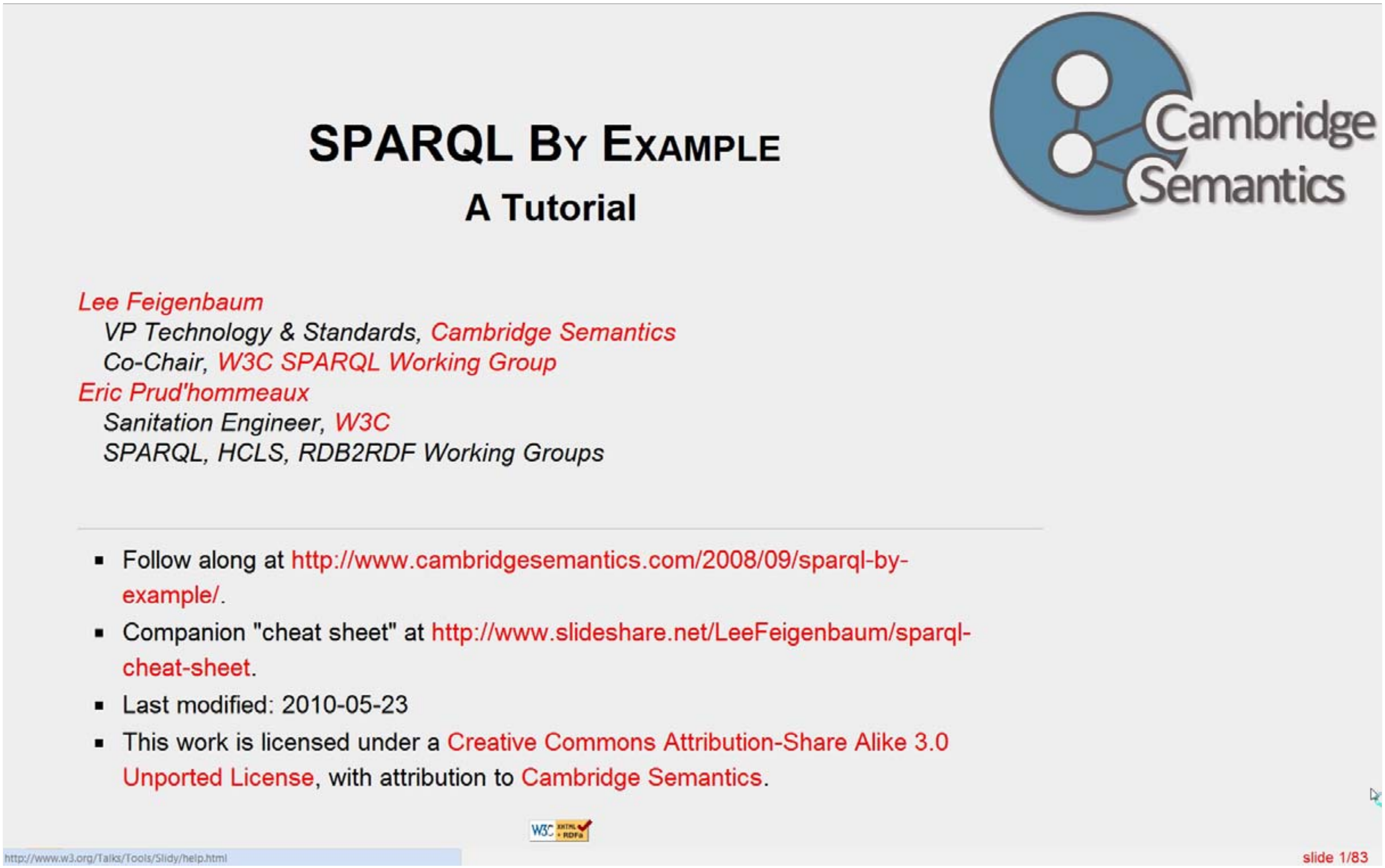
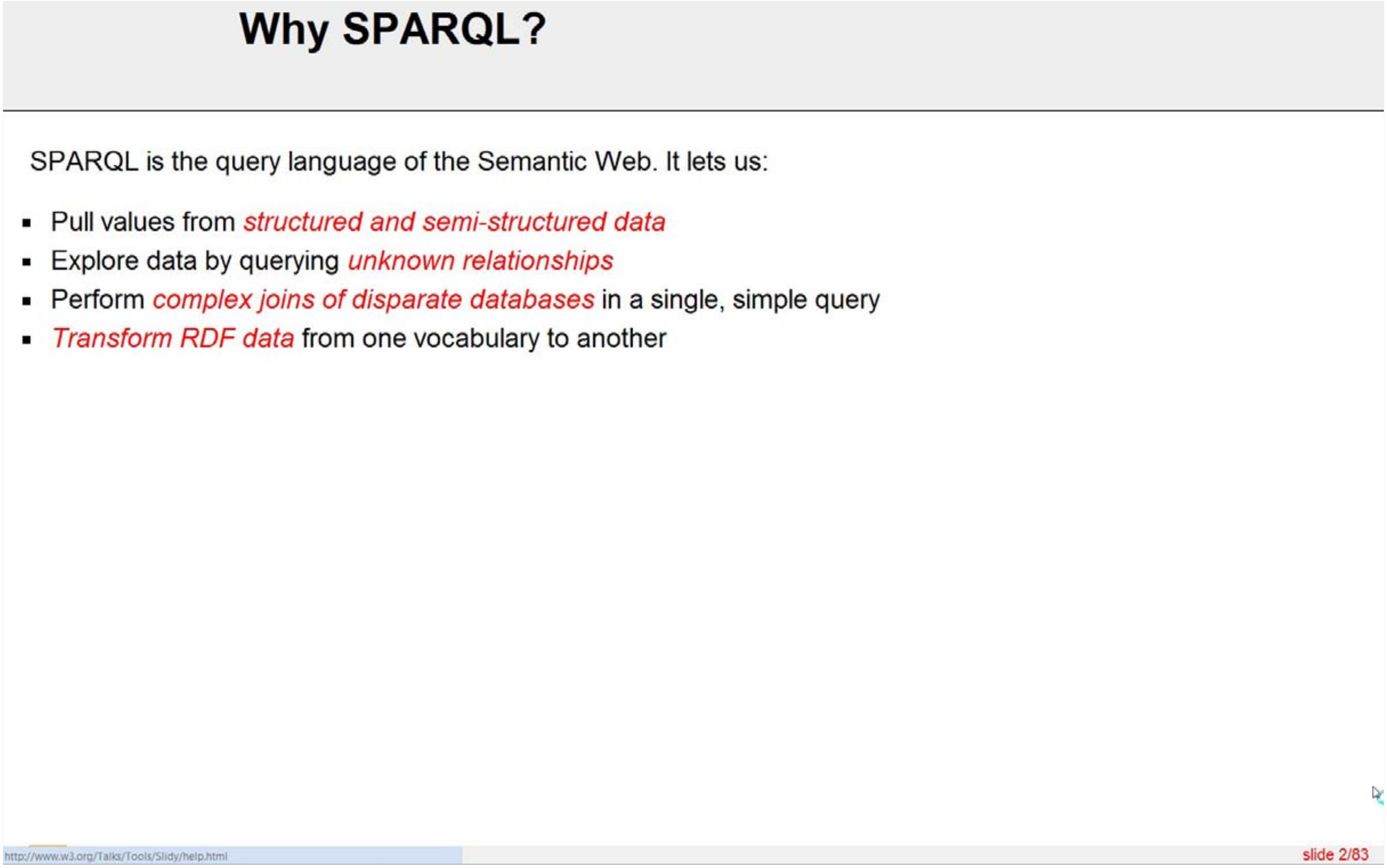


SPARQL By Example

A Tutorial

Lee Feigenbaum
VP Technology & Standards, Cambridge Semantics
Co-Chair, W3C SPARQL Working Group
Eric Prud'hommeaux
Sanitation Engineer, W3C
SPARQL, HCLS, RDB2RDF Working Groups

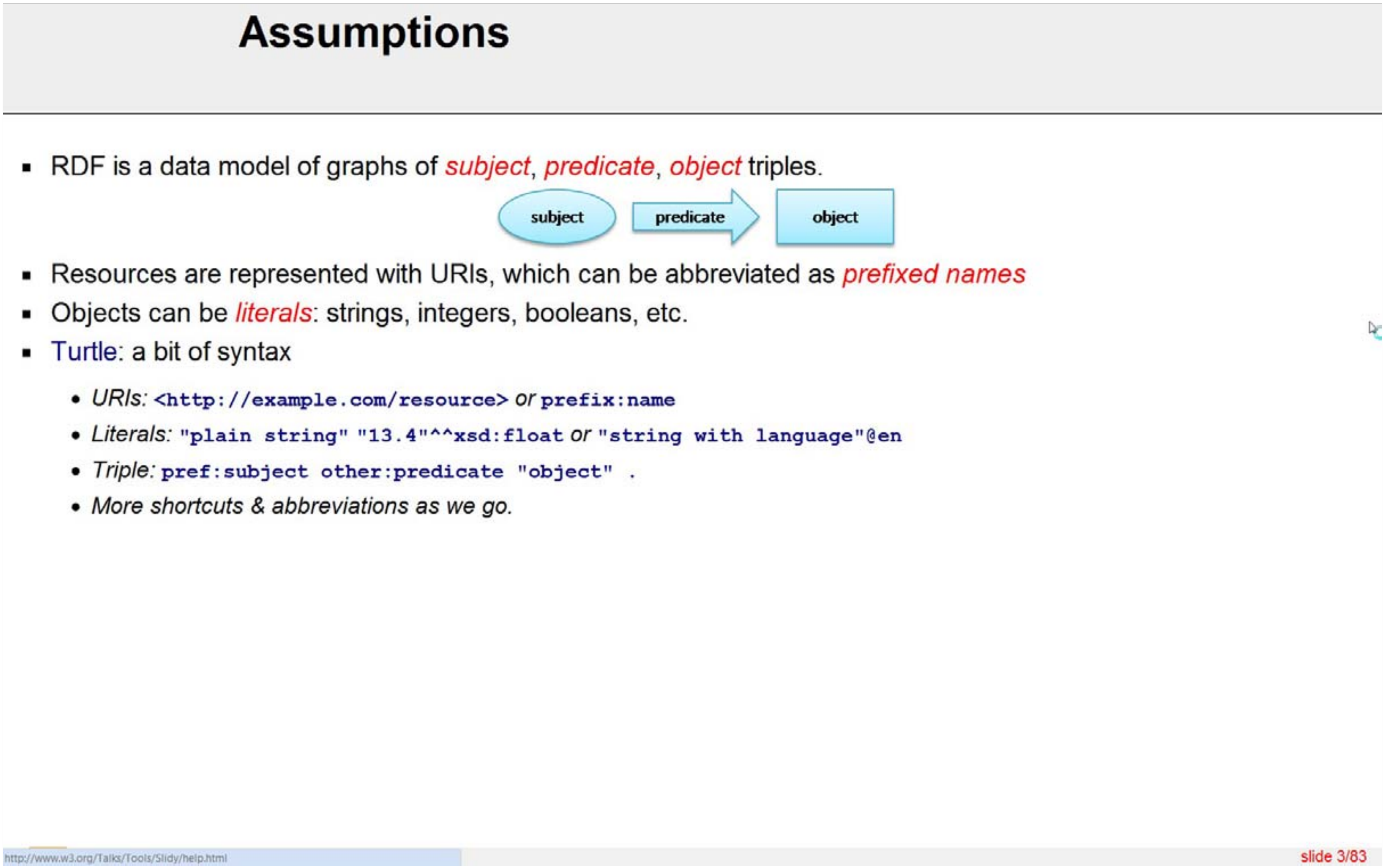
- Follow along at http://www.cambridgesemantics.com/2008/09/sparql-by-example/.
- Companion "cheat sheet" at http://www.slideshare.net/LeeFeigenbaum/sparql-cheat-sheet.
- Last modified: 2010-05-23
- This work is licensed under a Creative Commons Attribution-Share Alike 3.0 Unported License, with attribution to Cambridge Semantics.



Why SPARQL?

SPARQL is the query language of the Semantic Web. It lets us:

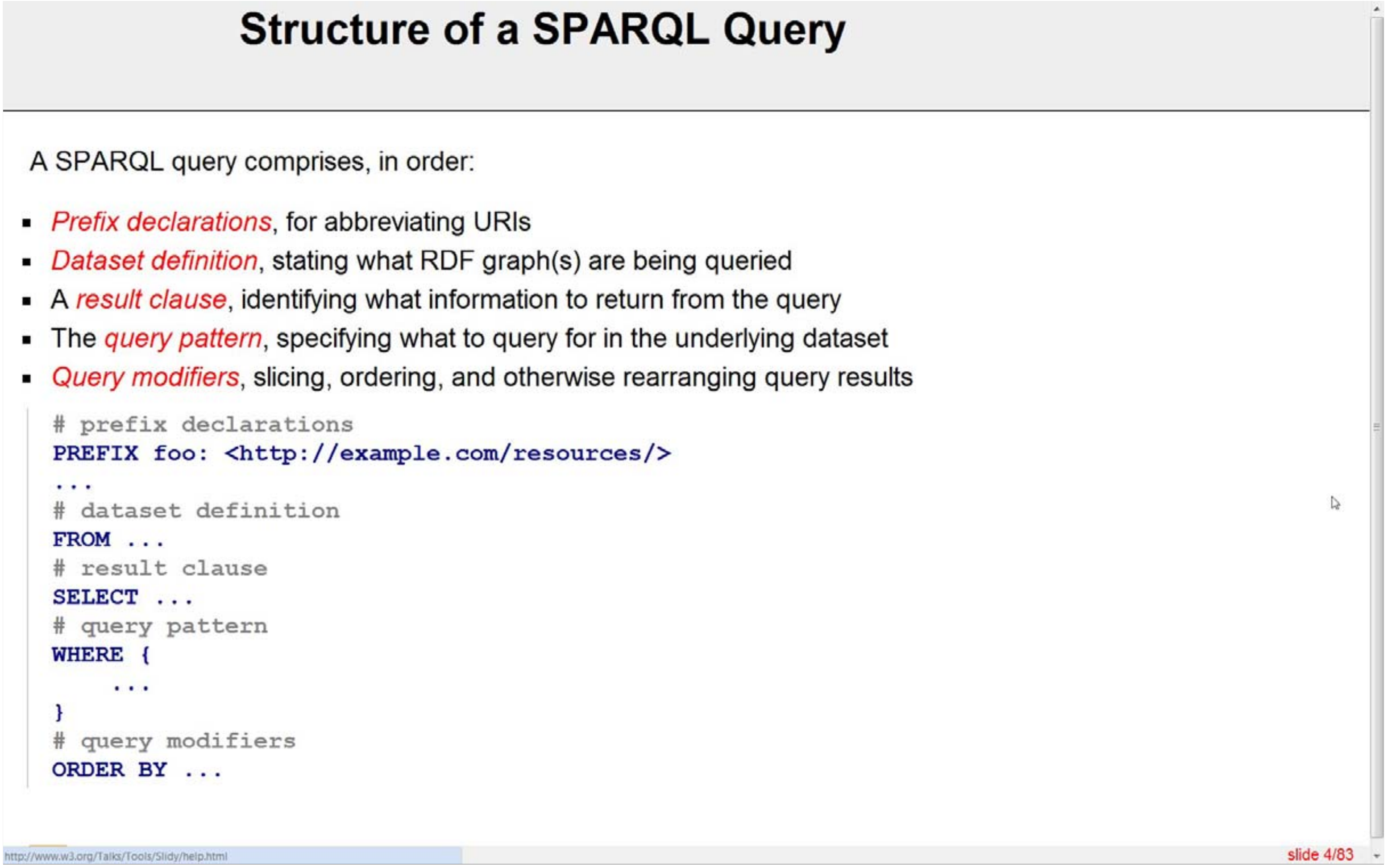
- Pull values from *structured and semi-structured data*
- Explore data by querying *unknown relationships*
- Perform *complex joins of disparate databases* in a single, simple query
- *Transform RDF data* from one vocabulary to another



Assumptions

- RDF is a data model of graphs of *subject*, *predicate*, *object* triples.

- Resources are represented with URIs, which can be abbreviated as *prefixed names*
- Objects can be *literals*: strings, integers, booleans, etc.
- Turtle: a bit of syntax
  - *URIs:* `<http://example.com/resource>` *or* `prefix:name`
  - *Literals:* `"plain string" "13.4"^^xsd:float` *or* `"string with language"@en`
  - *Triple:* `pref:subject other:predicate "object" .`
  - *More shortcuts & abbreviations as we go.*

