate	object	context
:Peggy	:loves	:John	<i>explicit</i>

# Jena + Fuseki

- RDF API for processing RDF data in various notations
- Ontology API for OWL and RDFS
- Rule-based inference engine and Inference API
- TDB – a native triple store
- SPARQL query processor (ARQ).
- Fuseki – a SPARQL end-point accessible over HTTP

public example:

<https://ec.europa.eu/esco/spargl/>



# StarDog inferencing

- **Runtime Query Execution**
  - upon query execution new data are inferred
- slower for queries
- faster for updates

subject	predicate	object	context
:John	:loves	:Peggy	:people

:loves <rdf:type> <owl:SymmetricProperty>

subject	predicate	object	context
:Peggy	:loves	:John	<i>implicit</i>



# Access Control

# Access Control

- generally difficult, most systems offer RBAC only
- full data security is not solved, but approximations exist:
  - Fluree - distributed cloud triplestore -  
<https://github.com/fluree/db>
  - StarDog - property-based security -  
<https://docs.stardog.com/operating-stardog/security/fine-grained-security>

# StarDog - property-based security

- Defining *sensitive* predicates **P**
  - users with R permission to P
  - users without R permission to P
- For users without R permission to P, each SPARQL query is first prepended with the following one:

```
INSERT { ?subject ?property ?masked }  
DELETE { ?subject ?property ?object }  
WHERE {  
    ?subject ?property ?object .  
    FILTER (?property in { P }) # i.e., P is sensitive  
    BIND(mask(?object) AS ?masked)  
}
```

- This masks the value. As a side-effect it also disconnects the graph on ?object not allowing to follow the obfuscated link.

# Application access to ontologies

(Java/Kotlin)

# Low-level vs. High-level APIs

- **Low-level APIs**

- OWLAPI
- JENA
- RDF4J-API
- ....

work with individual statements

- **High-level APIs**

- JOPA
- JAQB
- ....

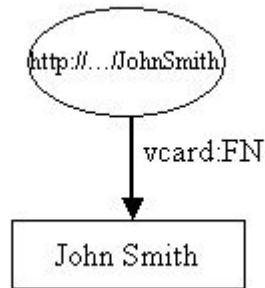
work with objects

- **Reference implementation of OWL 2**
  - complete
  - pluggable architecture for reasoners

```
OWLOntologyManager m = create();  
  
OWLOntology o = m.loadOntologyFromOntologyDocument(pizza_iri);  
for (OWLClass cls : o.getClassesInSignature()) {  
    System.out.println(cls);  
}
```

- **Long-history implementation of RDF**
  - complete
  - extended towards OWL (but incomplete support)
  - wide use

```
static String personURI = "http://somewhere/JohnSmith";  
static String fullName = "John Smith";  
  
Model model = ModelFactory.createDefaultModel();  
Resource johnSmith = model.createResource(personURI);  
johnSmith.addProperty(VCARD.FN, fullName);
```



# Java OWL persistence API (JOPA)

- Annotation-based object-ontological mapping
- Inheritance, inferred knowledge access
- Query API with automatic mapping to entities
  - JPQL-like query language also available
- Access to unmapped types and properties
- Transactions, second-level cache
- Integrity constraints
  - Mapping definition, validation of participation constraints at runtime
- Object model generator (OWL2Java)

```
8  @Namespace(prefix = "foaf", namespace = "http://xmlns.com/foaf/0.1/")
9  @OWLClass(iri = "foaf:person")
10 public class Person implements Serializable {
11
12     @Id(generated = true)
13     private URI uri;
14
15     @ParticipationConstraints(nonEmpty = true)
16     @OWLDataProperty(iri = "foaf:firstName")
17     private String firstName;
18
19     @ParticipationConstraints(nonEmpty = true)
20     @OWLDataProperty(iri = "foaf:lastName")
21     private String lastName;
22
23     @OWLObjectProperty(iri = "foaf:knows")
24     private Set<Person> acquaintances;
25
26     @Inferred
27     @Types
28     private Set<String> types;
29
30     @Properties
31     private Map<String, Set<String>> properties;
32 }
33
```

<https://github.com/kbss-cvut/jopa>