# Lecture 5



## RDF & SPARQL (Contd.)

*The slides are prepared by Dr. Davoud Mougouei*

# Lecture Outline

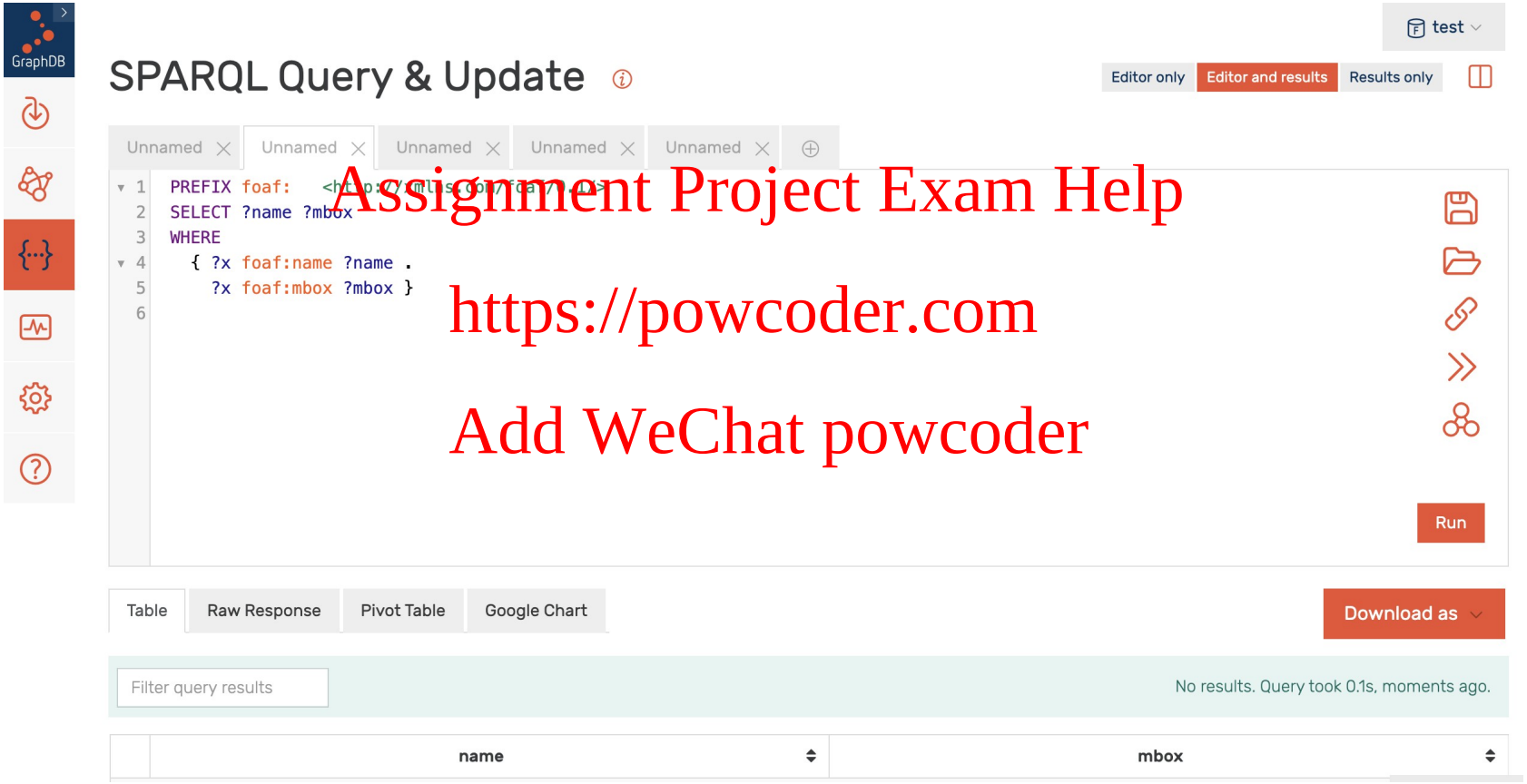
1. Programming the semantic Web
2. Working with RDF documents in Apache Jena
3. SPARQL queries in Jena
4. Remote data and local query processor
5. SPARQL endpoint
6. Understanding the semantic of SPARQL queries
7. Movie example