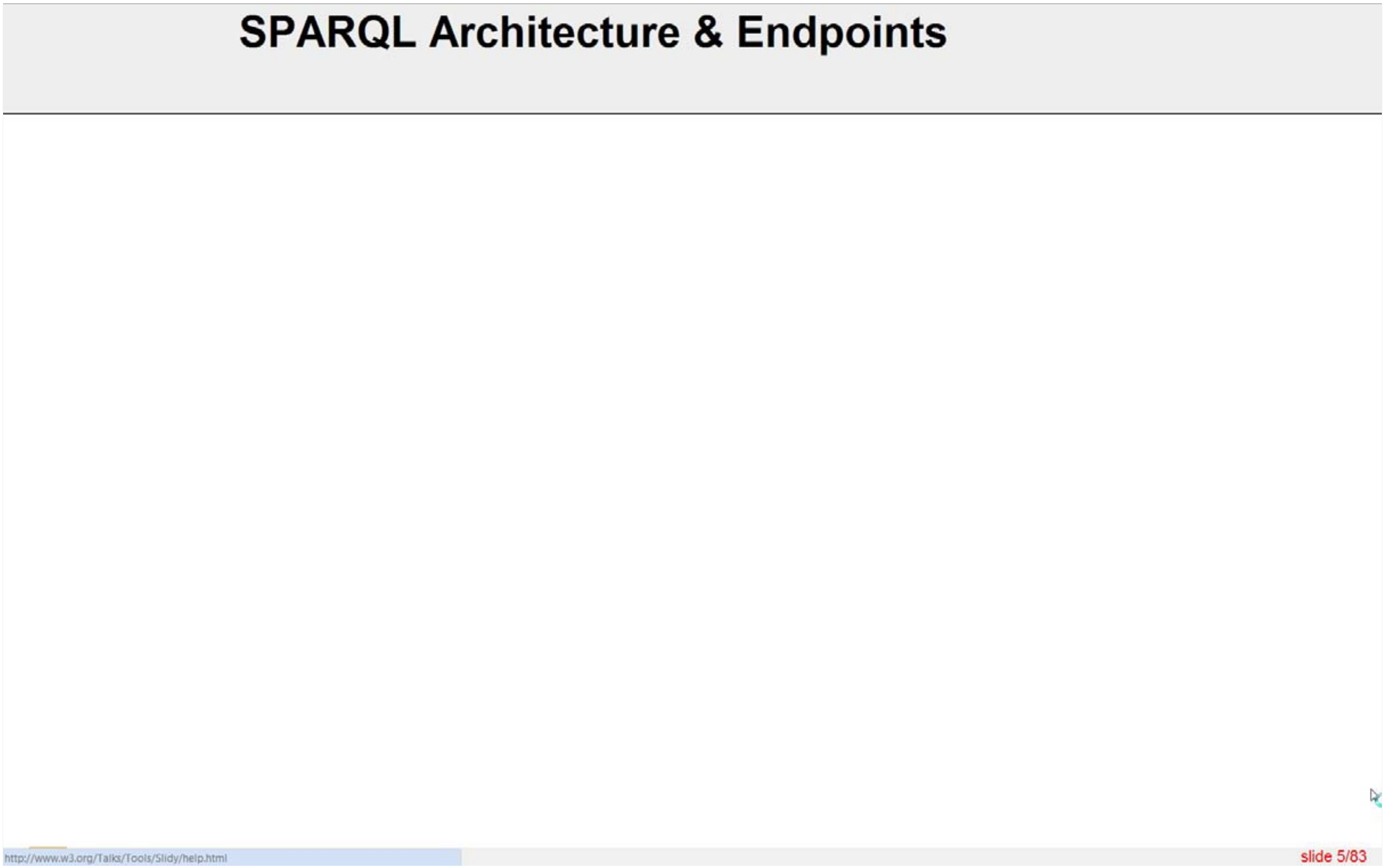


Structure of a SPARQL Query

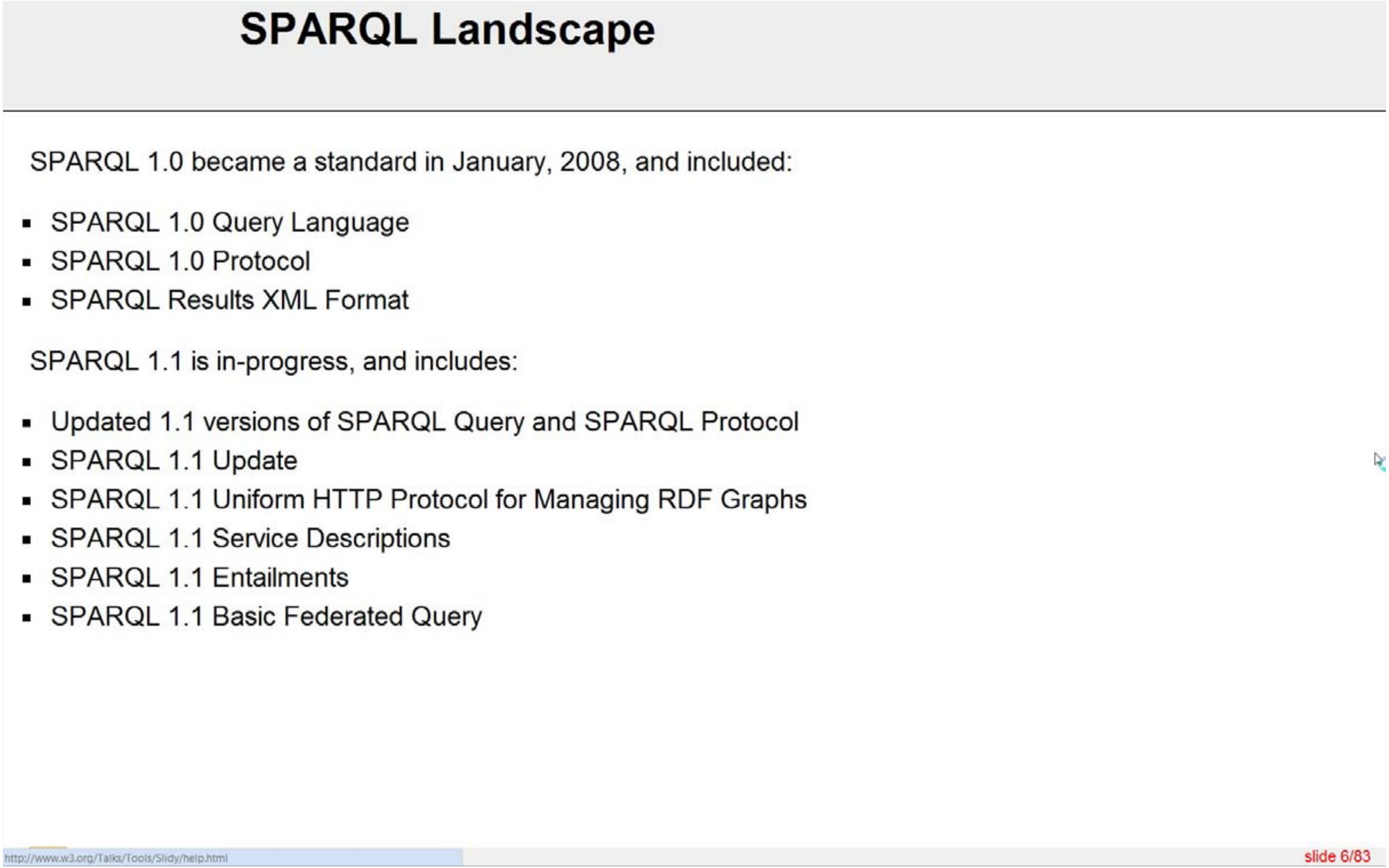
A SPARQL query comprises, in order:

- *Prefix declarations*, for abbreviating URIs
- *Dataset definition*, stating what RDF graph(s) are being queried
- A *result clause*, identifying what information to return from the query
- The *query pattern*, specifying what to query for in the underlying dataset
- *Query modifiers*, slicing, ordering, and otherwise rearranging query results

```
# prefix declarations
PREFIX foo: <http://example.com/resources/>
...
# dataset definition
FROM ...
# result clause
SELECT ...
# query pattern
WHERE {
    ...
}
# query modifiers
ORDER BY ...
```



SPARQL Architecture & Endpoints



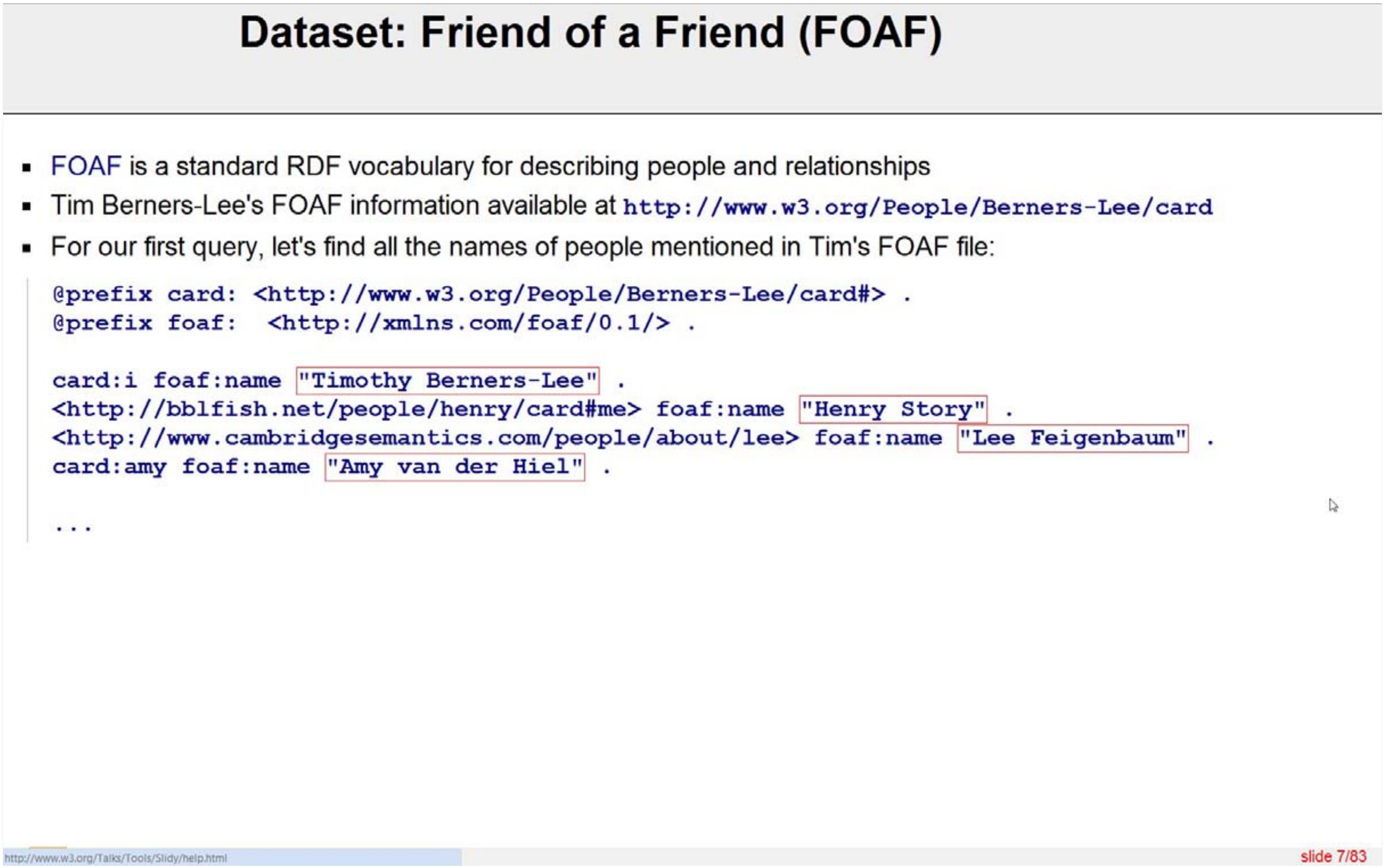
SPARQL Landscape

SPARQL 1.0 became a standard in January, 2008, and included:

- SPARQL 1.0 Query Language
- SPARQL 1.0 Protocol
- SPARQL Results XML Format

SPARQL 1.1 is in-progress, and includes:

- Updated 1.1 versions of SPARQL Query and SPARQL Protocol
- SPARQL 1.1 Update
- SPARQL 1.1 Uniform HTTP Protocol for Managing RDF Graphs
- SPARQL 1.1 Service Descriptions
- SPARQL 1.1 Entailments
- SPARQL 1.1 Basic Federated Query



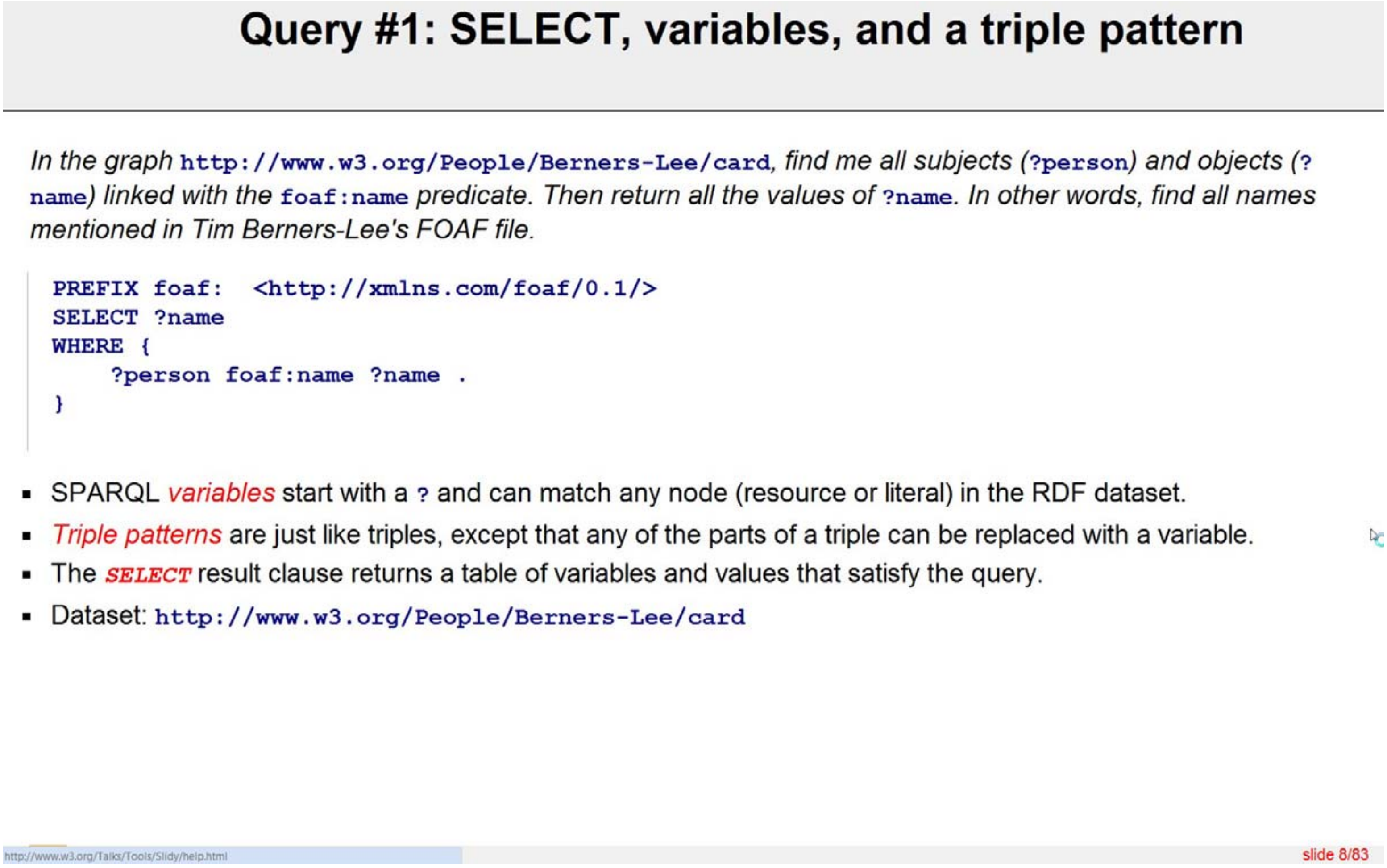
Dataset: Friend of a Friend (FOAF)

- FOAF is a standard RDF vocabulary for describing people and relationships
- Tim Berners-Lee's FOAF information available at `http://www.w3.org/People/Berners-Lee/card`
- For our first query, let's find all the names of people mentioned in Tim's FOAF file:

```
@prefix card: <http://www.w3.org/People/Berners-Lee/card#> .
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .

card:i foaf:name "Timothy Berners-Lee" .
<http://bblfish.net/people/henry/card#me> foaf:name "Henry Story" .
<http://www.cambridgesemantics.com/people/about/lee> foaf:name "Lee Feigenbaum" .
card:amy foaf:name "Amy van der Hiel" .

...
```

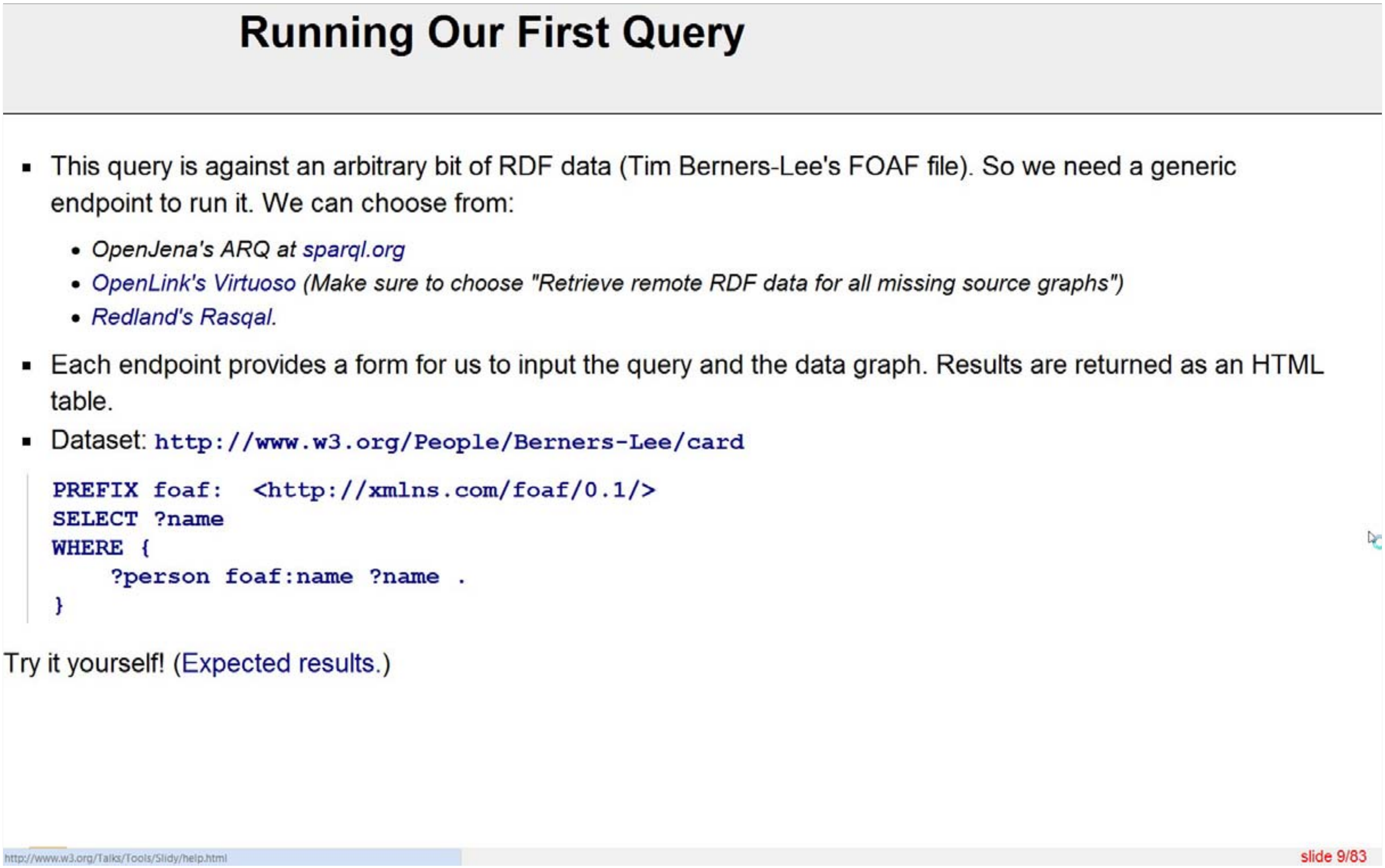


Query #1: SELECT, variables, and a triple pattern

In the graph `http://www.w3.org/People/Berners-Lee/card`*, find me all subjects (*`?person`*) and objects (*`?name`*) linked with the* `foaf:name` *predicate. Then return all the values of* `?name`*. In other words, find all names mentioned in Tim Berners-Lee's FOAF file.*