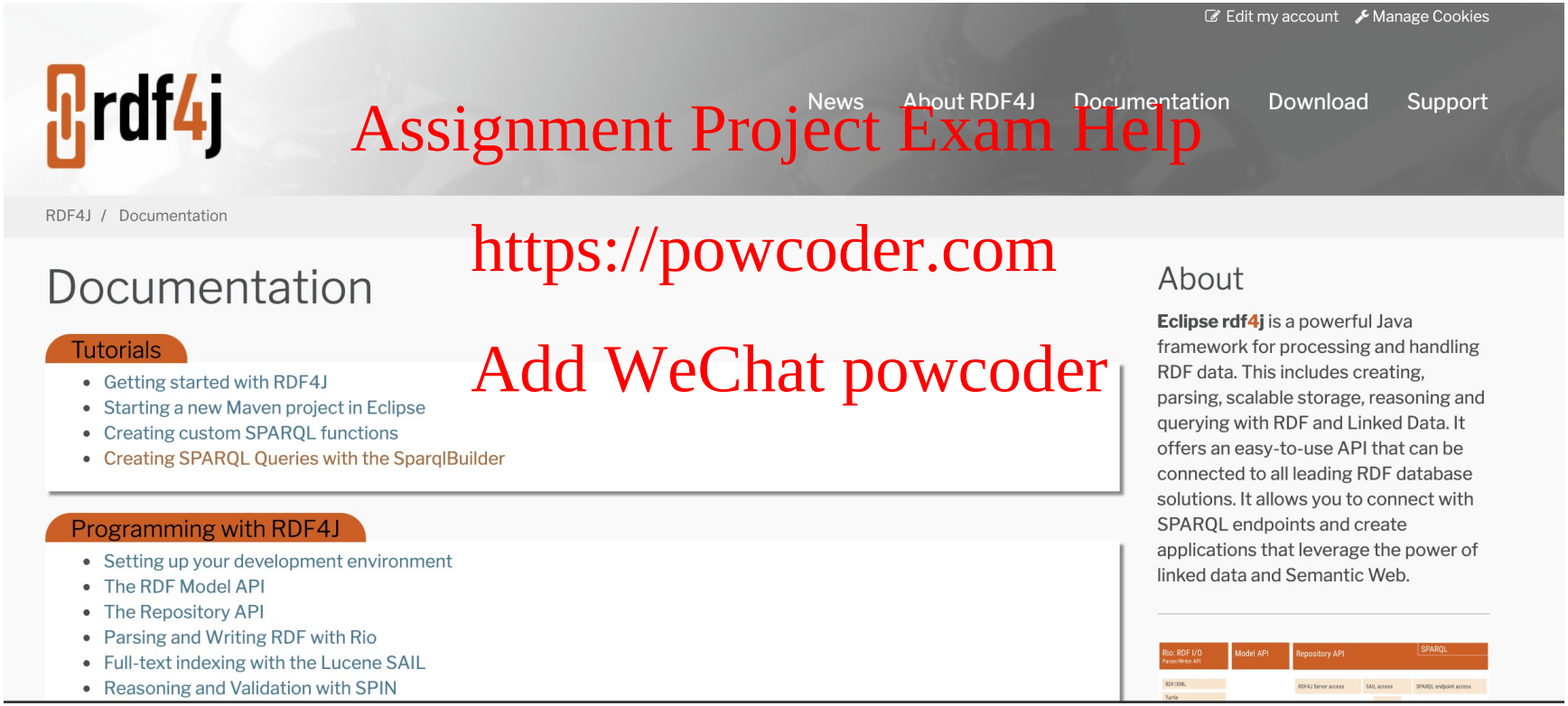
w

# Programming the Semantic Web

# GraphDB



# RDF4J

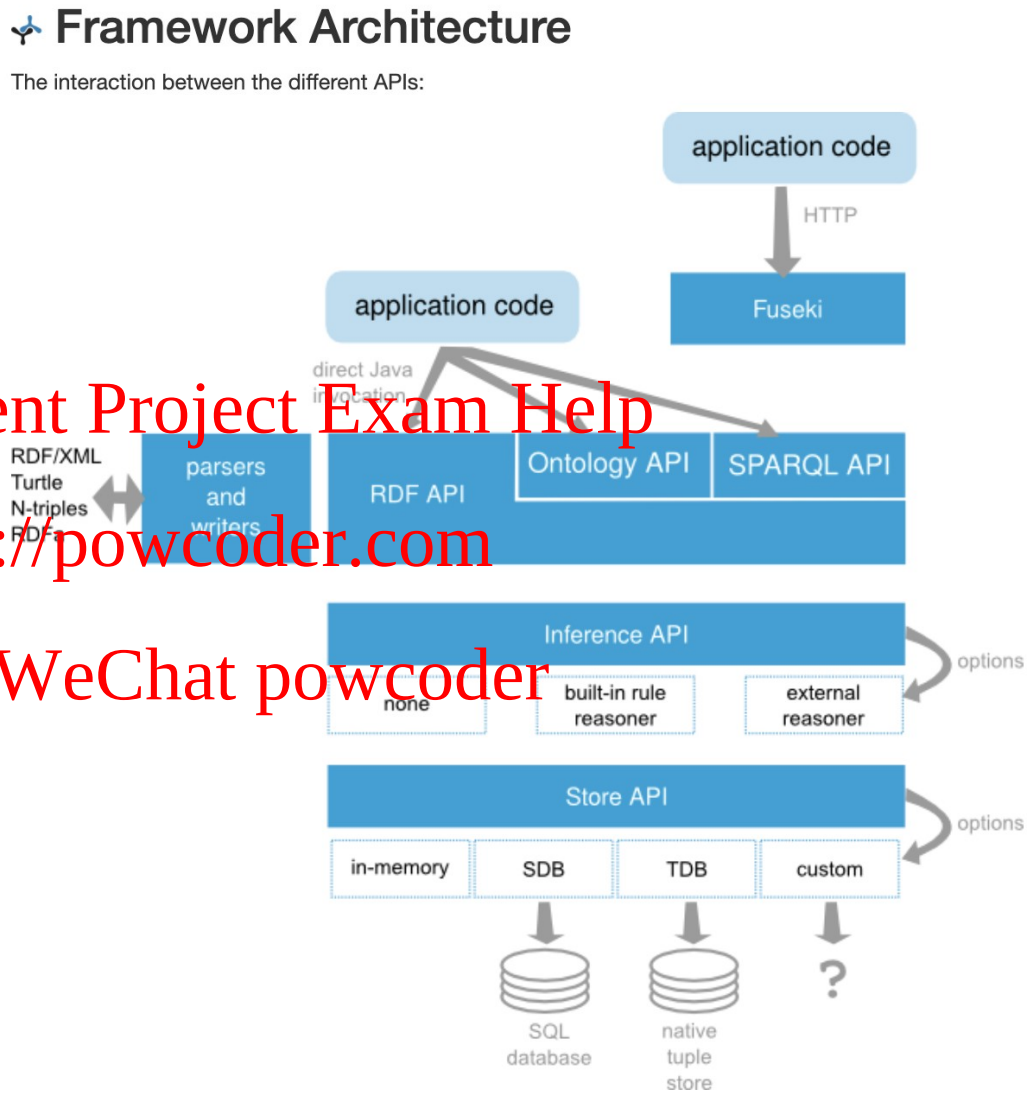





https://rdf4j.org/documentation/programming/

# Apache Jena

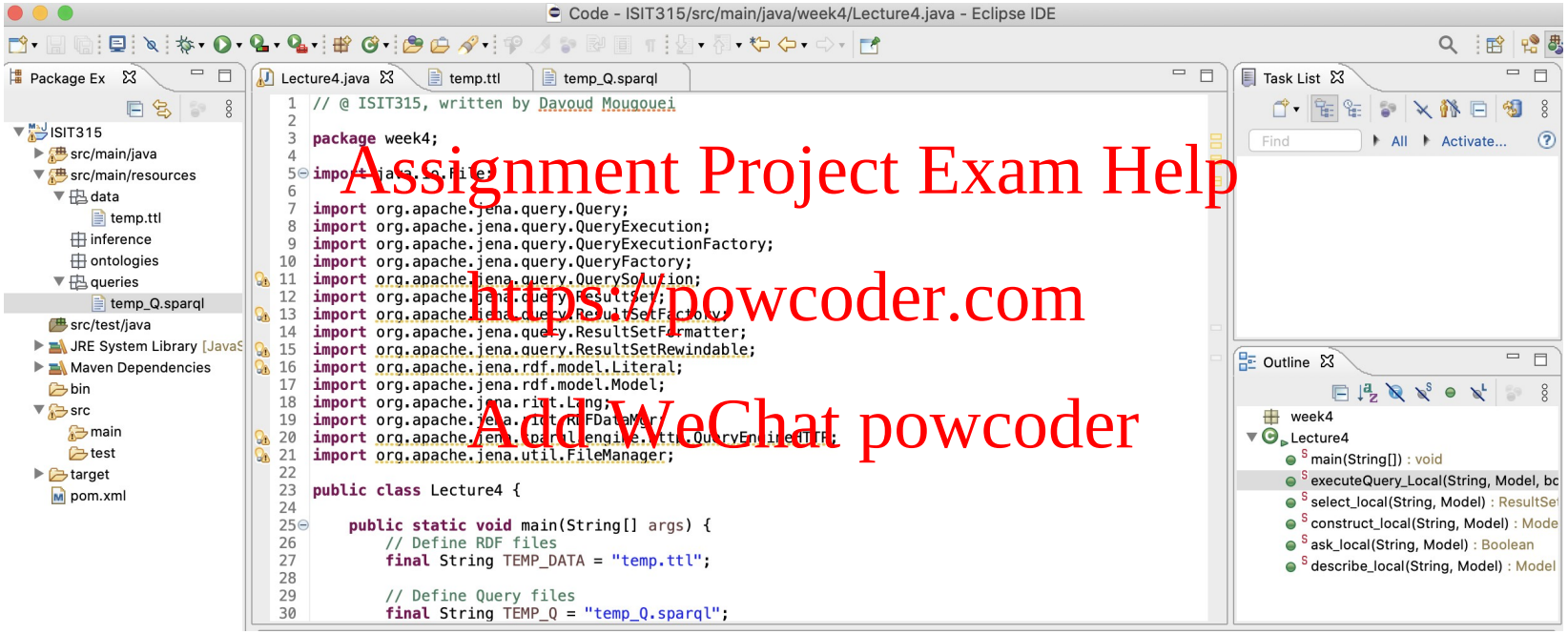
## Architecture






https://jena.apache.org/getting_started/index.html

# Getting Started



# Maven Dependencies

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>ISIT315</groupId>
  <artifactId>ISIT315</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <build>
    <sourceDirectory>src</sourceDirectory>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.0</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
  <dependencies>
   <dependency>
    <groupId>org.apache.jena</groupId>
    <artifactId>apache-jena-libs</artifactId>
    <type>pom</type>
    <version>3.16.0</version>
   </dependency>
   <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.13</version>
    </dependency>
   <dependency>
       <groupId>org.slf4j</groupId>
       <artifactId>slf4j-api</artifactId>
       <version>1.7.5</version>
   </dependency>
   <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.13.3</version>
  </dependency>
   <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.13.3</version>
  </dependency>
   <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-slf4j-impl</artifactId>
    <version>2.13.3</version>
  </dependency>
  </dependencies>
</project>
```

# Copyright (Jena Examples)

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
// modified by Davoud Mougouei
```

# Working with RDF Documents in Jena



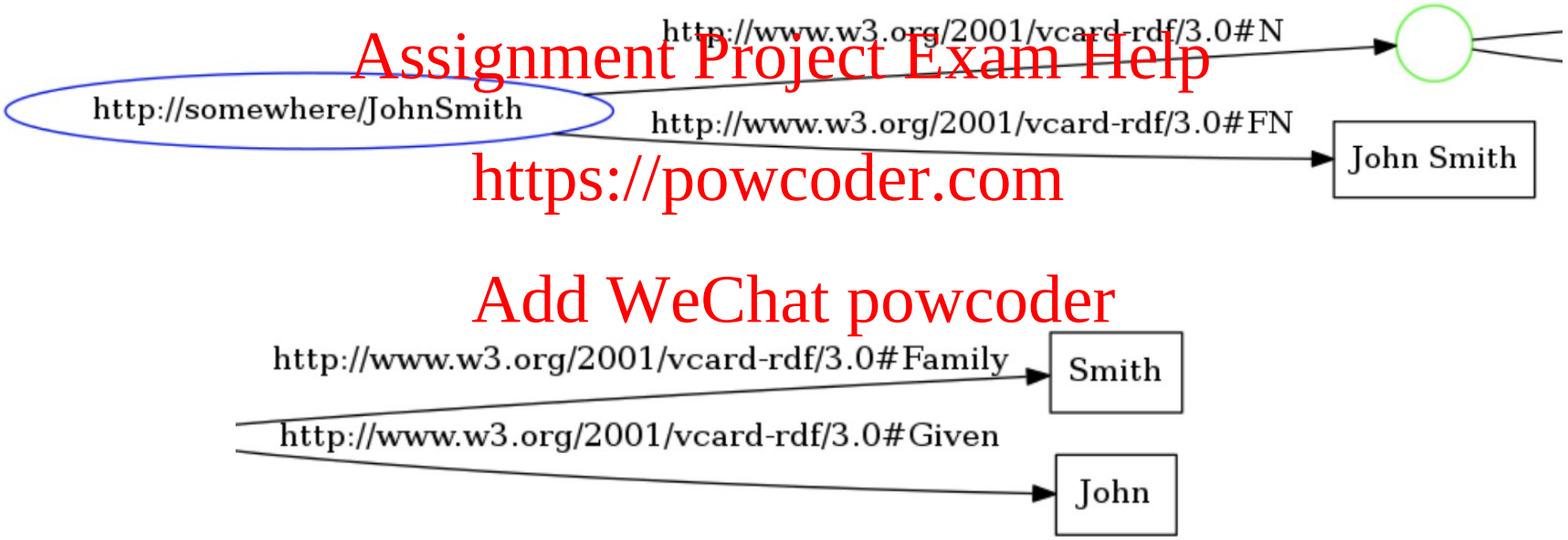
# A Simple RDF Model

```
20 package week5 ;
21
22 import org.apache.jena.rdf.model.*;
23 import org.apache.jena.vocabulary.*;
24
25 /** Tutorial 1 creating a simple model
26  */
27
28 public class Tutorial01 extends Object {
29     // some definitions
30     static String personURI    = "http://somewhere/JohnSmith";
31     static String fullName     = "John Smith";
32
33       public static void main(String args[]) {
34         // create an empty model
35         Model model = ModelFactory.createDefaultModel();
36
37        // create the resource
38        Resource johnSmith = model.createResource(personURI);
39
40       // add the property
41       johnSmith.addProperty(VCARD.FN, fullName);
42       model.write(System.out,"Turtle");
43       }
44 }
```

# Resources as Property Values

```
26 public class Tutorial02 extends Object {
27
28       public static void main (String args[]) {
29         // some definitions
30         String personURI    = "http://somewhere/JohnSmith";
31         String givenName    = "John";
32         String familyName   = "Smith";
33         String fullName     = givenName + " " + familyName;
34
35         // create an empty Model
36         Model model = ModelFactory.createDefaultModel();
37
38         // create the resource
39         //   and add the properties cascading style
40         Resource johnSmith  = model.createResource(personURI)
41              .addProperty(VCARD.FN, fullName)
42              .addProperty(VCARD.N,
43                       model.createResource()
44                            .addProperty(VCARD.Given, givenName)
45                            .addProperty(VCARD.Family, familyName));
46
47         model.write(System.out,"Turtle");
48       }
49 }
```

# Resources as Property Values



# Attribute Accessor Methods

```
30 public class Tutorial03 extends Object {
31     public static void main (String args[]) {
32 
33         // some definitions
34         String personURI    = "http://somewhere/JohnSmith";
35         String givenName    = "John";
36         String familyName   = "Smith";
37         String fullName     = givenName + " " + familyName;
38         // create an empty model
39         Model model = ModelFactory.createDefaultModel();
40 
41         // create the resource
42         //   and add the properties cascading style
43         Resource johnSmith
44           = model.createResource(personURI)
45                  .addProperty(VCARD.FN, fullName)
46                  .addProperty(VCARD.N,
47                               model.createResource()
48                                    .addProperty(VCARD.Given, givenName)
49                                    .addProperty(VCARD.Family, familyName));
50 
51         model.write(System.out,"Turtle");
52         // list the statements in the Graph
53         StmtIterator iter = model.listStatements();
54 
55         // print out the predicate, subject and object of each statement
56         while (iter.hasNext()) {
57             Statement stmt      = iter.nextStatement();         // get next statement
58             Resource  subject   = stmt.getSubject();   // get the subject
59             Property  predicate = stmt.getPredicate(); // get the predicate
60             RDFNode   object    = stmt.getObject();    // get the object
61 
62             System.out.print(subject.toString());
63             System.out.print(" " + predicate.toString() + " ");
64             if (object instanceof Resource) {
65                 System.out.print(object.toString());
66             } else {
67                 // object is a literal
68                 System.out.print(" \"" + object.toString() + "\"");
69             }
70             System.out.println(" .");
71         }
72     }
73 }
```

# Write RDF/XML into File

```
29 public class Tutorial04 extends Object {
30
31     // some definitions
32     static String tutorialURI  = "http://hostname/rdf/tutorial/";
33     static String briansName   = "Brian McBride";
34     static String briansEmail1 = "brian_mcbride@hp.com";
35     static String briansEmail2 = "brian_mcbride@hpl.hp.com";
36     static String title        = "An Introduction to RDF and the Jena API";
37     static String date         = "23/01/2001";
38     static final String PATH_DATA = System.getProperty("user.dir")
39             + File.separator + "src" + File.separator + "main"
40             + File.separator + "resources" + File.separator + "data" + File.separator + "tutorial4.rdf";
41
42     public static void main (String args[]) {
43
44         // some definitions
45         String personURI    = "http://somewhere/JohnSmith";
46         String givenName    = "John";
47         String familyName   = "Smith";
48         String fullName     = givenName + " " + familyName;
49         // create an empty model
50         Model model = ModelFactory.createDefaultModel();
51
52         // create the resource
53         //   and add the properties cascading style
54         Resource johnSmith
55           = model.createResource(personURI)
56                  .addProperty(VCARD.FN, fullName)
57                  .addProperty(VCARD.N,
58                               model.createResource()
59                                    .addProperty(VCARD.Given, givenName)
60                                    .addProperty(VCARD.Family, familyName));
61
62         // now write the model in XML form to a file
63         model.write(System.out);
64         try{
65             model.write(new FileOutputStream(PATH_DATA));
66         }catch(Exception ex) {}
67     }
68 }
```

# Read RDF XML from a File

```
29 public class Tutorial05 extends Object {
30
31     static final String PATH_DATA = System.getProperty("user.dir")
32             + File.separator + "src" + File.separator + "main"
33             + File.separator + "resources" + File.separator + "data" + File.separator + "vc-db-1.rdf";
34
35     public static void main(String args[]) {
36         // create an empty model
37         Model model = ModelFactory.createDefaultModel();
38
39         InputStream in = FileManager.get().open(PATH_DATA);
40         if (in == null) {
41             throw new IllegalArgumentException("File: " + PATH_DATA + " not found");
42         }
43
44         // read the RDF/XML file
45         model.read(in, "");
46
47         // write it to standard out
48         model.write(System.out);
49     }
50 }
```