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE {
    ?person foaf:name ?name .
}
```

- SPARQL *variables* start with a `?` and can match any node (resource or literal) in the RDF dataset.
- *Triple patterns* are just like triples, except that any of the parts of a triple can be replaced with a variable.
- The ***`SELECT`*** result clause returns a table of variables and values that satisfy the query.
- Dataset: `http://www.w3.org/People/Berners-Lee/card`

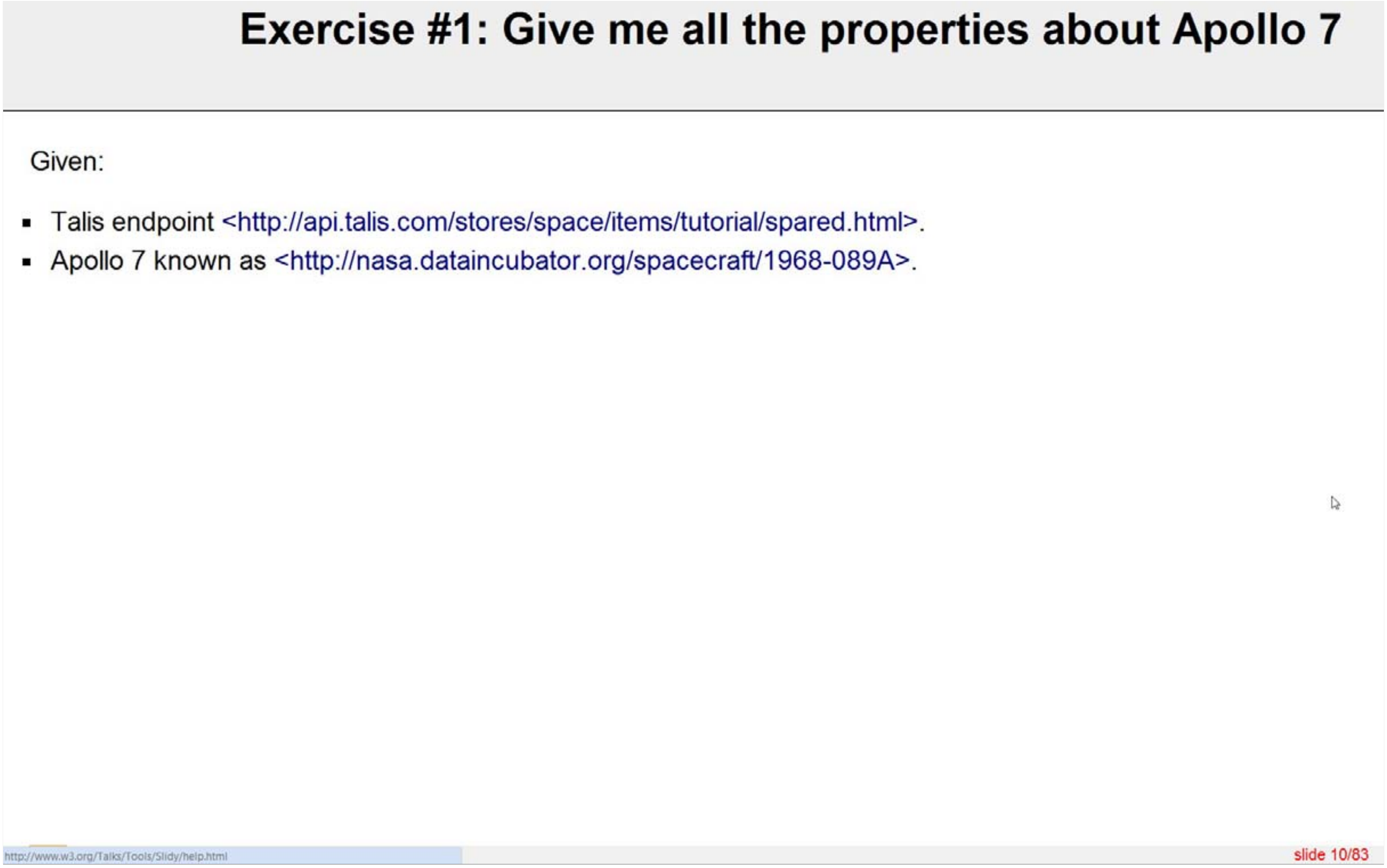


Running Our First Query

- This query is against an arbitrary bit of RDF data (Tim Berners-Lee's FOAF file). So we need a generic endpoint to run it. We can choose from:
  - *OpenJena's ARQ at sparql.org*
  - *OpenLink's Virtuoso (Make sure to choose "Retrieve remote RDF data for all missing source graphs")*
  - *Redland's Rasqal.*
- Each endpoint provides a form for us to input the query and the data graph. Results are returned as an HTML table.
- Dataset: **`http://www.w3.org/People/Berners-Lee/card`**

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE {
    ?person foaf:name ?name .
}
```

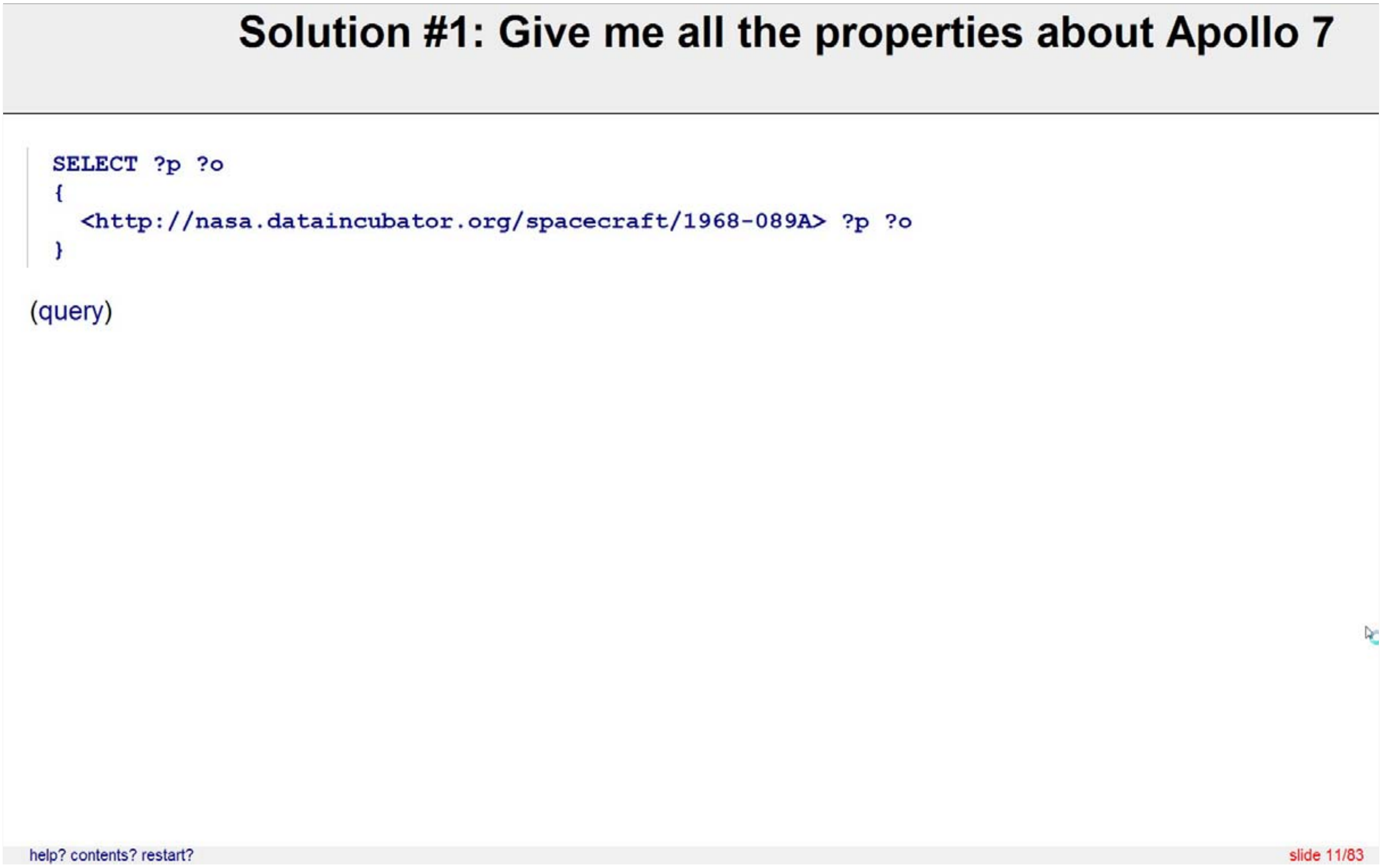
Try it yourself! (Expected results.)



Exercise #1: Give me all the properties about Apollo 7

Given:

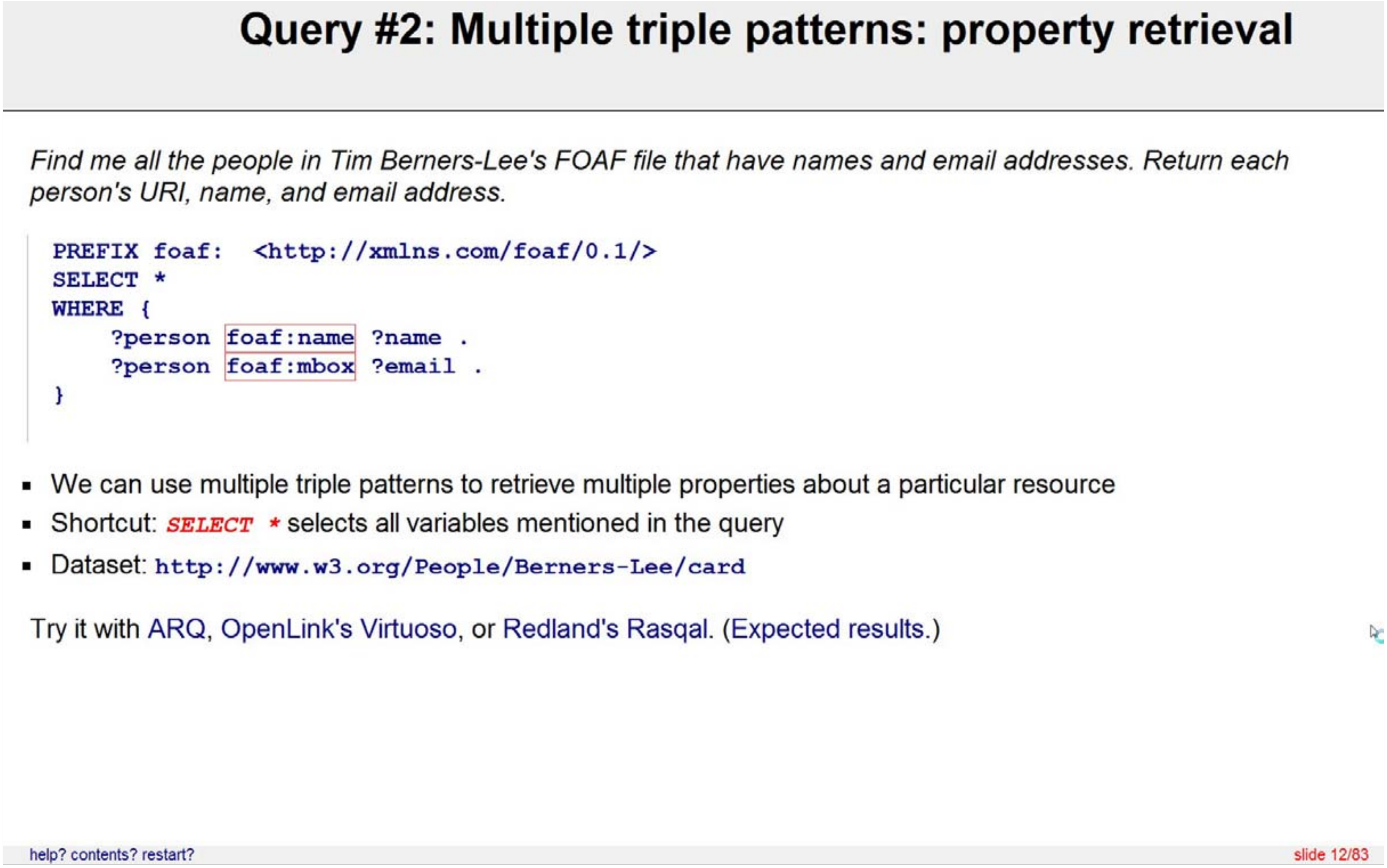
- Talis endpoint <http://api.talis.com/stores/space/items/tutorial/spared.html>.
- Apollo 7 known as <http://nasa.dataincubator.org/spacecraft/1968-089A>.



Solution #1: Give me all the properties about Apollo 7

```
SELECT ?p ?o
{
  <http://nasa.dataincubator.org/spacecraft/1968-089A> ?p ?o
}
```

(query)



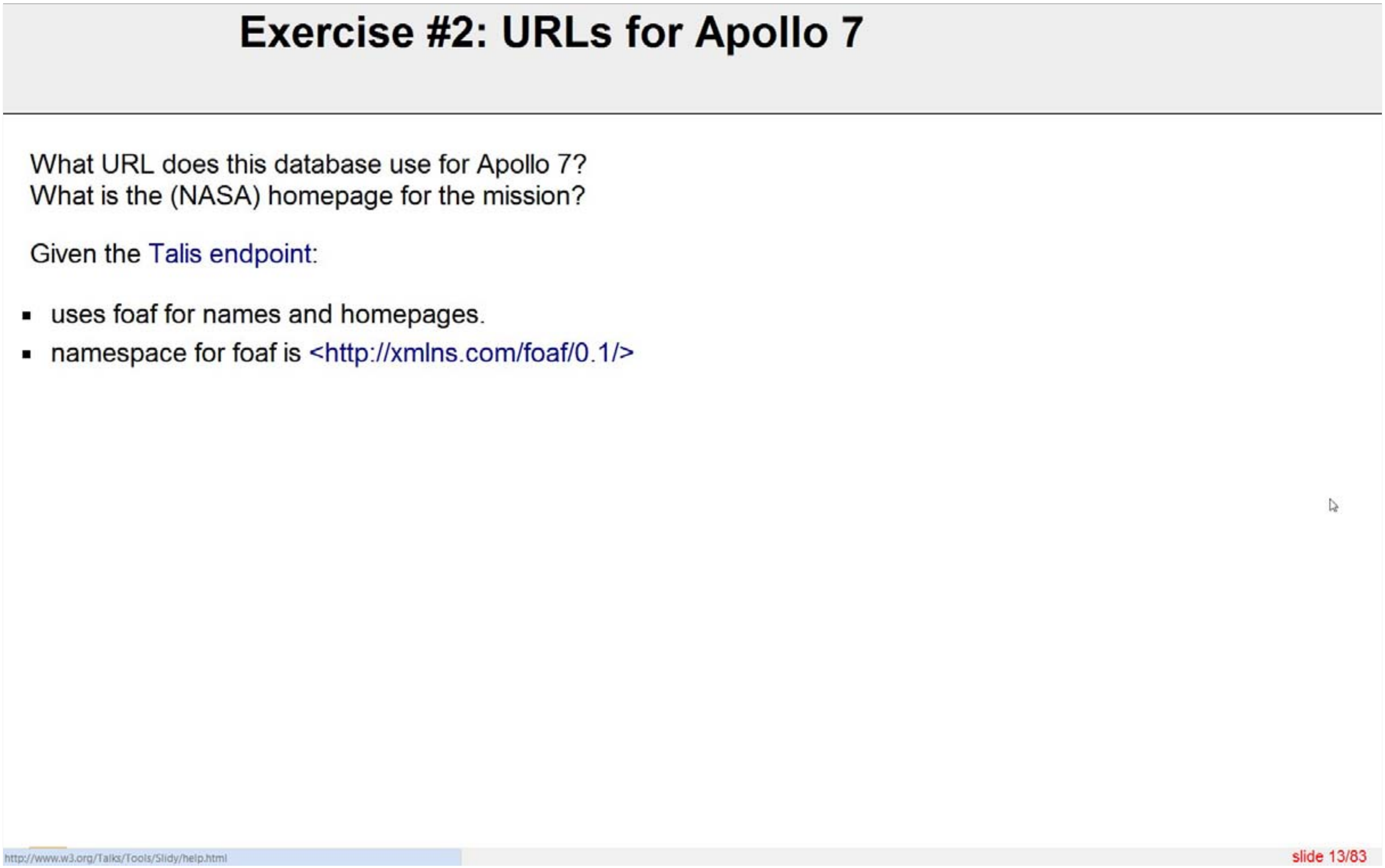
Query #2: Multiple triple patterns: property retrieval

Find me all the people in Tim Berners-Lee's FOAF file that have names and email addresses. Return each person's URI, name, and email address.

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT *
WHERE {
    ?person foaf:name ?name .
    ?person foaf:mbox ?email .
}
```

- We can use multiple triple patterns to retrieve multiple properties about a particular resource
- Shortcut: ***SELECT **** selects all variables mentioned in the query
- Dataset: **http://www.w3.org/People/Berners-Lee/card**

Try it with ARQ, OpenLink's Virtuoso, or Redland's Rasqal. (Expected results.)



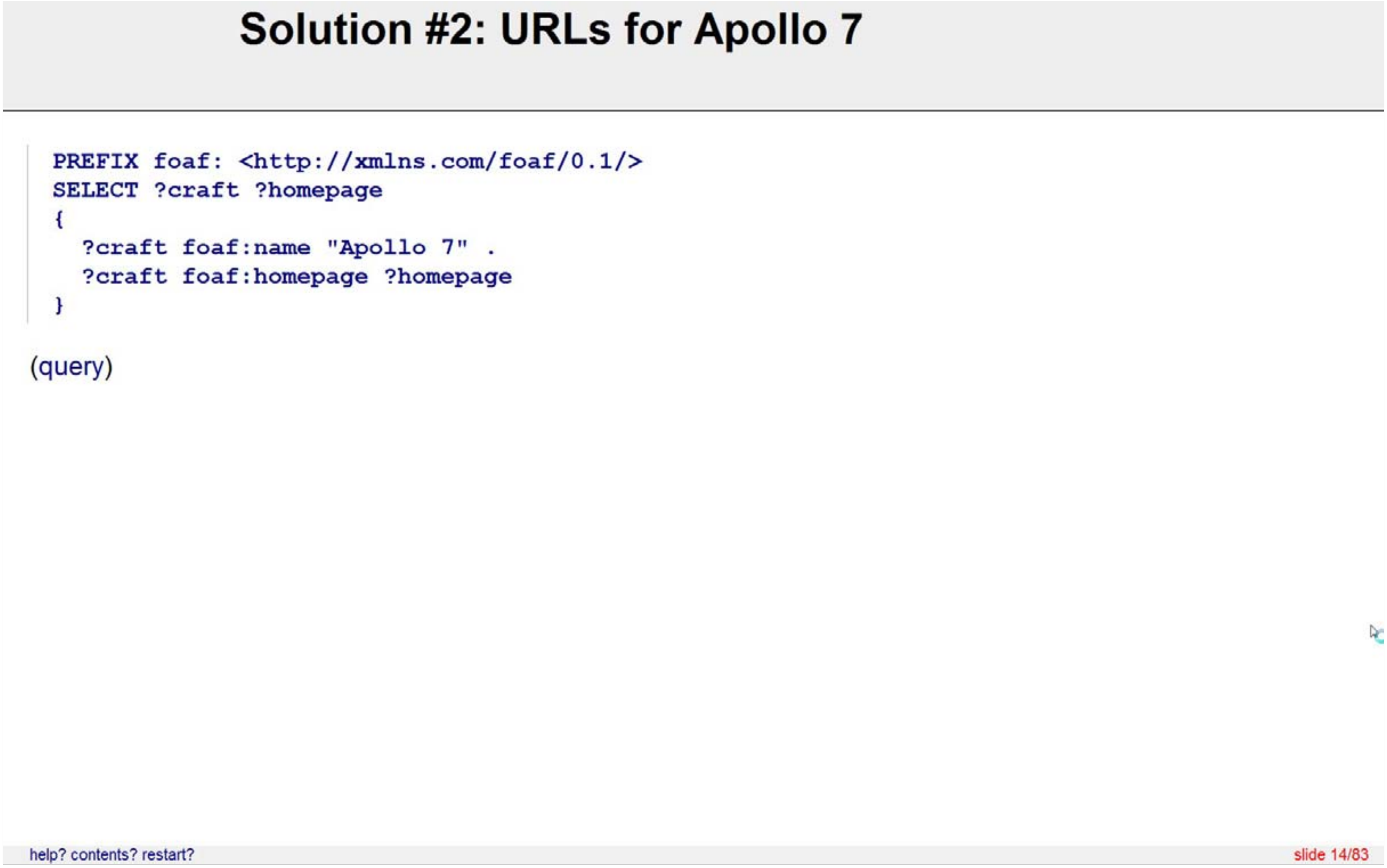
Exercise #2: URLs for Apollo 7

What URL does this database use for Apollo 7?
What is the (NASA) homepage for the mission?

Given the Talis endpoint:

- uses foaf for names and homepages.
- namespace for foaf is <http://xmlns.com/foaf/0.1/>

http://www.w3.org/Talks/Tools/Slidy/help.html

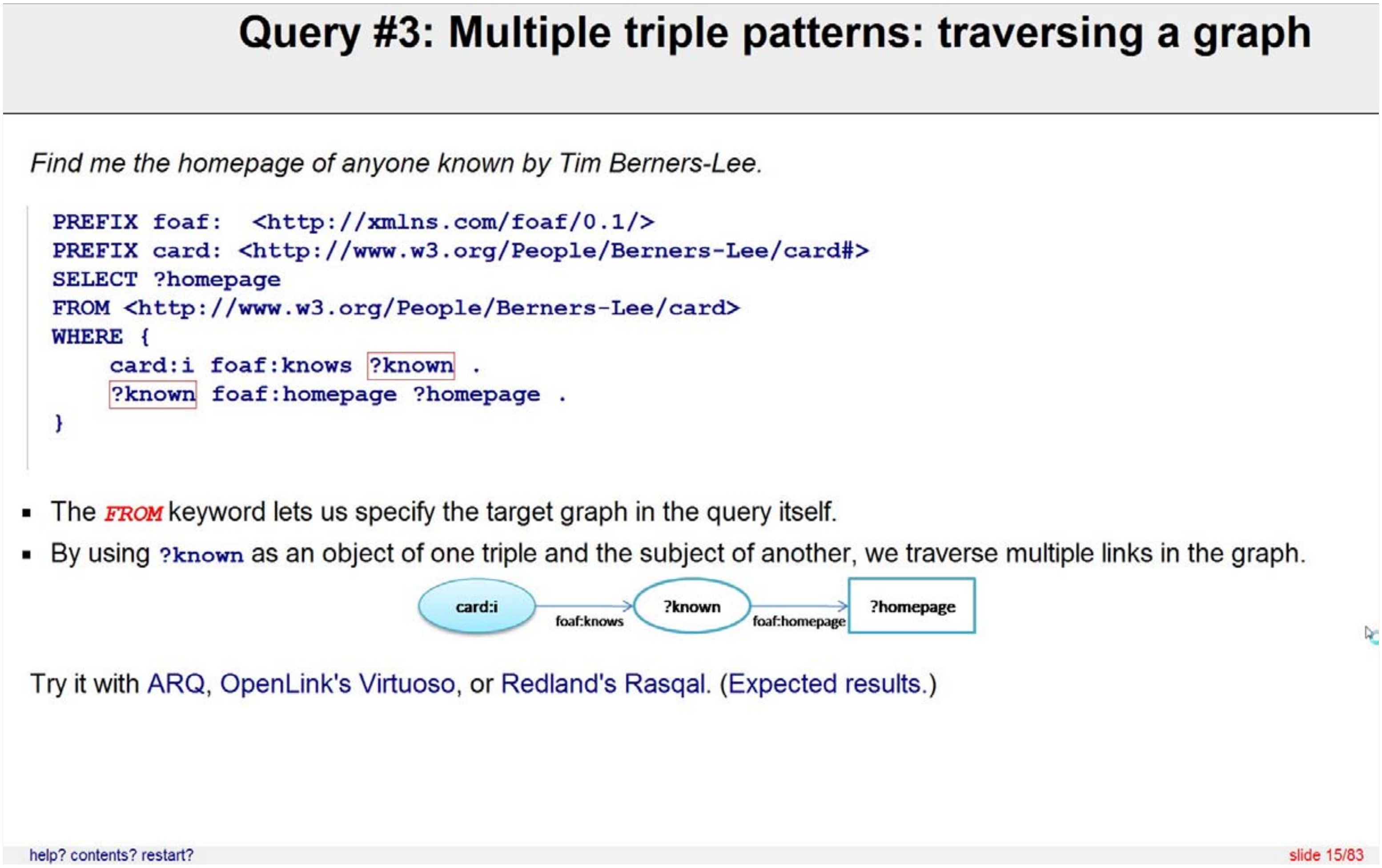


Solution #2: URLs for Apollo 7

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?craft ?homepage
{
  ?craft foaf:name "Apollo 7" .
  ?craft foaf:homepage ?homepage
}
```

(query)

help? contents? restart?



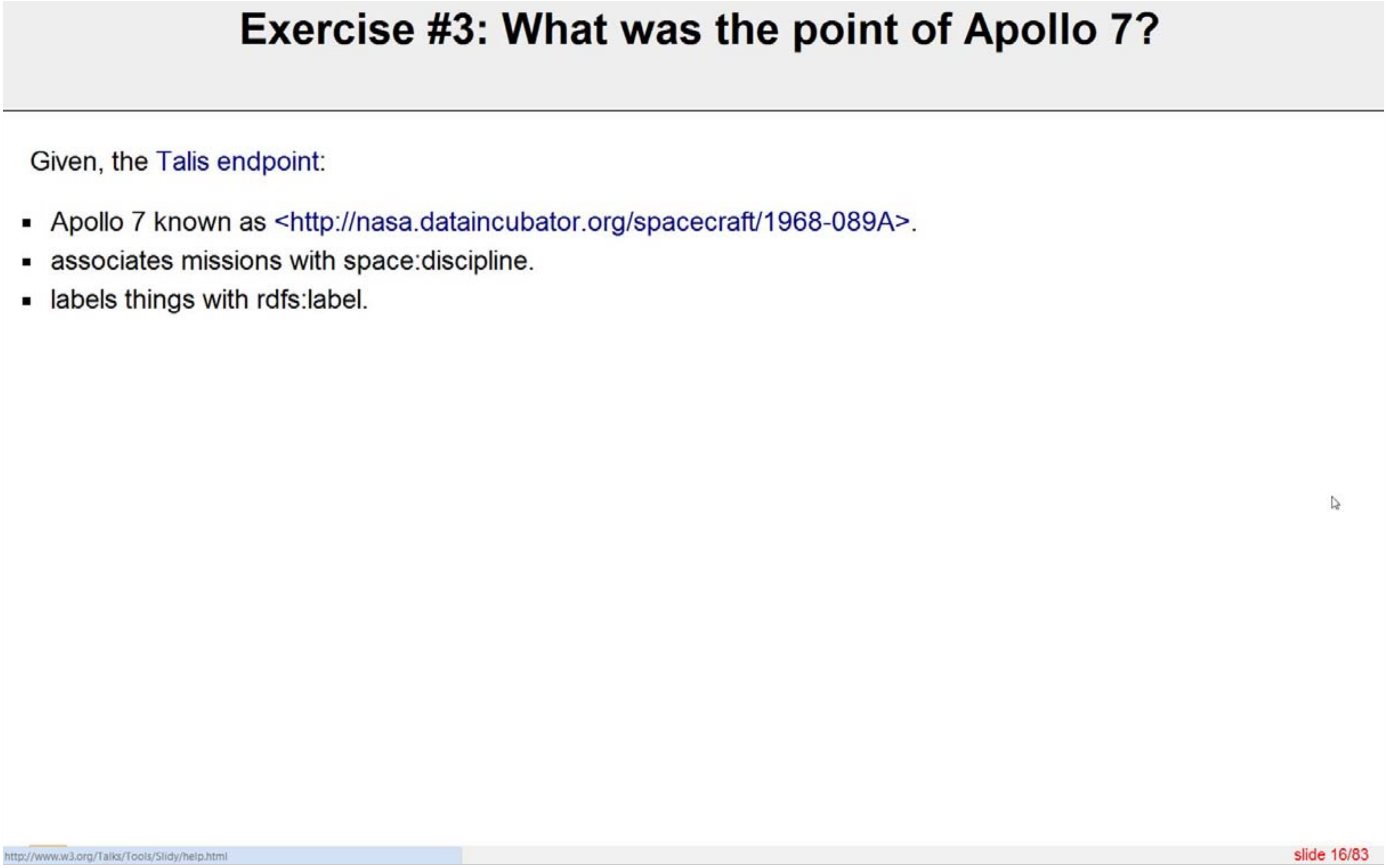
Query #3: Multiple triple patterns: traversing a graph

Find me the homepage of anyone known by Tim Berners-Lee.

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

- The ***FROM*** keyword lets us specify the target graph in the query itself.
- By using `?known` as an object of one triple and the subject of another, we traverse multiple links in the graph.

Try it with ARQ, OpenLink's Virtuoso, or Redland's Rasqal. (Expected results.)



Exercise #3: What was the point of Apollo 7?

Given, the Talis endpoint:

- Apollo 7 known as <http://nasa.dataincubator.org/spacecraft/1968-089A>.
- associates missions with space:discipline.
- labels things with rdfs:label.

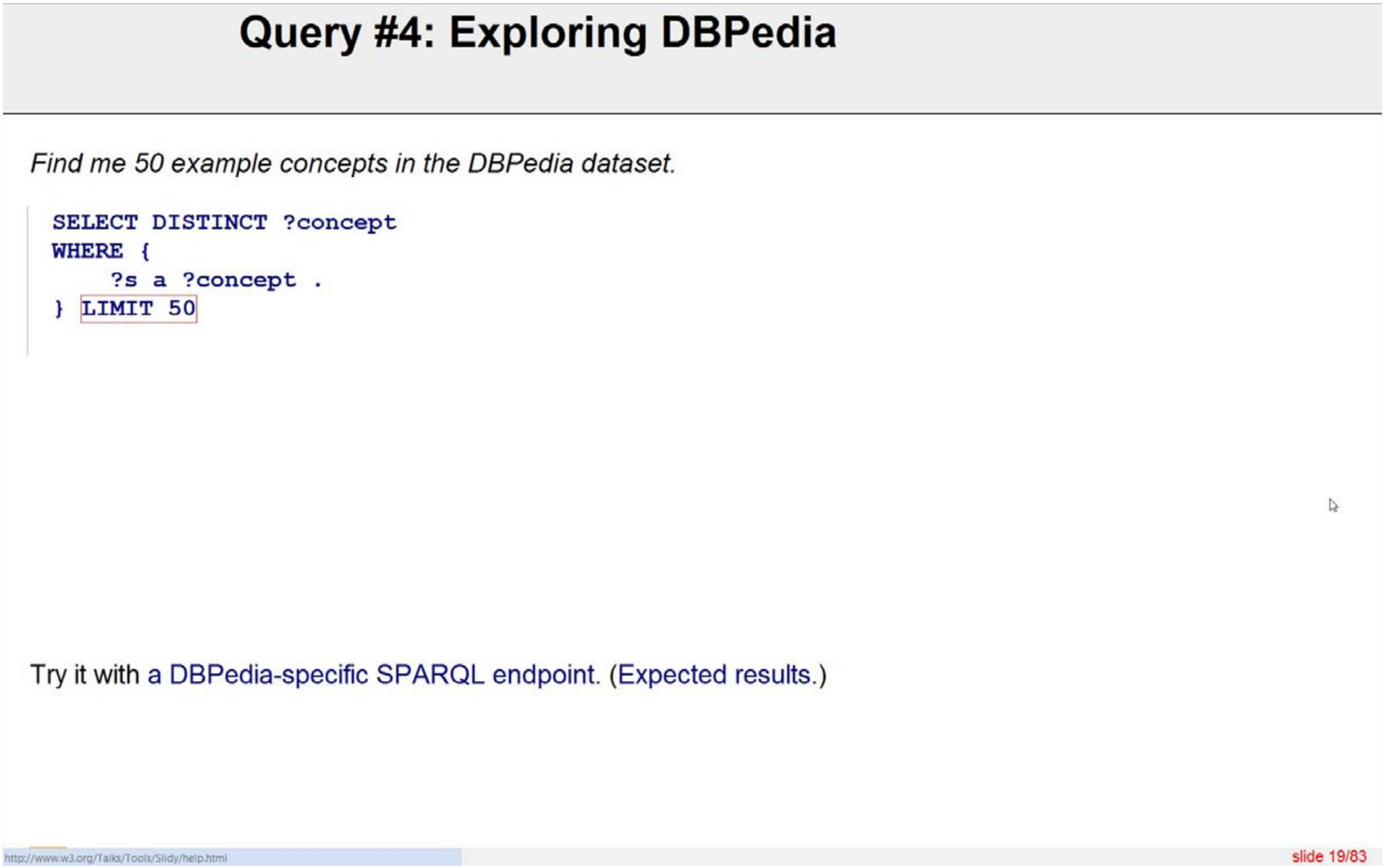
Solution #3: What was the point of Apollo 7?

```
PREFIX space: <http://purl.org/net/schemas/space/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?disc ?label
{
  <http://nasa.dataincubator.org/spacecraft/1968-089A> space:discipline ?disc .
  ?disc rdfs:label ?label
}
```

(query)

Dataset: DBPedia

- DBPedia is an RDF version of information from Wikipedia.
- DBPedia contains data derived from Wikipedia's infoboxes, category hierarchy, article abstracts, and various external links.
- DBpedia contains over 100 million triples.

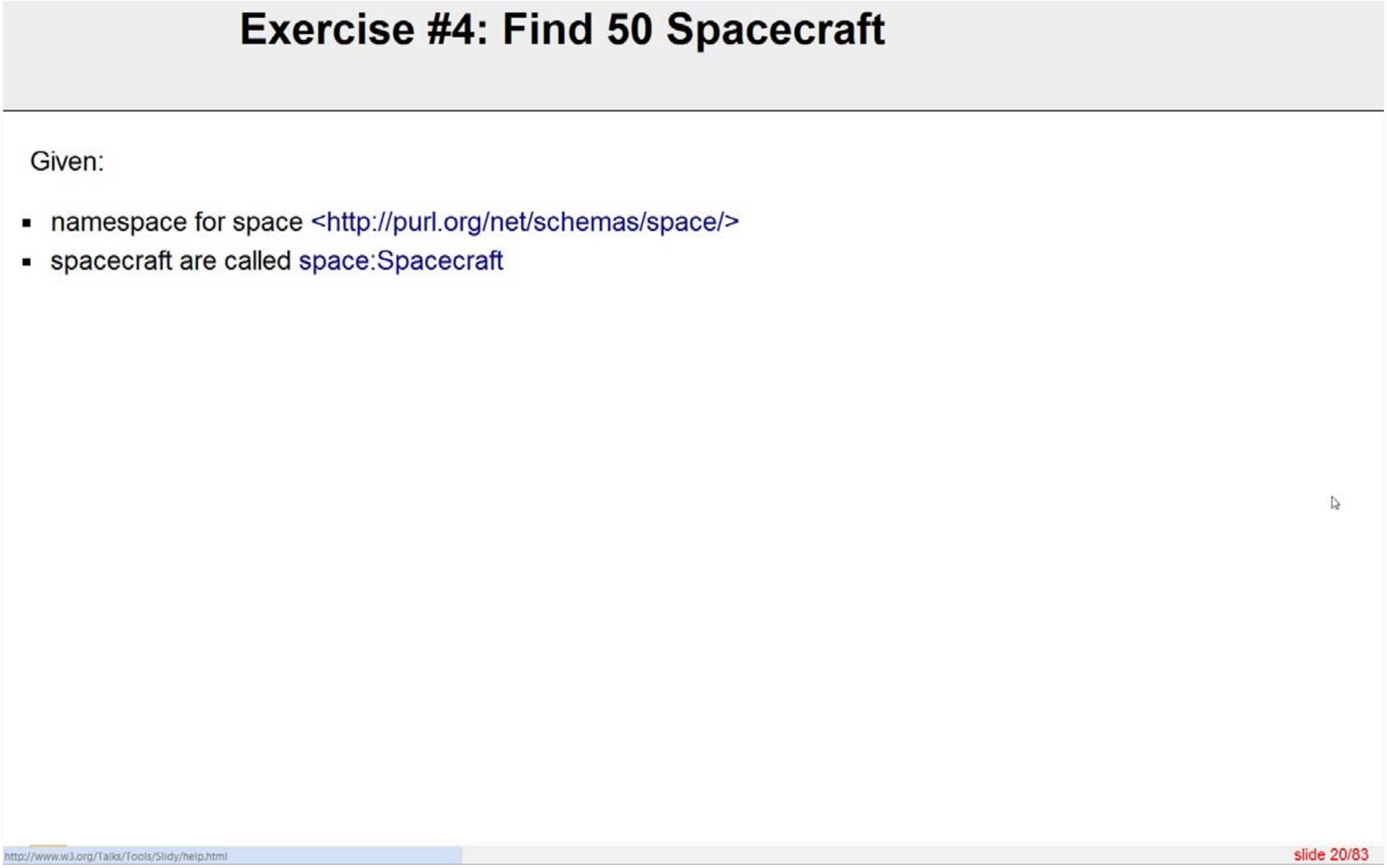


Query #4: Exploring DBPedia

Find me 50 example concepts in the DBPedia dataset.

```
SELECT DISTINCT ?concept
WHERE {
    ?s a ?concept .
} LIMIT 50
```

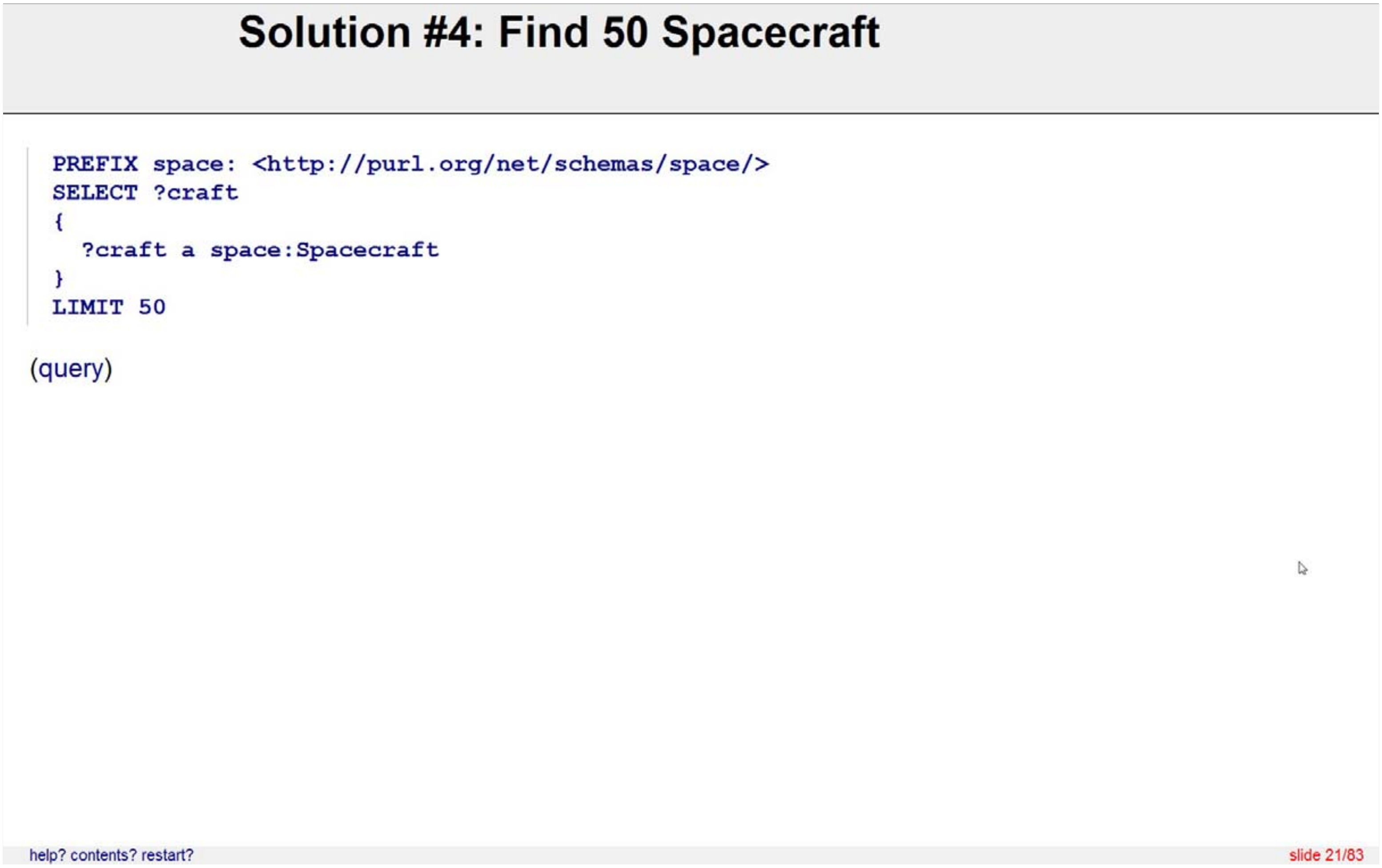
Try it with a DBPedia-specific SPARQL endpoint. (Expected results.)