# Navigating a Model

```
29 public class Tutorial06 extends Object {
30
31     static final String PATH_DATA = System.getProperty("user.dir")
32             + File.separator + "src" + File.separator + "main"
33             + File.separator + "resources" + File.separator + "data" + File.separator + "vc-db-1.rdf";
34
35     static final String johnSmithURI = "http://somewhere/JohnSmith";
36
37     public static void main (String args[]) {
38         // create an empty model
39         Model model = ModelFactory.createDefaultModel();
40
41         // use the FileManager to find the input file
42         InputStream in = FileManager.get().open(PATH_DATA);
43         if (in == null) {
44             throw new IllegalArgumentException( "File: " + PATH_DATA + " not found");
45         }
46
47         // read the RDF/XML file
48         model.read(new InputStreamReader(in), "");
49
50         // retrieve the Adam Smith vcard resource from the model
51         Resource vcard = model.getResource(johnSmithURI);
52
53         // retrieve the value of the N property
54         Resource name = (Resource) vcard.getRequiredProperty(VCARD.N)
55                                         .getObject();
56         // retrieve the given name property
57         String fullName = vcard.getRequiredProperty(VCARD.FN)
58                                .getString();
59         // add two nick name properties to vcard
60         vcard.addProperty(VCARD.NICKNAME, "Smithy")
61              .addProperty(VCARD.NICKNAME, "Adman");
62
63         // set up the output
64         System.out.println("The nicknames of \"" + fullName + "\" are:");
65         // list the nicknames
66         StmtIterator iter = vcard.listProperties(VCARD.NICKNAME);
67         while (iter.hasNext()) {
68             System.out.println("    " + iter.nextStatement().getObject()
69                                          .toString());
70         }
71     }
72 }
```

# Selecting the VCARD Resources

```
29 public class Tutorial07 extends Object {
30
31     static final String PATH_DATA = System.getProperty("user.dir")
32             + File.separator + "src" + File.separator + "main"
33             + File.separator + "resources" + File.separator + "data" + File.separator + "vc-db-1.rdf";
34
35     public static void main (String args[]) {
36         // create an empty model
37         Model model = ModelFactory.createDefaultModel();
38
39         // use the FileManager to find the input file
40         InputStream in = FileManager.get().open(PATH_DATA);
41         if (in == null) {
42             throw new IllegalArgumentException( "File: " + PATH_DATA + " not found");
43         }
44
45         // read the RDF/XML file
46         model.read( in, "" );
47
48         // select all the resources with a VCARD.FN property
49         ResIterator iter = model.listResourcesWithProperty(VCARD.FN);
50         if (iter.hasNext()) {
51             System.out.println("The database contains vcards for:");
52             while (iter.hasNext()) {
53                 System.out.println("  " + iter.nextResource()
54                                          .getRequiredProperty(VCARD.FN)
55                                          .getString() );
56             }
57         } else {
58             System.out.println("No vcards were found in the database");
59         }
60     }
61 }
```

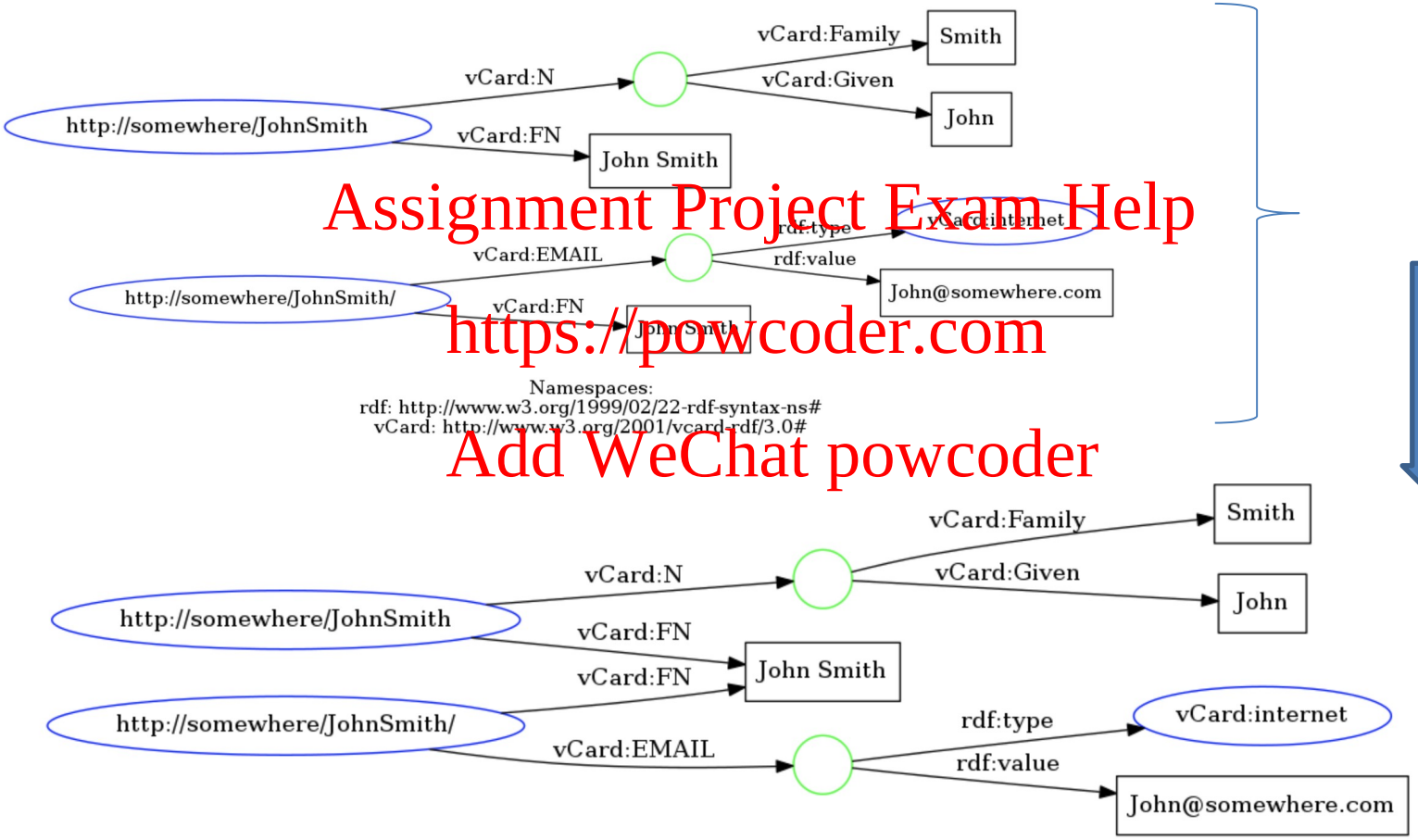
# Selector Methods

```
30 public class Tutorial08 extends Object {
31
32     static final String PATH_DATA = System.getProperty("user.dir")
33             + File.separator + "src" + File.separator + "main"
34             + File.separator + "resources" + File.separator + "data" + File.separator + "vc-db-1.rdf";
35
36     public static void main (String args[]) {
37         // create an empty model
38         Model model = ModelFactory.createDefaultModel();
39
40         // use the FileManager to find the input file
41         InputStream in = FileManager.get().open(PATH_DATA);
42         if (in == null) {
43             throw new IllegalArgumentException( "File: " + PATH_DATA + " not found");
44         }
45
46         // read the RDF/XML file
47         model.read( in, "" );
48
49         // select all the resources with a VCARD.FN property
50         // whose value ends with "Smith"
51         StmtIterator iter = model.listStatements(
52             new
53                 SimpleSelector(null, VCARD.FN, (RDFNode) null) {
54                     @Override
55                     public boolean selects(Statement s) {
56                             return s.getString().endsWith("Smith");
57                     }
58                 });
59         if (iter.hasNext()) {
60             System.out.println("The database contains vcards for:");
61             while (iter.hasNext()) {
62                 System.out.println("  " + iter.nextStatement()
63                                           .getString());
64             }
65         } else {
66             System.out.println("No Smith's were found in the database");
67         }
68     }
69 }
```

# Graph Operations

```
29 public class Tutorial09 extends Object {
30
31     static final String PATH_DATA_3 = System.getProperty("user.dir") + File.separator + "src" + File.separator + "main"
32             + File.separator + "resources" + File.separator + "data" + File.separator + "vc-db-3.rdf";
33
34     static final String PATH_DATA_4 = System.getProperty("user.dir") + File.separator + "src" + File.separator + "main"
35             + File.separator + "resources" + File.separator + "data" + File.separator + "vc-db-4.rdf";
36
37     public static void main(String args[]) {
38         // create an empty model
39         Model model1 = ModelFactory.createDefaultModel();
40         Model model2 = ModelFactory.createDefaultModel();
41
42         // use the class loader to find the input file
43         InputStream in1 = FileManager.get().open(PATH_DATA_3);
44         if (in1 == null) {
45             throw new IllegalArgumentException("File: " + PATH_DATA_3 + " not found");
46         }
47         InputStream in2 = FileManager.get().open(PATH_DATA_4);
48         if (in2 == null) {
49             throw new IllegalArgumentException("File: " + PATH_DATA_4 + " not found");
50         }
51
52         // read the RDF/XML files
53         model1.read(in1, "");
54         model2.read(in2, "");
55
56         // merge the graphs
57         Model model = model1.union(model2);
58
59         // print the graph as RDF/XML
60         model.write(System.out, "RDF/XML-ABBREV");
61         System.out.println();
62     }
63 }
64
```

# Graph Operations



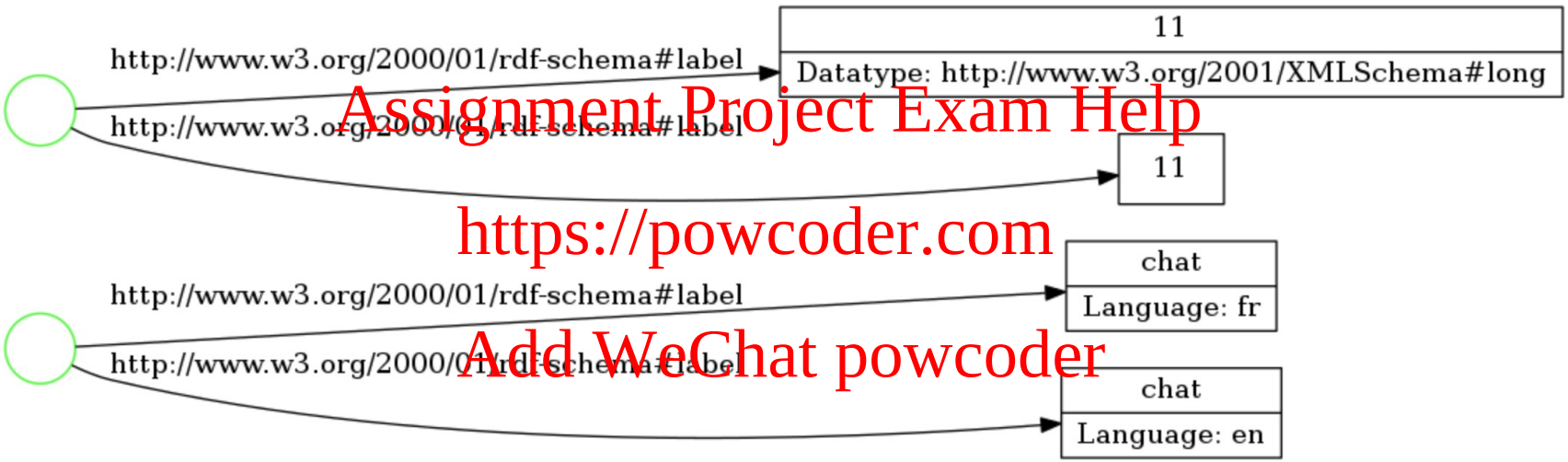
# Containers

```
38    public static void main (String args[]) {
39        // create an empty model
40        Model model = ModelFactory.createDefaultModel();
41
42        // use the class loader to find the input file
43        InputStream in = FileManager.get().open( PATH_DATA );
44        if (in == null) {
45            throw new IllegalArgumentException( "File: " + PATH_DATA + " not found");
46        }
47        // read the RDF/XML file
48        model.read(new InputStreamReader(in), "");
49        // create a bag
50        Bag smiths = model.createBag();
51
52        // select all the resources with a VCARD.FN property
53        // whose value ends with "Smith"
54        StmtIterator iter = model.listStatements(
55            new
56                SimpleSelector(null, VCARD.FN, (RDFNode) null) {
57                    @Override
58                    public boolean selects(Statement s) {
59                            return s.getString().endsWith("Smith");
60                    }
61                });
62        // add the Smith's to the bag
63        while (iter.hasNext()) {
64            smiths.add(iter.nextStatement().getSubject());
65        }
66        // print the graph as RDF/XML
67        model.write(new PrintWriter(System.out));
68        System.out.println();
69        // print out the members of the bag
70        NodeIterator iter2 = smiths.iterator();
71        if (iter2.hasNext()) {
72            System.out.println("The bag contains:");
73            while (iter2.hasNext()) {
74                System.out.println("  " +
75                    ((Resource) iter2.next())
76                                .getRequiredProperty(VCARD.FN)
77                                .getString());
78            }
79        } else {
80            System.out.println("The bag is empty");
81        }
82    }
83 }
```

# Literals

```
29 public class Tutorial11 extends Object {
30
31     public static void main(String args[]) {
32         // create an empty graph
33         Model model = ModelFactory.createDefaultModel();
34
35         // create the resource
36         Resource r = model.createResource();
37
38         // add the property
39         r.addProperty(RDFS.label, model.createLiteral("chat", "en"))
40                 .addProperty(RDFS.label, model.createLiteral("chat", "fr"));
41
42         // write out the graph
43         model.write(new PrintWriter(System.out), "Turtle");
44         System.out.println();
45
46         // create an empty graph
47         model = ModelFactory.createDefaultModel();
48
49         // create the resource
50         r = model.createResource();
51
52         // add the property
53         r.addProperty(RDFS.label, "11").addLiteral(RDFS.label, 11);
54
55         // write out the graph
56         model.write(System.out, "Turtle");
57     }
58 }
```

# Literals

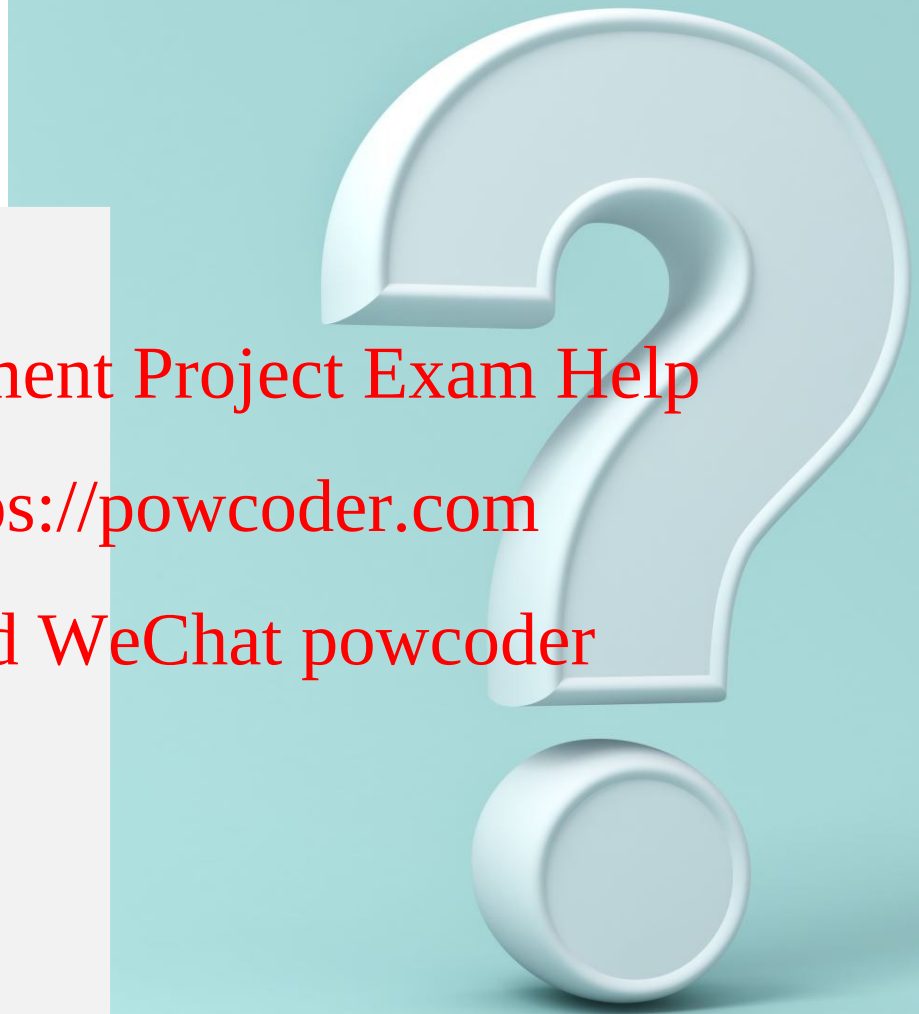





[ <http://www.w3.org/2000/01/rdf-schema#label>
"11"^^<http://www.w3.org/2001/XMLSchema#long> , "11" ] .