Exercise #4: Find 50 Spacecraft

Given:

- namespace for space <http://purl.org/net/schemas/space/>
- spacecraft are called space:Spacecraft



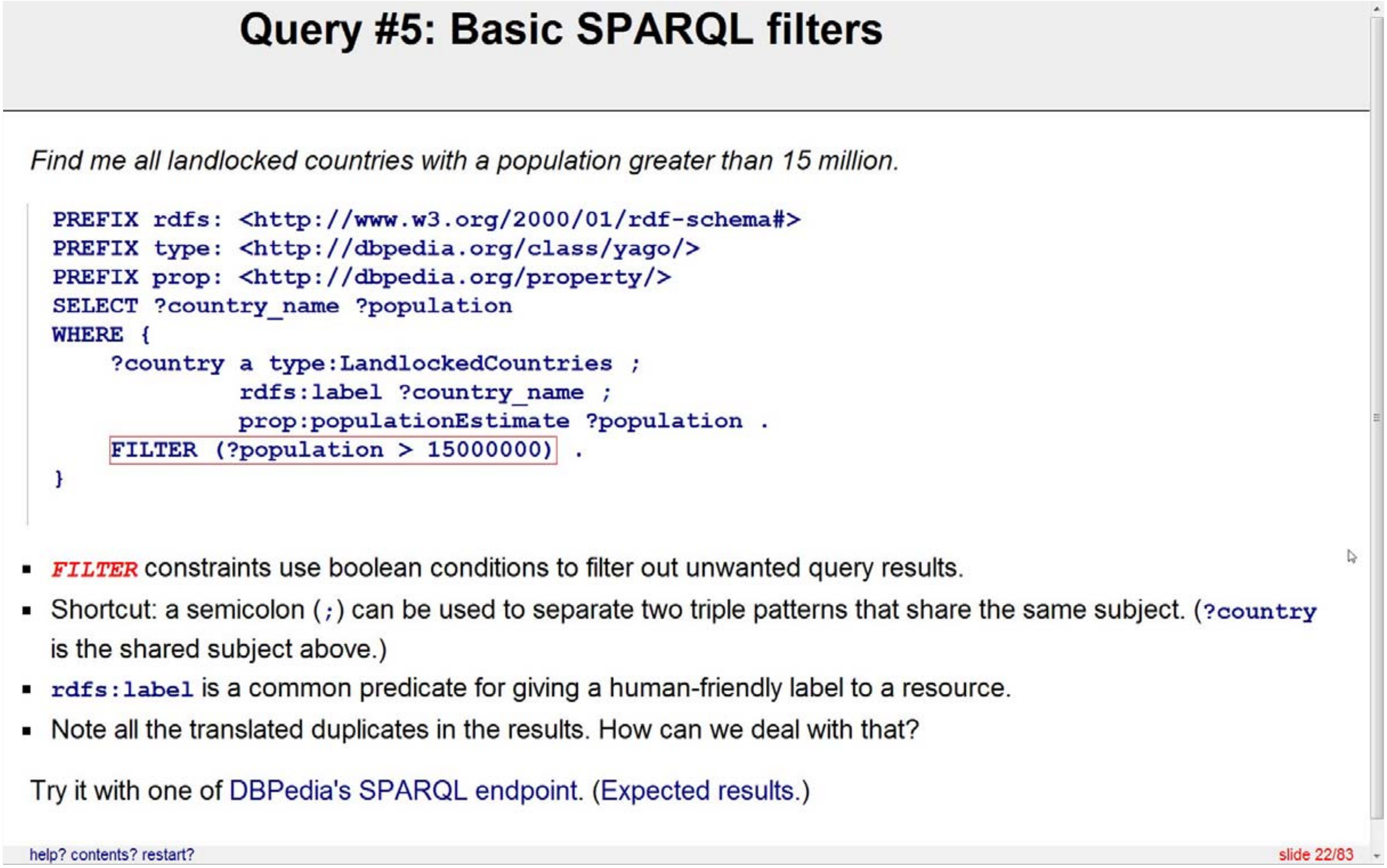
Solution #4: Find 50 Spacecraft

```
PREFIX space: <http://purl.org/net/schemas/space/>
SELECT ?craft
{
  ?craft a space:Spacecraft
}
LIMIT 50
```

(query)

help? contents? restart?

slide 21/83



Query #5: Basic SPARQL filters

Find me all landlocked countries with a population greater than 15 million.

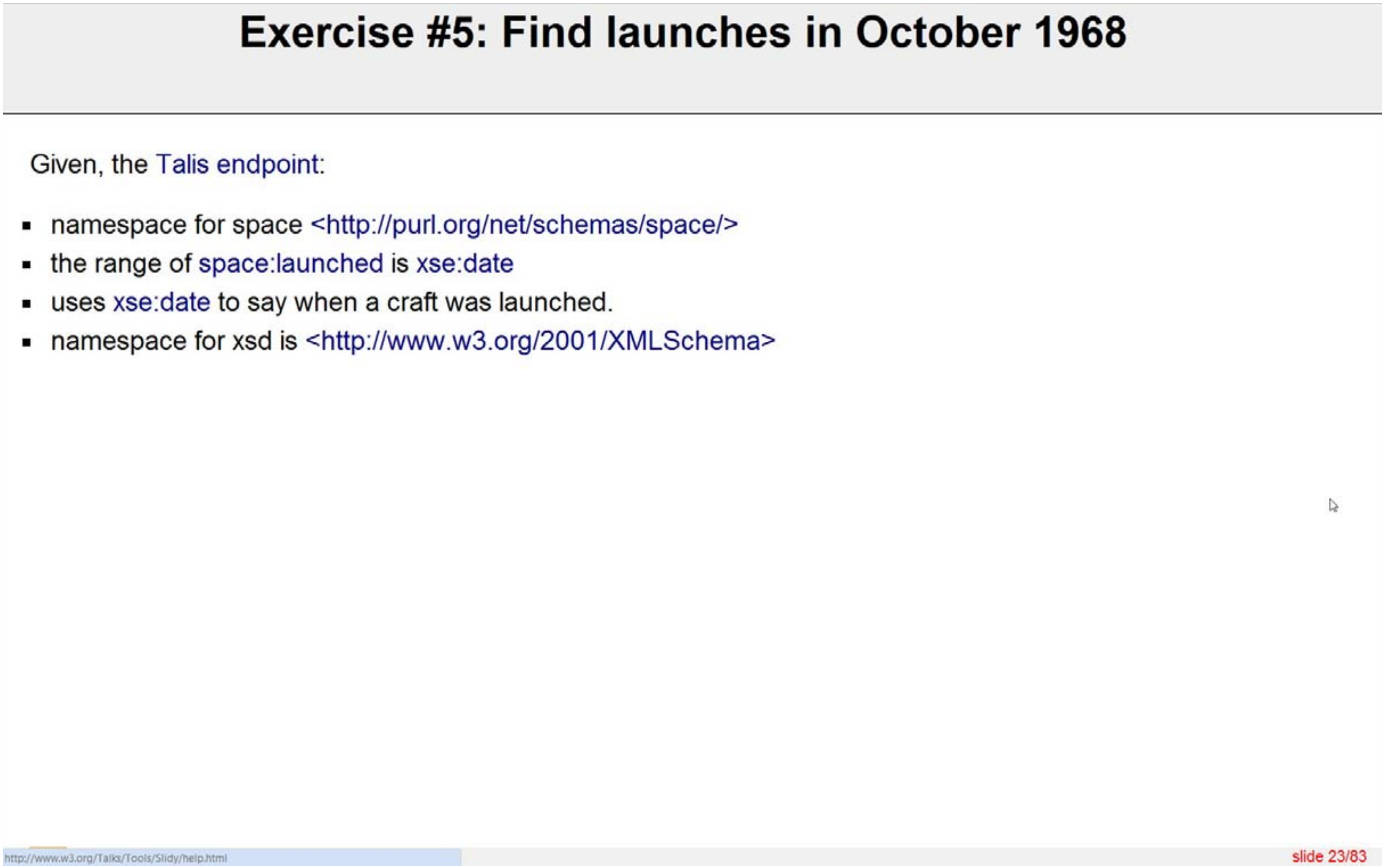
```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX type: <http://dbpedia.org/class/yago/>
PREFIX prop: <http://dbpedia.org/property/>
SELECT ?country_name ?population
WHERE {
    ?country a type:LandlockedCountries ;
             rdfs:label ?country_name ;
             prop:populationEstimate ?population .
    FILTER (?population > 15000000) .
}
```

- ***`FILTER`*** constraints use boolean conditions to filter out unwanted query results.
- Shortcut: a semicolon (`;`) can be used to separate two triple patterns that share the same subject. (`?country` is the shared subject above.)
- `rdfs:label` is a common predicate for giving a human-friendly label to a resource.
- Note all the translated duplicates in the results. How can we deal with that?

Try it with one of DBPedia's SPARQL endpoint. (Expected results.)

help? contents? restart?

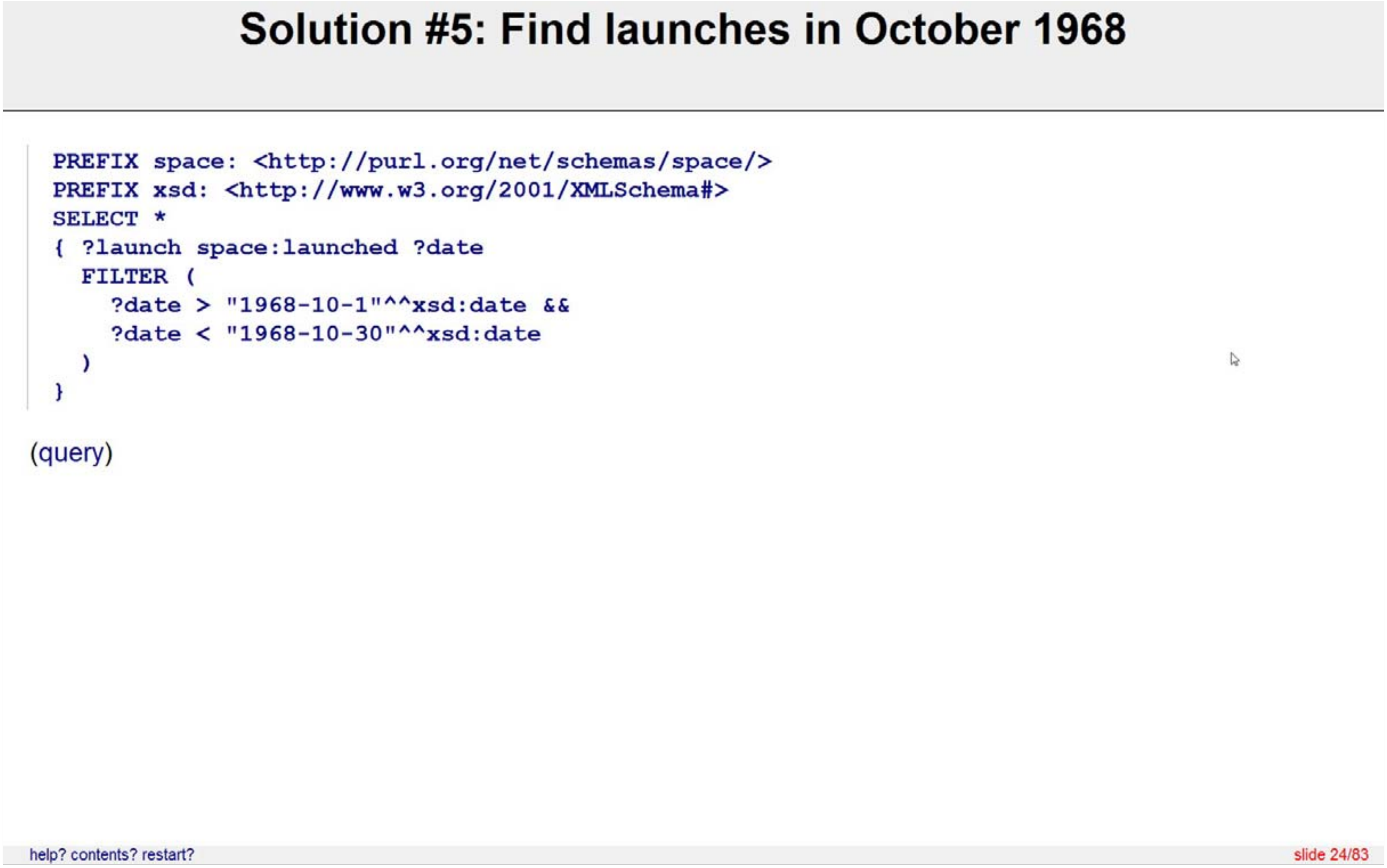
slide 22/83



Exercise #5: Find launches in October 1968

Given, the Talis endpoint:

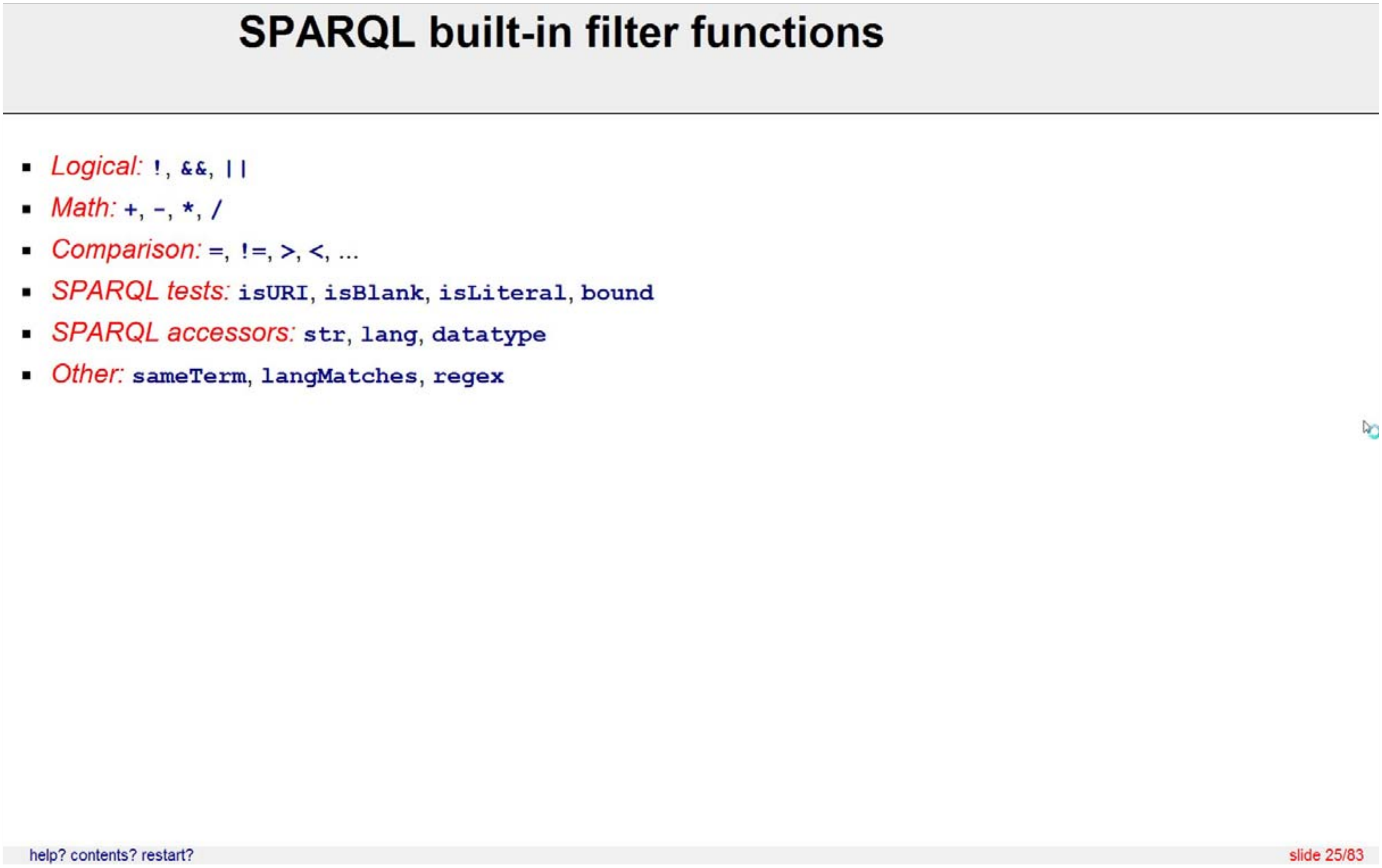
- namespace for space <http://purl.org/net/schemas/space/>
- the range of space:launched is xse:date
- uses xse:date to say when a craft was launched.
- namespace for xsd is <http://www.w3.org/2001/XMLSchema>



Solution #5: Find launches in October 1968

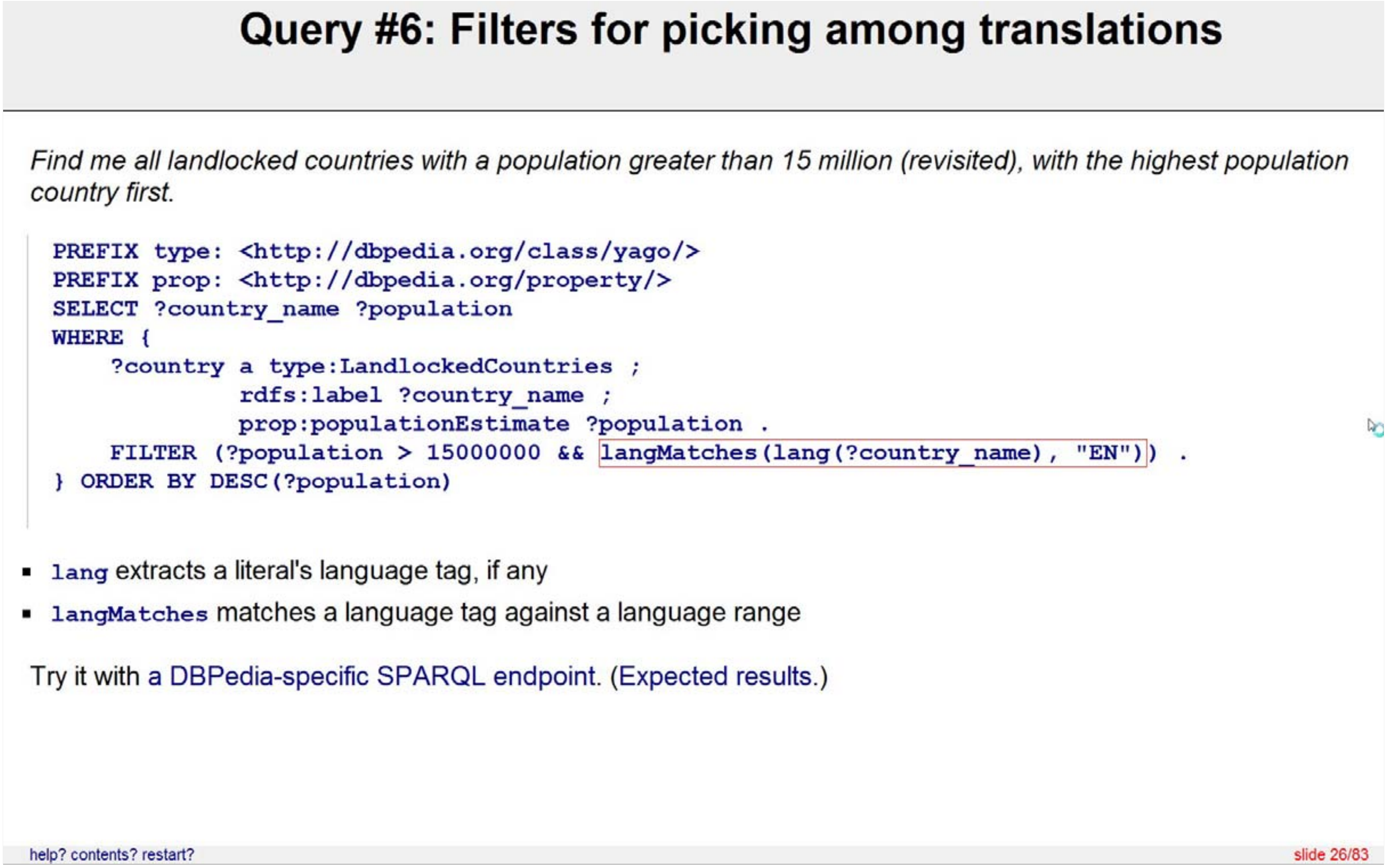
```
PREFIX space: <http://purl.org/net/schemas/space/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT *
{ ?launch space:launched ?date
  FILTER (
    ?date > "1968-10-1"^^xsd:date &&
    ?date < "1968-10-30"^^xsd:date
  )
}
```