[ <http://www.w3.org/2000/01/rdf-schema#label> "chat"@fr , "chat"@en ] .

# SPARQL Queries in Jena



# SPARQL Queries

## Libraries

```
import org.apache.jena.query.Query;
import org.apache.jena.query.QueryExecution;
import org.apache.jena.query.QueryExecutionFactory;
import org.apache.jena.query.QueryFactory;
import org.apache.jena.query.QuerySolution;
import org.apache.jena.query.ResultSet;
import org.apache.jena.query.ResultSetFactory;
import org.apache.jena.query.ResultSetFormatter;
import org.apache.jena.query.ResultSetRewindable;
import org.apache.jena.rdf.model.Literal;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.riot.Lang;
import org.apache.jena.riot.RDFDataMgr;
import org.apache.jena.sparql.engine.http.QueryEngineHTTP;
import org.apache.jena.util.FileManager;
```

# SPARQL Queries

## Select (local)

```
public static ResultSet select_local(String query_string, Model model) {
    ResultSet results = null;
    Query query = QueryFactory.create(query_string);
    QueryExecution qexec = QueryExecutionFactory.create(query, model);
    try {
        results = qexec.execSelect();
    } finally {
        if (results != null)
            ResultSetFormatter.out(System.out, results, query);
        else
            System.out.println("no results were found!");
        qexec.close();
    }
    return results;
}
```

# SPARQL Queries

## Select (remote)

```
public static ResultSet select_remote(String query_string, String ENDPOINT) {
    ResultSet results = null;
    Query query = QueryFactory.create(query_string);
    QueryExecution qexec = QueryExecutionFactory.sparqlService(ENDPOINT, query);
    try {
        results = qexec.execSelect();
    } finally {
        if (results != null)
            ResultSetFormatter.out(System.out, results, query);
        else
            System.out.println("No results were found");
        qexec.close();
    }
    return results;
}
```



```
String ENDPOINT = "http://dbpedia.org/sparql";
```

# SPARQL Queries

## Construct (local)

```
public static Model construct_local(String query_string, Model model) {
    Model graph = null;
    Query query = QueryFactory.create(query_string);
    QueryExecution qexec = QueryExecutionFactory.create(query, model);
    try {
        graph = qexec.execConstruct();
    } finally {
        qexec.close();
    }
    return graph;
}
```

# SPARQL Queries

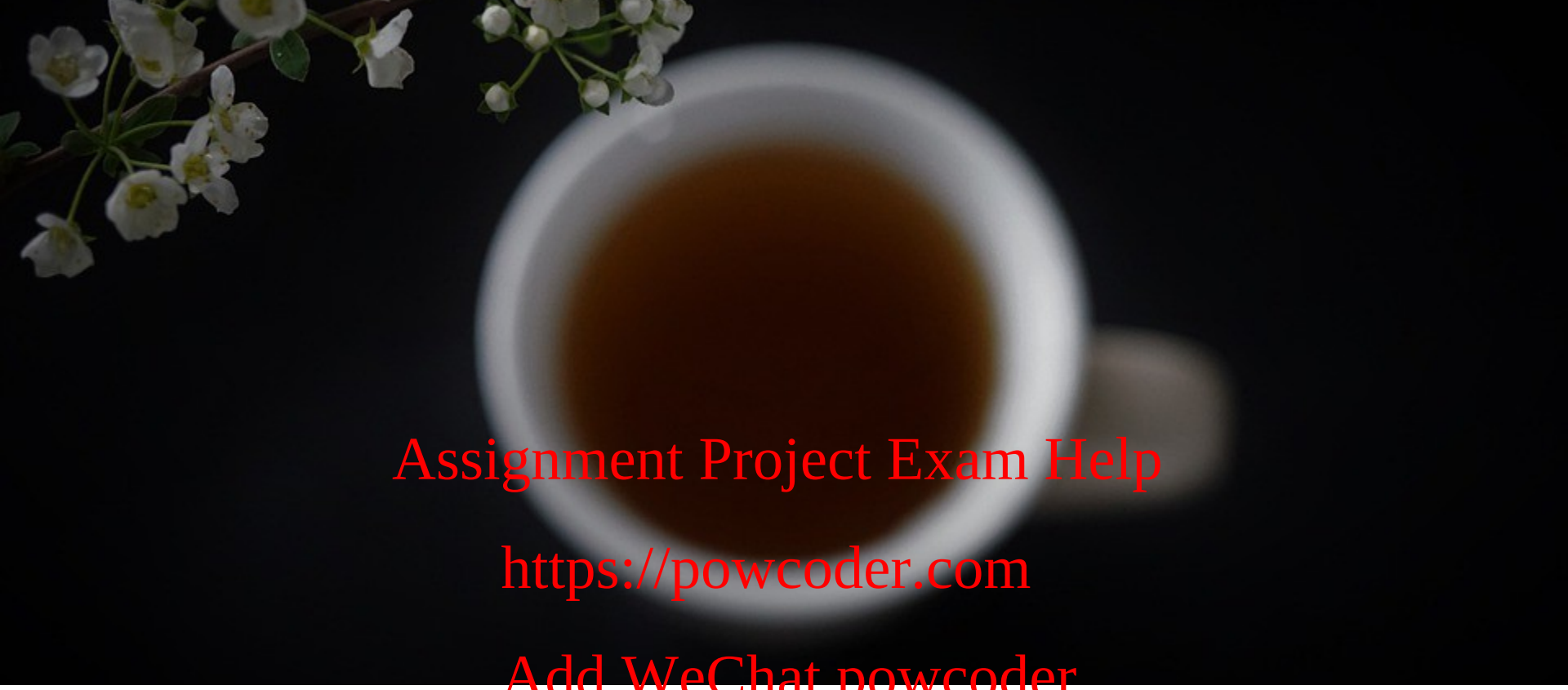
## Ask (local)

```
public static Boolean ask_local(String query_string, Model model) {
    Boolean results = null;
    Query query = QueryFactory.create(query_string);
    QueryExecution qexec = QueryExecutionFactory.create(query, model);
    try {
        results = qexec.execAsk();
    } finally {
        qexec.close();
    }
    return results;
}
```

# SPARQL Queries

## Describe (local)

```
public static Model describe_local(String query_string, Model model) {
    Model results = null;
    Query query = QueryFactory.create(query_string);
    QueryExecution qexec = QueryExecutionFactory.create(query, model);
    try {
        results = qexec.execDescribe();
    } finally {
        qexec.close();
    }
    return results;
}
```





Take a break ☺



# Remote Data & Local Query Processor

- Data is accessed via URI
- Query is processed using Jena

# SPARQL Queries

## Remote Data & Local Query Processor

Data:



http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf



**Query:**



```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT * WHERE {
    ?s foaf:name ?name . FILTER regex (?name,"^A" , "i")
} limit 50
```

**Results:**

???



Write a query that finds the names that start with either A or B and end with C.