(query)



SPARQL built-in filter functions

- *Logical:* **!**, **&&**, **||**
- *Math:* **+**, **-**, *****, **/**
- *Comparison:* =, **!=**, **>**, **<**, ...
- *SPARQL tests:* **isURI**, **isBlank**, **isLiteral**, **bound**
- *SPARQL accessors:* **str**, **lang**, **datatype**
- *Other:* **sameTerm**, **langMatches**, **regex**



Query #6: Filters for picking among translations

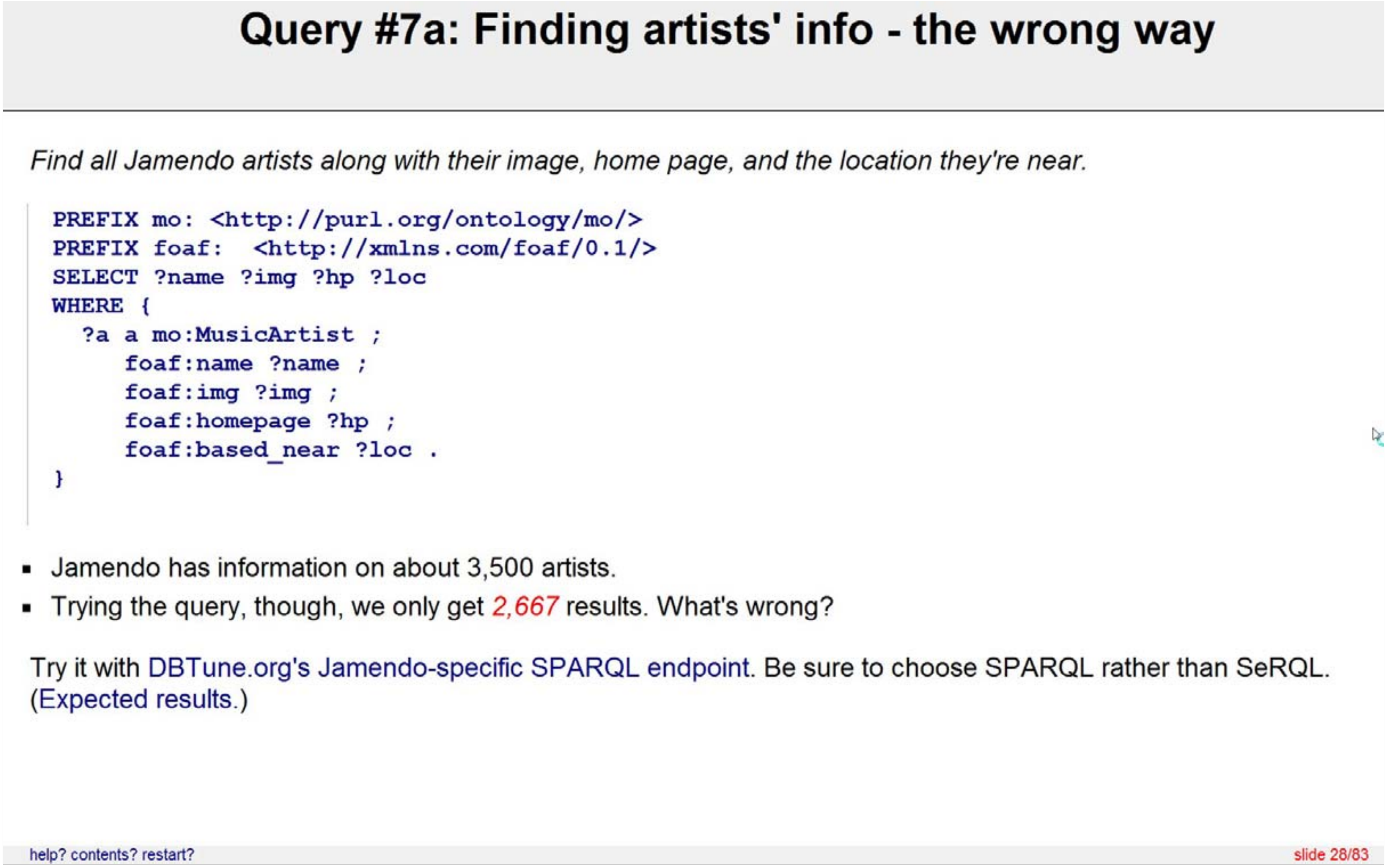
Find me all landlocked countries with a population greater than 15 million (revisited), with the highest population country first.

```
PREFIX type: <http://dbpedia.org/class/yago/>
PREFIX prop: <http://dbpedia.org/property/>
SELECT ?country_name ?population
WHERE {
    ?country a type:LandlockedCountries ;
             rdfs:label ?country_name ;
             prop:populationEstimate ?population .
    FILTER (?population > 15000000 && langMatches(lang(?country_name), "EN")) .
} ORDER BY DESC(?population)
```

- **lang** extracts a literal's language tag, if any
- **langMatches** matches a language tag against a language range

Try it with a DBPedia-specific SPARQL endpoint. (Expected results.)

Dataset: Jamendo



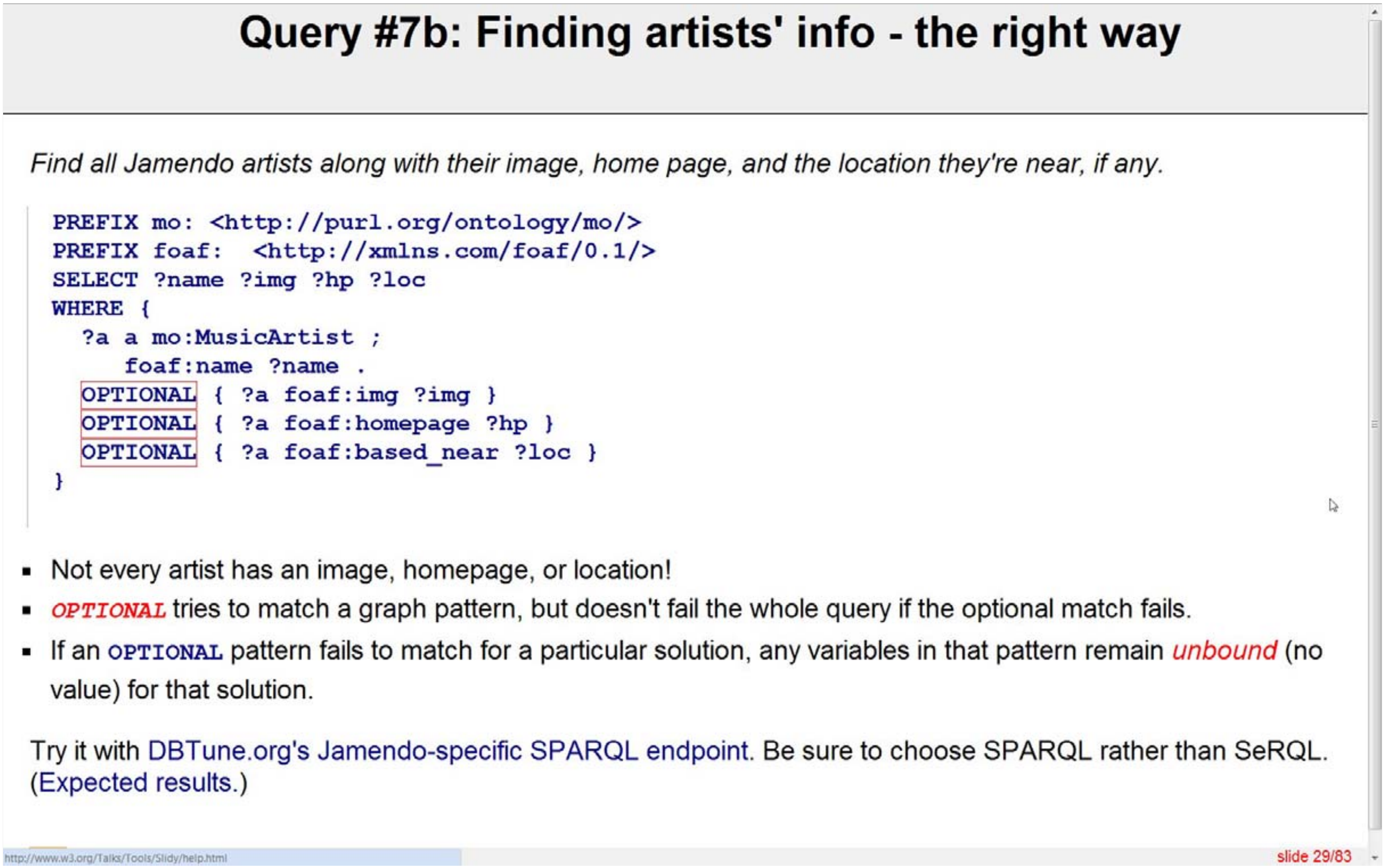
Query #7a: Finding artists' info - the wrong way

Find all Jamendo artists along with their image, home page, and the location they're near.

```
PREFIX mo: <http://purl.org/ontology/mo/>
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name ?img ?hp ?loc
WHERE {
  ?a a mo:MusicArtist ;
     foaf:name ?name ;
     foaf:img ?img ;
     foaf:homepage ?hp ;
     foaf:based_near ?loc .
}
```

- Jamendo has information on about 3,500 artists.
- Trying the query, though, we only get *2,667* results. What's wrong?

Try it with DBTune.org's Jamendo-specific SPARQL endpoint. Be sure to choose SPARQL rather than SeRQL. (Expected results.)



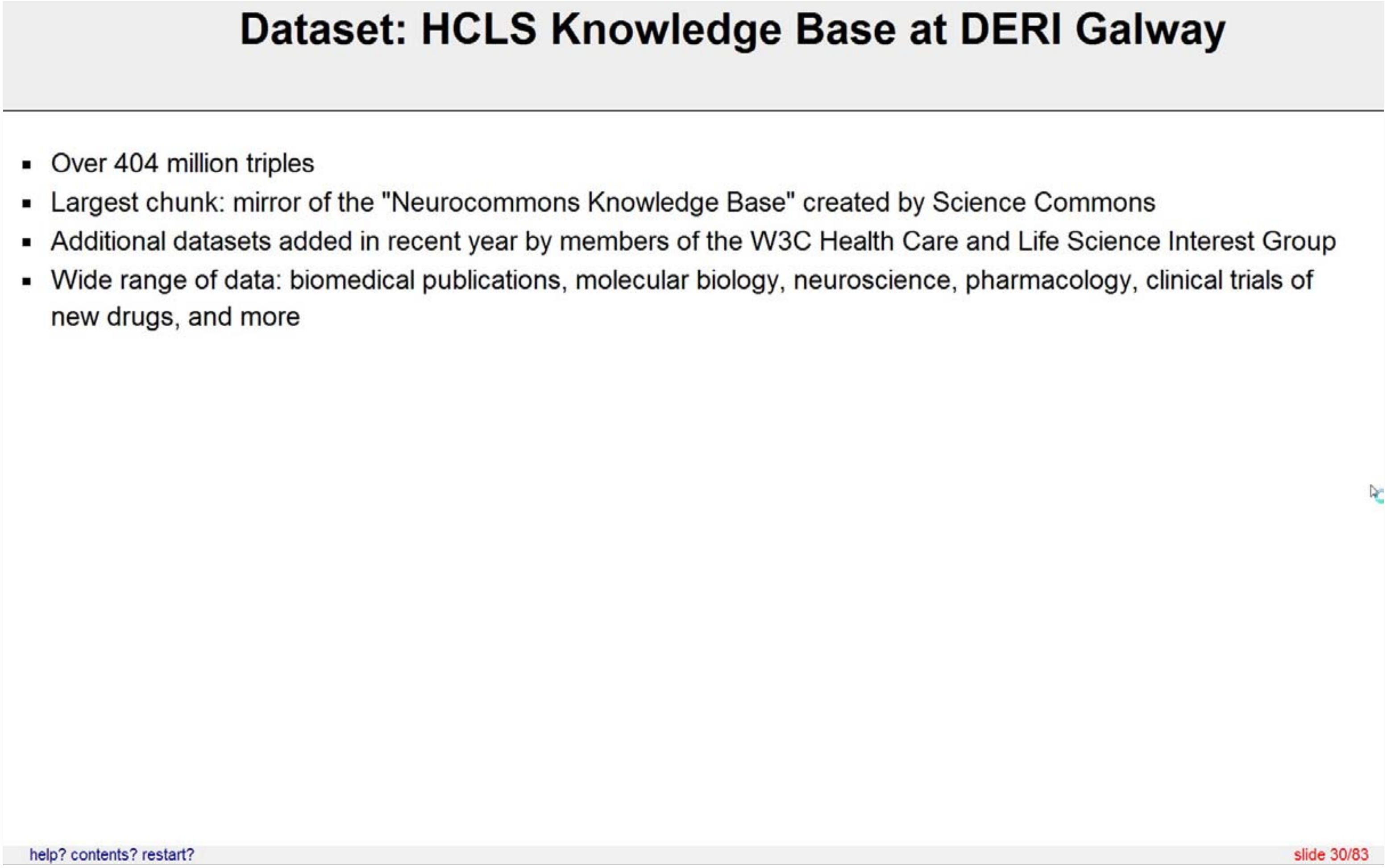
Query #7b: Finding artists' info - the right way

Find all Jamendo artists along with their image, home page, and the location they're near, if any.

```
PREFIX mo: <http://purl.org/ontology/mo/>
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name ?img ?hp ?loc
WHERE {
  ?a a mo:MusicArtist ;
     foaf:name ?name .
  OPTIONAL { ?a foaf:img ?img }
  OPTIONAL { ?a foaf:homepage ?hp }
  OPTIONAL { ?a foaf:based_near ?loc }
}
```

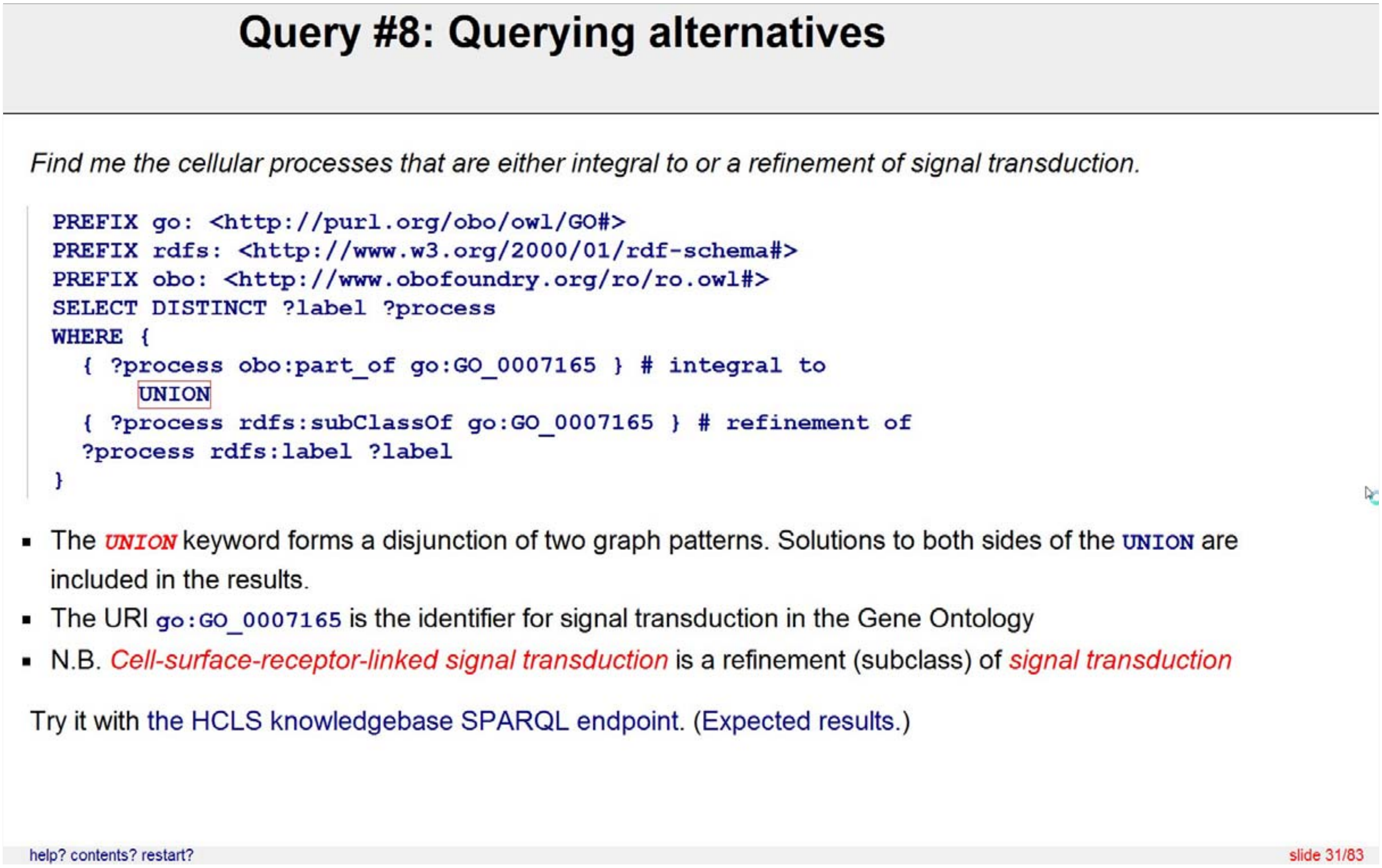
- Not every artist has an image, homepage, or location!
- *OPTIONAL* tries to match a graph pattern, but doesn't fail the whole query if the optional match fails.
- If an OPTIONAL pattern fails to match for a particular solution, any variables in that pattern remain *unbound* (no value) for that solution.

Try it with DBTune.org's Jamendo-specific SPARQL endpoint. Be sure to choose SPARQL rather than SeRQL. (Expected results.)