# SPARQL Queries

## Remote Data & Local Query Processor

Data:

http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf



**Query:**

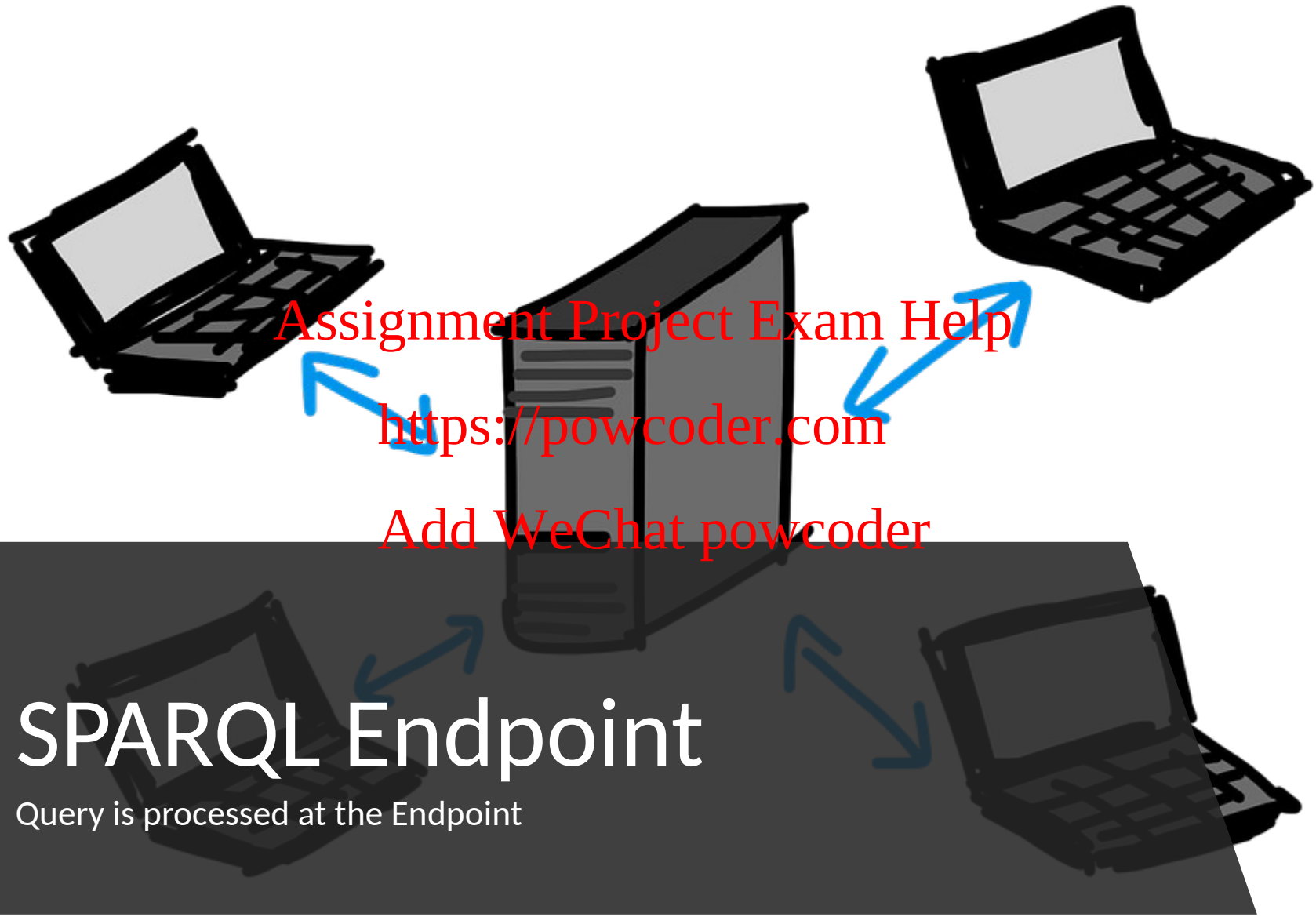


```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
PREFIX card: <http://www.w3.org/People/Berners-Lee/card#>
SELECT ?homepage
WHERE {
   card:i foaf:knows ?known .
   ?known foaf:homepage ?homepage .
}
```



**Results:**

???



# SPARQL Endpoint

Query is processed at the Endpoint

# SPARQL Queries

## SPARQL Endpoint

| Endpoint: |
| --- |
| http://dbpedia.org/sparql |



| **Prefixes:** |
| --- |



```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX : <http://dbpedia.org/resource/>
PREFIX dbpedia2: <http://dbpedia.org/property/>
PREFIX dbpedia: <http://dbpedia.org/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX dbo: <http://dbpedia.org/ontology/>
```

# SPARQL Queries

## SPARQL Endpoint

Semantic:

People who were born in Sydney before 1985



**Query:**



```
SELECT ?name ?birth ?death ?person
WHERE {
  ?person dbo:birthPlace :Sydney .
  ?person dbo:birthDate ?birth .
  ?person foaf:name ?name .
  ?person dbo:deathDate ?death .
  FILTER (?birth < ”1985-01-01"^^xsd:date) .
}  ORDER BY ?name LIMIT 50
```



**Results:**

???

# SPARQL Queries

## SPARQL Endpoint

Semantic:

Musicians with German and English descriptions.

**Query:**



```
SELECT ?name ?description_en ?description_de ?musician
WHERE {
 ?musician foaf:name ?name .
 ?musician a dbo:MusicalArtist .
  OPTIONAL { ?musician rdfs:comment ?description_en .
      FILTER (LANG(?description_en) = 'en') . }
  OPTIONAL { ?musician rdfs:comment ?description_de .
      FILTER (LANG(?description_de) = 'de') . }
} LIMIT 50
```





**Results:**

???

# SPARQL Queries

## SPARQL Endpoint

**Semantic:**

Musicians who were born in Berlin.



**Query:**



```
SELECT ?name ?birth ?description ?person WHERE {
      ?person a dbo:MusicalArtist .
      ?person dbo:birthPlace :Berlin .
      ?person dbo:birthDate ?birth .
      ?person foaf:name ?name .
      ?person rdfs:comment ?description .
      FILTER (LANG(?description) = 'en') .
} ORDER BY ?name LIMIT 50
```



**Results:**

???

# SPARQL Queries

## SPARQL Endpoint

**Semantic:**

Soccer players who are born in a country with more than 10 million inhabitants, who played as goalkeeper for a club that has a stadium with more than 30,000 seats and the club country is different from the birth country.

**Query:**



```
SELECT distinct ?soccerplayer ?countryOfBirth ?team ?countryOfTeam ?stadiumcapacity
{
  ?soccerplayer a dbo:SoccerPlayer ;
  dbo:position <http://dbpedia.org/resource/Goalkeeper_(association_football)> ;
  dbo:birthPlace/dbo:country* ?countryOfBirth ;
  #dbo:number 13 ;
  dbo:team ?team .
  ?team dbo:capacity ?stadiumcapacity ; dbo:ground ?countryOfTeam .
  ?countryOfBirth a dbo:Country ; dbo:populationTotal ?population .
  ?countryOfTeam a dbo:Country .
FILTER (?countryOfTeam != ?countryOfBirth)
FILTER (?stadiumcapacity > 30000)
FILTER (?population > 10000000)
} order by ?soccerplayer
```





**Results:**

???