Dataset: HCLS Knowledge Base at DERI Galway

- Over 404 million triples
- Largest chunk: mirror of the "Neurocommons Knowledge Base" created by Science Commons
- Additional datasets added in recent year by members of the W3C Health Care and Life Science Interest Group
- Wide range of data: biomedical publications, molecular biology, neuroscience, pharmacology, clinical trials of new drugs, and more



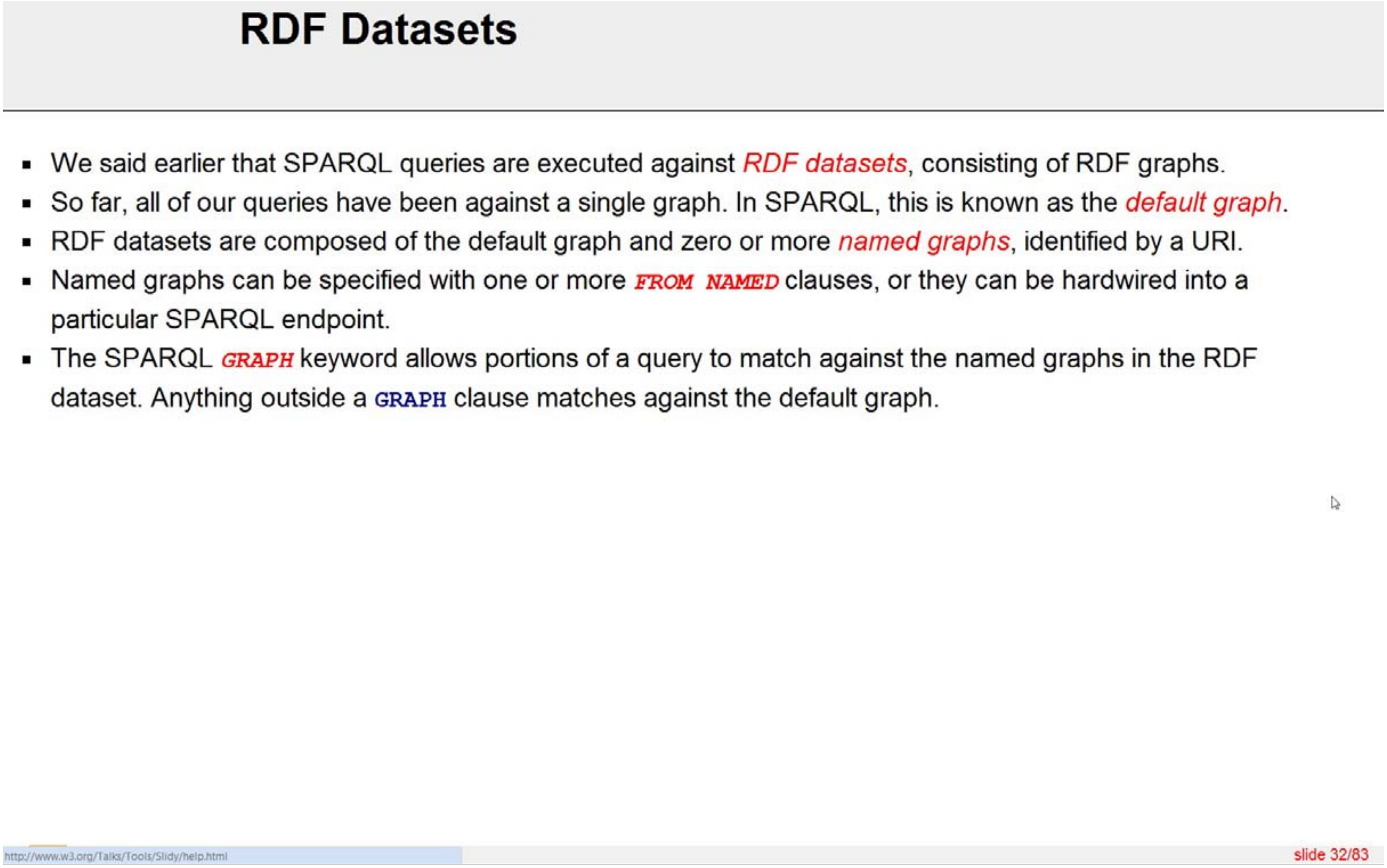
Query #8: Querying alternatives

Find me the cellular processes that are either integral to or a refinement of signal transduction.

```
PREFIX go: <http://purl.org/obo/owl/GO#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX obo: <http://www.obofoundry.org/ro/ro.owl#>
SELECT DISTINCT ?label ?process
WHERE {
  { ?process obo:part_of go:GO_0007165 } # integral to
      UNION
  { ?process rdfs:subClassOf go:GO_0007165 } # refinement of
  ?process rdfs:label ?label
}
```

- The ***`UNION`*** keyword forms a disjunction of two graph patterns. Solutions to both sides of the `UNION` are included in the results.
- The URI `go:GO_0007165` is the identifier for signal transduction in the Gene Ontology
- N.B. *Cell-surface-receptor-linked signal transduction* is a refinement (subclass) of *signal transduction*

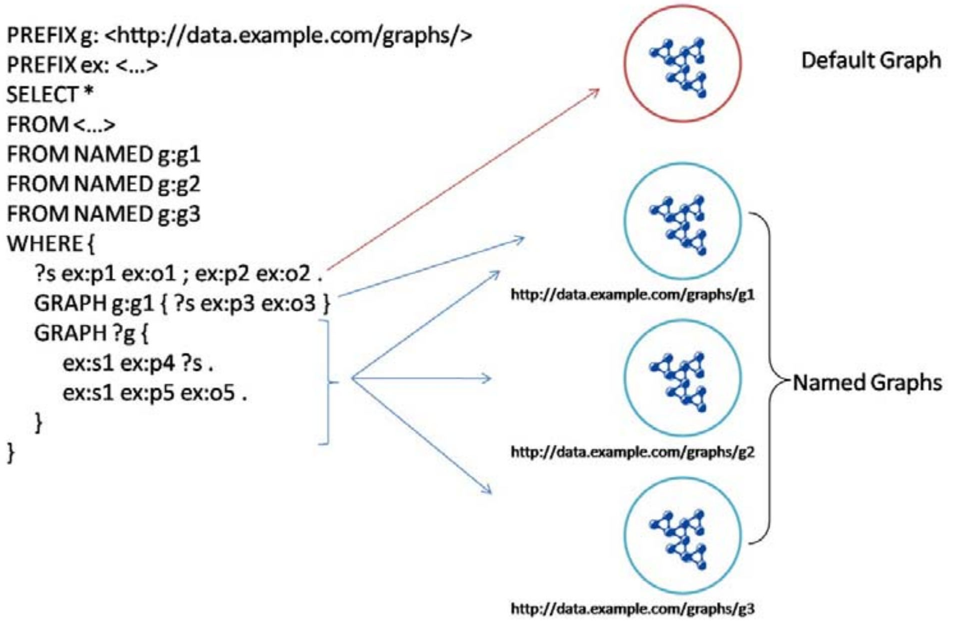
Try it with the HCLS knowledgebase SPARQL endpoint. (Expected results.)



RDF Datasets

- We said earlier that SPARQL queries are executed against *RDF datasets*, consisting of RDF graphs.
- So far, all of our queries have been against a single graph. In SPARQL, this is known as the *default graph*.
- RDF datasets are composed of the default graph and zero or more *named graphs*, identified by a URI.
- Named graphs can be specified with one or more ***`FROM NAMED`*** clauses, or they can be hardwired into a particular SPARQL endpoint.
- The SPARQL ***`GRAPH`*** keyword allows portions of a query to match against the named graphs in the RDF dataset. Anything outside a `GRAPH` clause matches against the default graph.

RDF Datasets

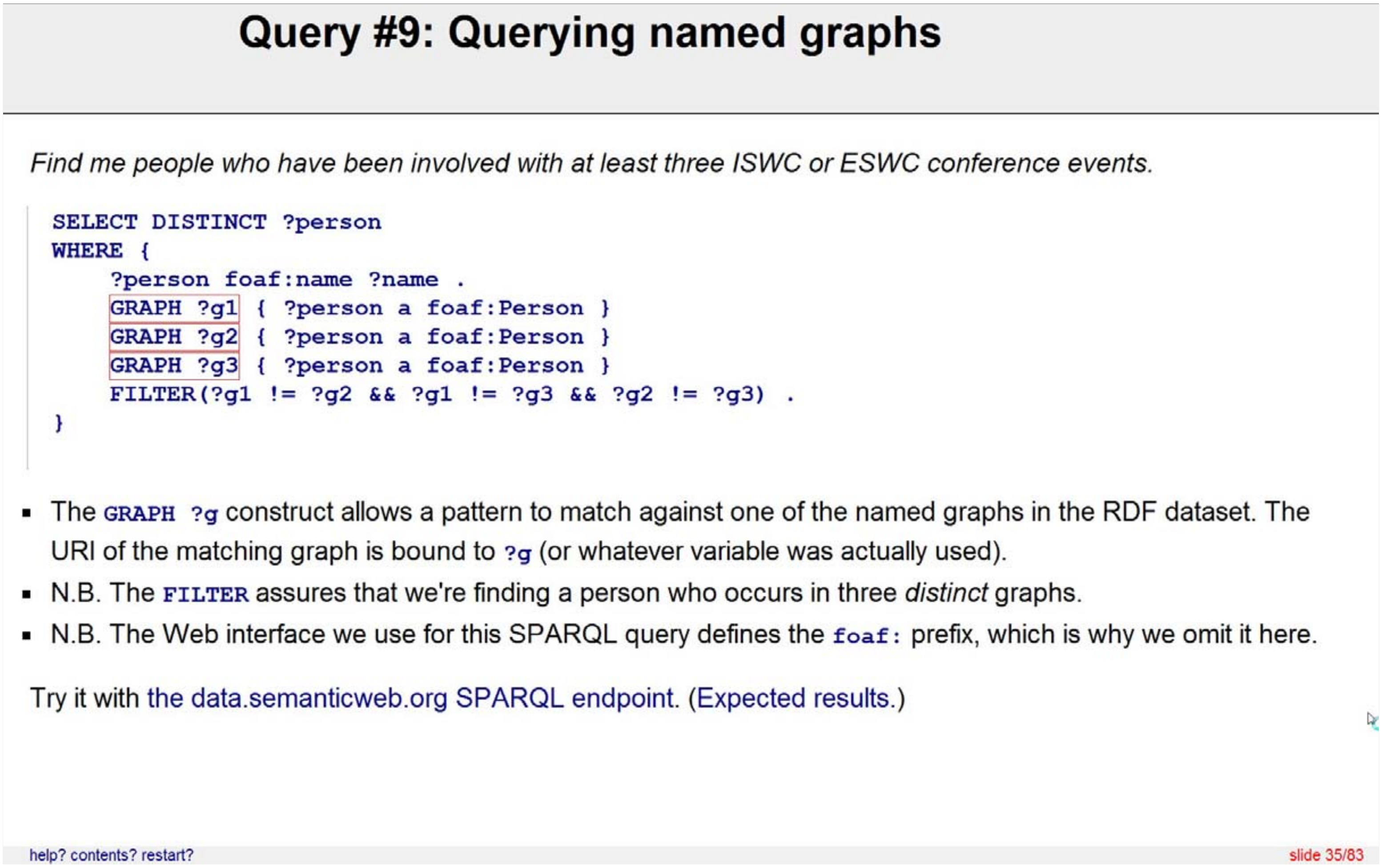


```
PREFIX g: <http://data.example.com/graphs/>
PREFIX ex: <...>
SELECT *
FROM <...>
FROM NAMED g:g1
FROM NAMED g:g2
FROM NAMED g:g3
WHERE {
   ?s ex:p1 ex:o1 ; ex:p2 ex:o2 .
   GRAPH g:g1 { ?s ex:p3 ex:o3 }
   GRAPH ?g {
      ex:s1 ex:p4 ?s .
      ex:s1 ex:p5 ex:o5 .
   }
}
```

Default Graph
http://data.example.com/graphs/g1
http://data.example.com/graphs/g2
http://data.example.com/graphs/g3
Named Graphs

Dataset: semanticweb.org

- data.semanticweb.org hosts RDF data regarding workshops, schedules, and presenters for the International Semantic Web (ISWC) and European Semantic Web Conference (ESWC) series of events.
- Presents data via FOAF, SWRC, and iCal ontologies.
- The data for each individual ISWC or ESWC event is stored in its own named graph; that is, there is one named graph per conference event contained in this dataset.



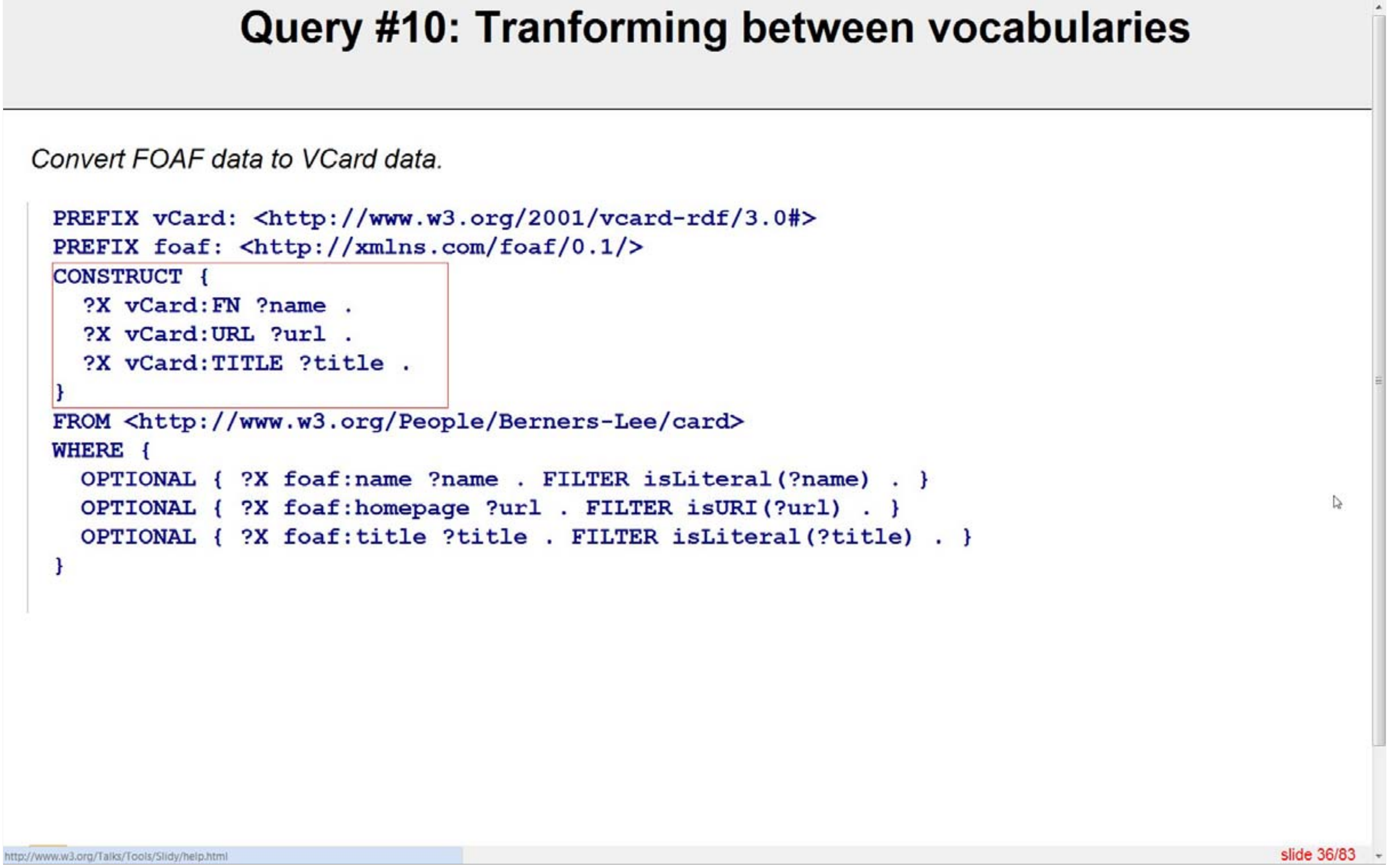
Query #9: Querying named graphs

Find me people who have been involved with at least three ISWC or ESWC conference events.

```
SELECT DISTINCT ?person
WHERE {
    ?person foaf:name ?name .
    GRAPH ?g1 { ?person a foaf:Person }
    GRAPH ?g2 { ?person a foaf:Person }
    GRAPH ?g3 { ?person a foaf:Person }
    FILTER(?g1 != ?g2 && ?g1 != ?g3 && ?g2 != ?g3) .
}
```

- The `GRAPH ?g` construct allows a pattern to match against one of the named graphs in the RDF dataset. The URI of the matching graph is bound to `?g` (or whatever variable was actually used).
- N.B. The `FILTER` assures that we're finding a person who occurs in three *distinct* graphs.
- N.B. The Web interface we use for this SPARQL query defines the `foaf:` prefix, which is why we omit it here.

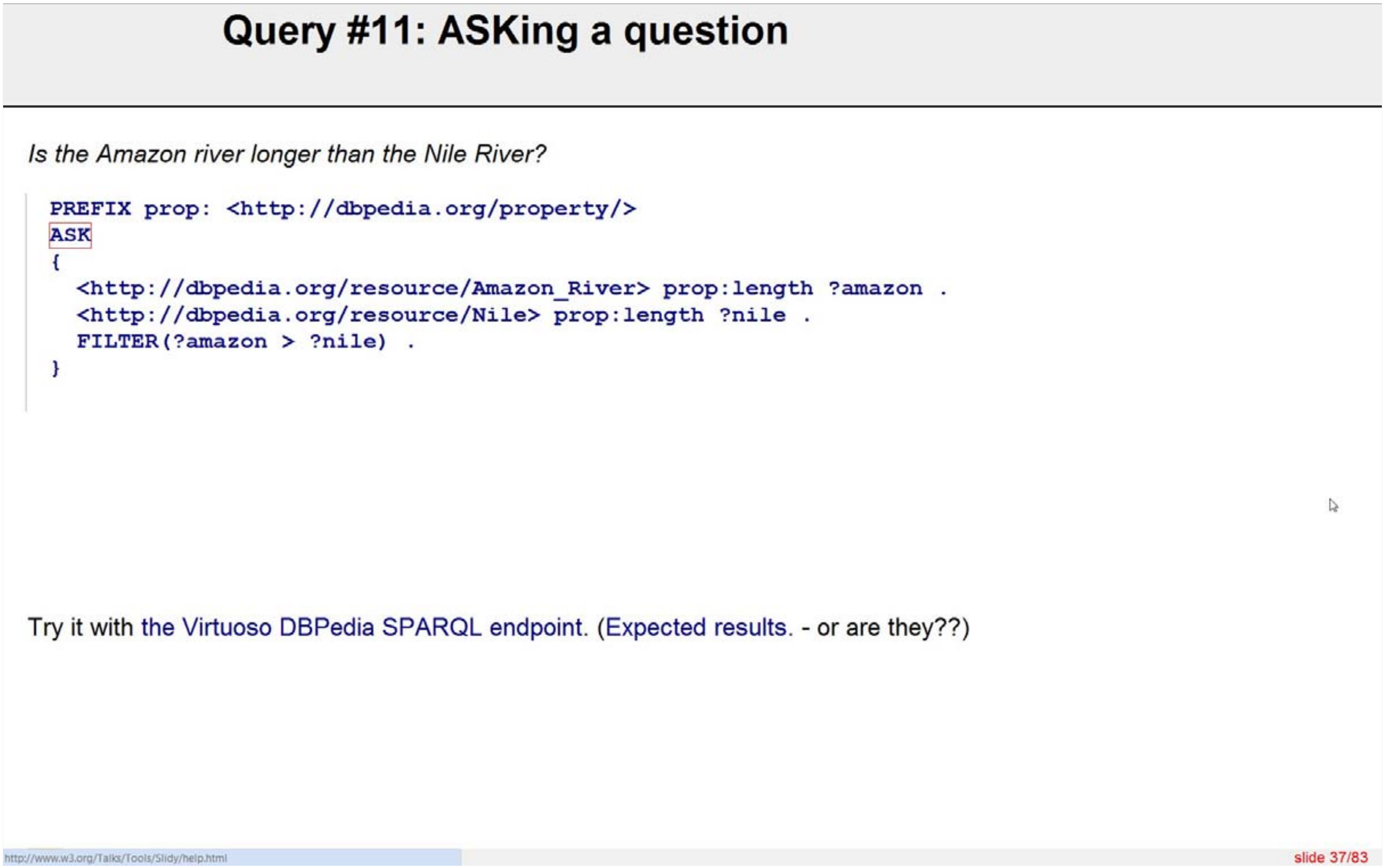
Try it with the data.semanticweb.org SPARQL endpoint. (Expected results.)