# Understanding the Semantic of SPARQL Queries

# SPARQL Queries

## Query Semantic

Query :

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/> PREFIX res:
<http://dbpedia.org/resource/>
SELECT DISTINCT ?string WHERE { res:Tom_Cruise foaf:homepage ?string . }
```

**Semantic:**

???

**Results:**

???

# SPARQL Queries

## Query Semantic

| Query : |
| --- |
| PREFIX dbo: <http://dbpedia.org/ontology/><br>PREFIX res: <http://dbpedia.org/resource/><br>PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#><br>PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#><br>SELECT DISTINCT ?uri ?string WHERE {<br>?uri rdf:type dbo:Film . ?uri dbo:starring res:Julia_Roberts . ?uri dbo:starring res:Richard_Gere.<br>OPTIONAL {?uri rdfs:label ?string . FILTER (lang(?string) = 'en') }<br>} LIMIT 50 |
| **Semantic:** |
| ??? |
| **Results:** |
| ??? |

# SPARQL Queries

## Query Semantic

| Query : |
| --- |
| PREFIX dbo: <http://dbpedia.org/ontology/><br>PREFIX res: <http://dbpedia.org/resource/><br>PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#><br>PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#><br>SELECT DISTINCT ?uri ?string WHERE {<br>?uri rdf:type dbo:Book . ?uri dbo:author res:Danielle_Steel .<br>OPTIONAL { ?uri rdfs:label ?string . FILTER (lang(?string) = 'en') }<br>} LIMIT 50 |
| **Semantic:** |
| ??? |
| **Results:** |
| ??? |

# SPARQL Queries

## Query Semantic

Query :

```
PREFIX res: <http://dbpedia.org/resource/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?uri ?string WHERE {
res:WikiLeaks dbp:awards ?uri .
OPTIONAL { ?uri rdfs:label ?string. FILTER (lang(?string) = 'en') }
} LIMIT 50
```



**Semantic:**

???

**Results:**

???

# SPARQL Queries

## Query Semantic

Query :

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX res: <http://dbpedia.org/resource/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?uri ?string WHERE {
res:Nile dbo:sourceCountry ?uri . OPTIONAL {
?uri rdfs:label ?string. FILTER (lang(?string) = 'en') }
} LIMIT 50
```



**Semantic:**

???

**Results:**

???

# SPARQL Queries

## Query Semantic

Query :

```
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX res: <http://dbpedia.org/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?uri ?string WHERE {
?uri rdf:type dbo:Company . ?uri dbp:industry ?industry . FILTER regex(?
industry,'advertising','i') . OPTIONAL {?uri rdfs:label ?string. FILTER (lang(?string) =
'en') }
} LIMIT 50
```

**Semantic:**

???

**Results:**

???

# SPARQL Queries

## Query Semantic

**Query :**

```
PREFIX res: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?uri ?string WHERE { res:Bruce_Carver dbo:deathCause ?uri .
OPTIONAL {?uri rdfs:label ?string. FILTER (lang(?string) = 'en') }
} LIMIT 50
```



**Semantic:**

???

**Results:**

???

# SPARQL Queries

## Query Semantic

| Query : |
| --- |
| PREFIX dbo: <http://dbpedia.org/ontology/><br>PREFIX res: <http://dbpedia.org/resource/><br>PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#><br>PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#><br>SELECT DISTINCT ?uri ?string WHERE {<br>?uri rdf:type dbo:Airport . ?uri dbo:location res:California .<br>OPTIONAL { ?uri rdfs:label ?string . FILTER (lang(?string) = 'en') }<br>} LIMIT 50 |
| **Semantic:** |
| ??? |
| **Results:** |
| ??? |

# SPARQL Queries

## Query Semantic

| Query : |
| --- |
| PREFIX dbo: <http://dbpedia.org/ontology/><br>PREFIX res: <http://dbpedia.org/resource/><br>PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#><br>PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#><br>SELECT DISTINCT ?uri ?string WHERE {<br>?uri rdf:type dbo:Film. ?uri dbo:starring res:Tom_Cruise .<br>OPTIONAL {?uri rdfs:label ?string . FILTER (lang(?string) = 'en') }<br>} LIMIT 50 |
| **Semantic:** |
| ??? |
| **Results:** |
| ??? |

# SPARQL Queries

## Query Semantic

Query :

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX res: <http://dbpedia.org/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?uri ?string WHERE {
?company rdf:type dbo:Organisation .
?company dbo:foundationPlace res:California .
?uri dbo:developer ?company .
?uri rdf:type dbo:Software . OPTIONAL { ?uri rdfs:label ?string . FILTER (lang(?string)
= 'en') } } LIMIT 50
```



**Semantic:**

???

**Results:**

???

# SPARQL Queries

## Query Semantic

Query :

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX res: <http://dbpedia.org/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT COUNT(DISTINCT ?uri) WHERE {
?uri rdf:type dbo:Film .
?uri dbo:starring res:Leonardo_DiCaprio .
} LIMIT 50
```



**Semantic:**

???

**Results:**

???

# Movies Example

# Movies Example

## Prefixes



```
@prefix fn: <http://www.w3.org/2005/xpath-functions#> .
@prefix mov: <http://example.com/movie/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix sesame: <http://www.openrdf.org/schema/sesame#> .
@prefix spif: <http://spinrdf.org/spif#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
```

# Movies Example

## RDF (TTL)

```
mov:Movie1 a mov:Movie;
  mov:Name "Movie1";
  mov:actor_1_facebook_likes "1000";
  mov:actor_1_name "CCH Pounder";
  mov:actor_2_facebook_likes "936";
  mov:actor_2_name "Joel David Moore";
  mov:actor_3_facebook_likes "855";
  mov:actor_3_name "Wes Studi";
  mov:aspect_ratio "1.78";
  mov:budget "237000000";
  mov:cast_total_facebook_likes "4834";
  mov:color "Color";
  mov:content_rating "PG-13";
  mov:country "USA";
  mov:director_facebook_likes "0";
  mov:director_name "James Cameron";
  mov:duration "178";
  mov:facenumber_in_poster "0";
  mov:genres "Action|Adventure|Fantasy|Sci-Fi";
  mov:gross "760505847";
  mov:imdb_score "7.9";
  mov:language "English";
  mov:movie_facebook_likes "33000";
  mov:movie_imdb_link <http://www.imdb.com/title/tt0499549/?ref_=fn_tt_tt_1>;
  mov:movie_title "Avatar ";
  mov:num_critic_for_reviews "723";
  mov:num_user_for_reviews "3054";
  mov:num_voted_users "886204";
  mov:plot_keywords "avatar|future|marine|native|paraplegic";
  mov:title_year "2009" .
```

# Movies Example

## SPARQL Query

Query :

```
BASE <http://example.com/movie/>
PREFIX xs: <http://www.w3.org/2001/XMLSchema#>
SELECT DISTINCT ?p{
  ?s ?p ?o .
}
```

**Semantic:**

???

**Results:**

???

# Movies Example

## SPARQL Query

Query :

```
BASE <http://example.com/movie/>
PREFIX xs: <http://www.w3.org/2001/XMLSchema#>
SELECT ?m ?c WHERE {
   ?m <country> ?c FILTER (?c = "Australia").
}
```



**Semantic:**

???

**Results:**

???

# Movies Example

## SPARQL Query

Query :

```
BASE <http://example.com/movie/>
PREFIX xs: <http://www.w3.org/2001/XMLSchema#>
SELECT  (COUNT(?c) as ?ct) WHERE  {
  ?movie <country> ?c .
  ?movie <genres> ?g FILTER regex(?g,"Horror").
} GROUP BY ?c
```



**Semantic:**

???

**Results:**

???

# Movies Example

## SPARQL Query

Query :

```
BASE <http://example.com/movie/>
PREFIX xs: <http://www.w3.org/2001/XMLSchema#>
SELECT * {
    ?m <imdb_score> ?score
} ORDER BY DESC(?score) OFFSET 0 LIMIT 5
```

**Semantic:**

???

**Results:**

???

# Movies Example

## SPARQL Query

Query :

```
BASE <http://example.com/movie/>
PREFIX xs: <http://www.w3.org/2001/XMLSchema#>
SELECT ?c (AVG(xs:integer(?s)) as ?av) WHERE {
  ?m <imdb_score> ?s .
  ?m <country> ?c .
} GROUP BY (?c)
```



**Semantic:**

???

**Results:**

???

# Movies Example

## SPARQL Query

Query :

```
BASE <http://example.com/movie/>
PREFIX xs: <http://www.w3.org/2001/XMLSchema#>
SELECT ?c (SUM(xs:float(?d)) as ?total) WHERE {
  ?m <duration> ?d .
  ?m <country> ?c .
} GROUP BY (?c)
```



**Semantic:**

???

**Results:**

???

# Resources



- http://www.w3.org/TR/rdf-sparql-query/
- https://jena.apache.org/
- https://aifb-ls3-kos.aifb.kit.edu/projects/spartiqulator/examples.htm
- Chapter 3 of Semantic Web Primer