Query #10: Tranforming between vocabularies

Convert FOAF data to VCard data.

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
CONSTRUCT {
  ?X vCard:FN ?name .
  ?X vCard:URL ?url .
  ?X vCard:TITLE ?title .
}
FROM <http://www.w3.org/People/Berners-Lee/card>
WHERE {
  OPTIONAL { ?X foaf:name ?name . FILTER isLiteral(?name) . }
  OPTIONAL { ?X foaf:homepage ?url . FILTER isURI(?url) . }
  OPTIONAL { ?X foaf:title ?title . FILTER isLiteral(?title) . }
}
```

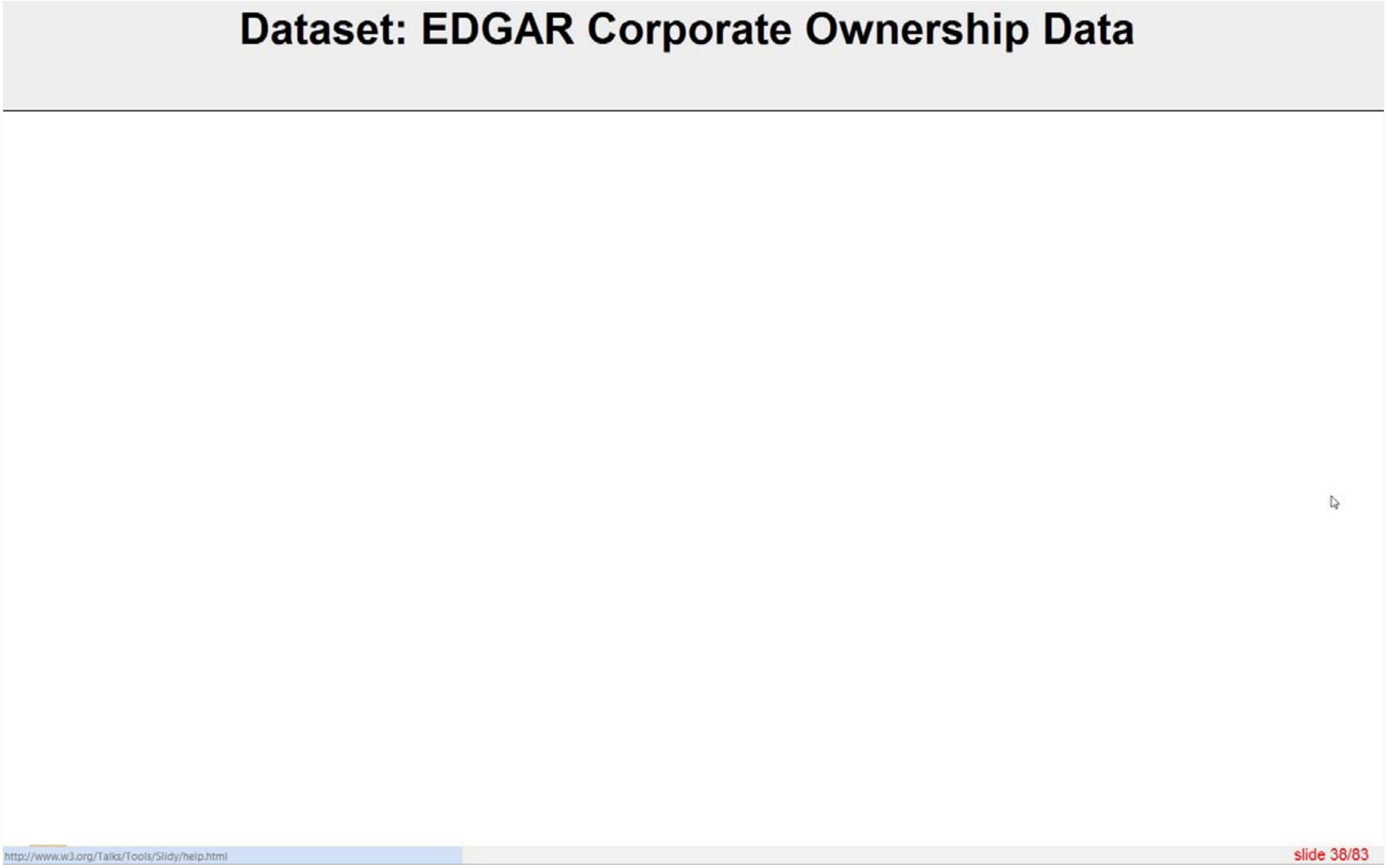


Query #11: ASKing a question

Is the Amazon river longer than the Nile River?

```
PREFIX prop: <http://dbpedia.org/property/>
ASK
{
  <http://dbpedia.org/resource/Amazon_River> prop:length ?amazon .
  <http://dbpedia.org/resource/Nile> prop:length ?nile .
  FILTER(?amazon > ?nile) .
}
```

Try it with the Virtuoso DBPedia SPARQL endpoint. (Expected results. - or are they??)



Dataset: EDGAR Corporate Ownership Data

Query #12: Learning about a resource

Tell me whatever you'd like to tell me about the Ford Motor Company.

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
DESCRIBE ?ford WHERE {
  ?ford foaf:name "FORD MOTOR CO" .
}
```

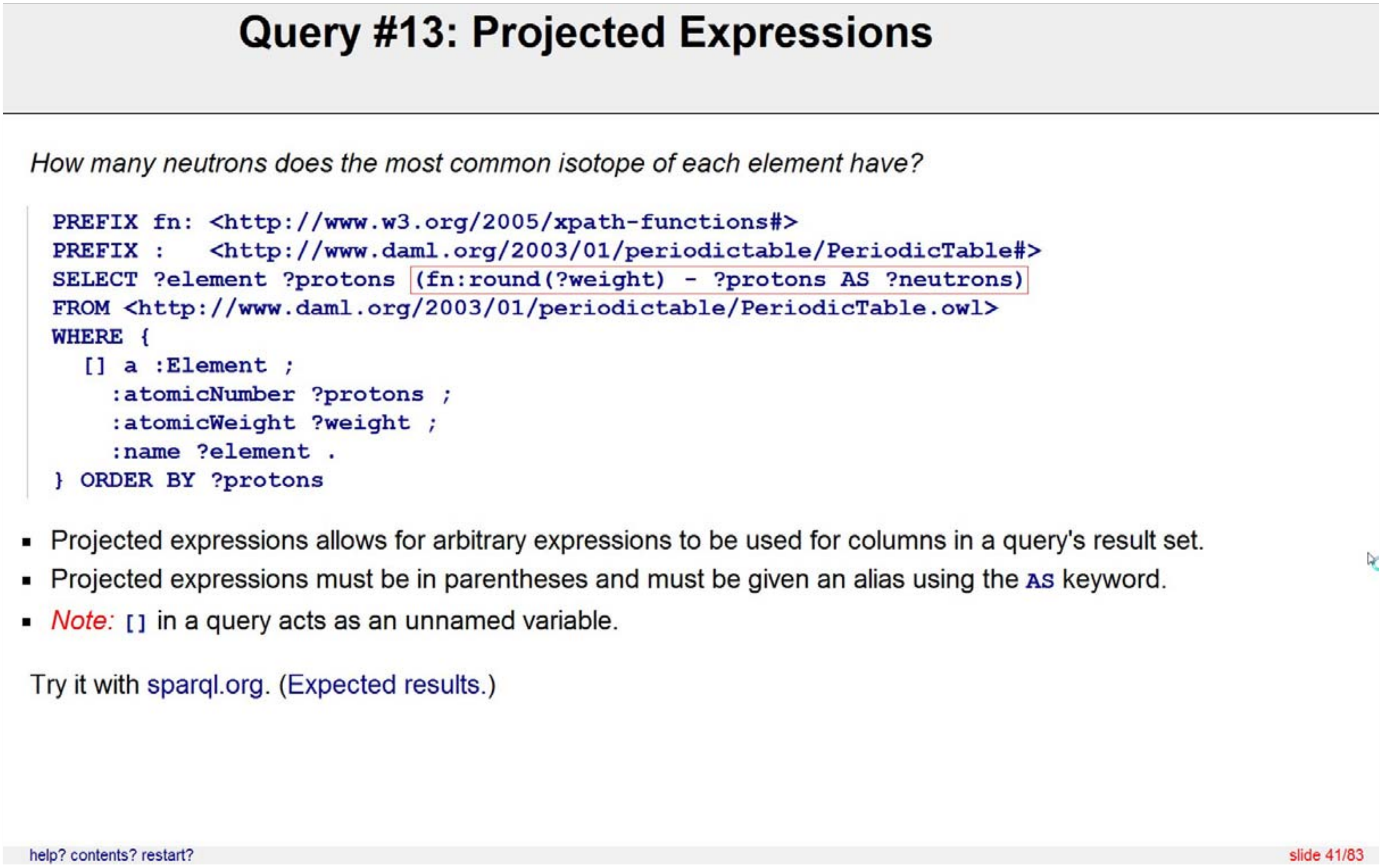
- The ***DESCRIBE*** query result clause allows the server to return whatever RDF it wants that describes the given resource(s).
- Because the server is free to interpret **DESCRIBE** as it sees fit, **DESCRIBE** queries are not interoperable.
- Common implementations include concise-bounded descriptions, named graphs, minimum self-contained graphs, and more.

Try it with the EDGAR-specific SPARQL endpoint. (Expected results.)

What's new in SPARQL 1.1 Query?

A new SPARQL WG was chartered in March 2009 to extend the SPARQL language and landscape. SPARQL 1.1 Query includes these extensions:

- *Projected expressions.* SPARQL 1.1 Query adds the ability for query results to contain values derived from constants, function calls, or other expressions in the SELECT list.
- *Aggregates.* SPARQL 1.1 Query adds the ability to group results and calculate aggregate values (e.g. count, min, max, avg, sum, ...).
- *Subqueries.* SPARQL 1.1 Query allows one query to be embedded within another.
- *Negation.* SPARQL 1.1 Query includes improved language syntax for querying negations.
- *Property paths.* SPARQL 1.1 Query adds the ability to query arbitrary length paths through a graph via a regular-expression-like syntax known as property paths.
- *Basic federated query.* SPARQL 1.1 Query defines a mechanism for splitting a single query among multiple SPARQL endpoints and combining together the results from each.



Query #13: Projected Expressions

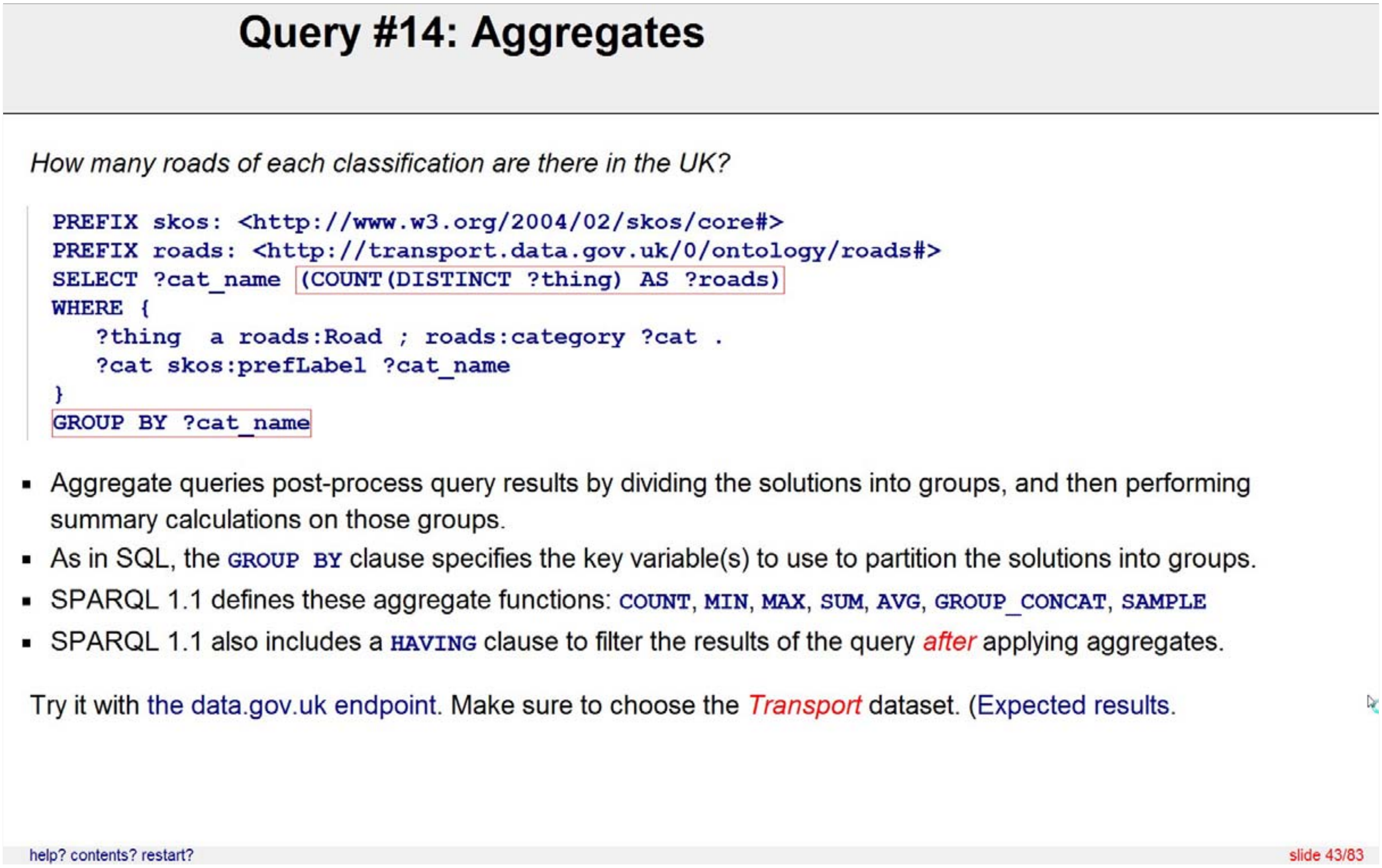
How many neutrons does the most common isotope of each element have?

```
PREFIX fn: <http://www.w3.org/2005/xpath-functions#>
PREFIX :   <http://www.daml.org/2003/01/periodictable/PeriodicTable#>
SELECT ?element ?protons (fn:round(?weight) - ?protons AS ?neutrons)
FROM <http://www.daml.org/2003/01/periodictable/PeriodicTable.owl>
WHERE {
  [] a :Element ;
    :atomicNumber ?protons ;
    :atomicWeight ?weight ;
    :name ?element .
} ORDER BY ?protons
```

- Projected expressions allows for arbitrary expressions to be used for columns in a query's result set.
- Projected expressions must be in parentheses and must be given an alias using the `AS` keyword.
- *Note:* `[]` in a query acts as an unnamed variable.

Try it with sparql.org. (Expected results.)

Dataset: UK Government Data



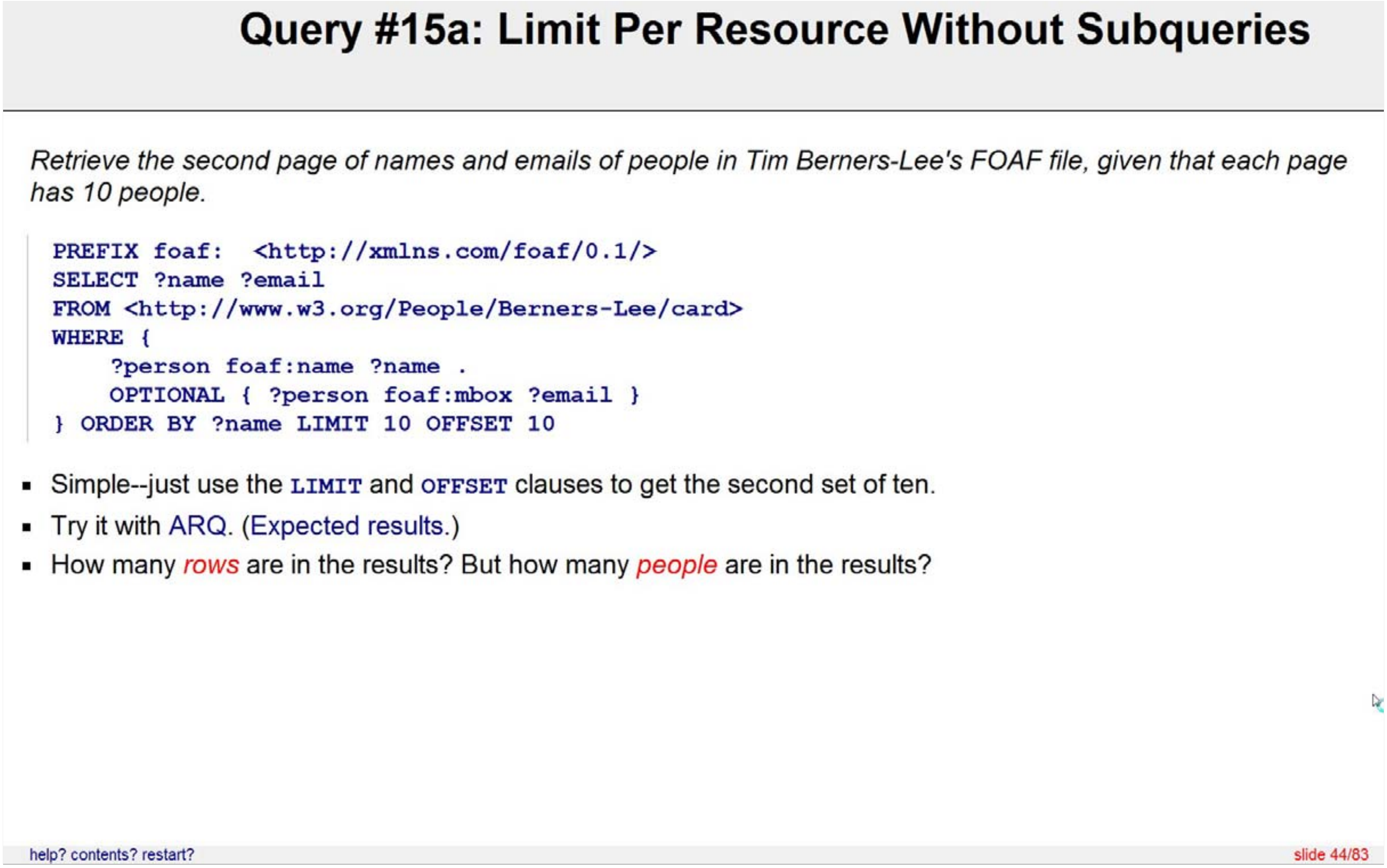
Query #14: Aggregates

How many roads of each classification are there in the UK?

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX roads: <http://transport.data.gov.uk/0/ontology/roads#>
SELECT ?cat_name (COUNT(DISTINCT ?thing) AS ?roads)
WHERE {
   ?thing  a roads:Road ; roads:category ?cat .
   ?cat skos:prefLabel ?cat_name
}
GROUP BY ?cat_name
```

- Aggregate queries post-process query results by dividing the solutions into groups, and then performing summary calculations on those groups.
- As in SQL, the `GROUP BY` clause specifies the key variable(s) to use to partition the solutions into groups.
- SPARQL 1.1 defines these aggregate functions: `COUNT`, `MIN`, `MAX`, `SUM`, `AVG`, `GROUP_CONCAT`, `SAMPLE`
- SPARQL 1.1 also includes a `HAVING` clause to filter the results of the query *after* applying aggregates.

Try it with the data.gov.uk endpoint. Make sure to choose the *Transport* dataset. (Expected results.

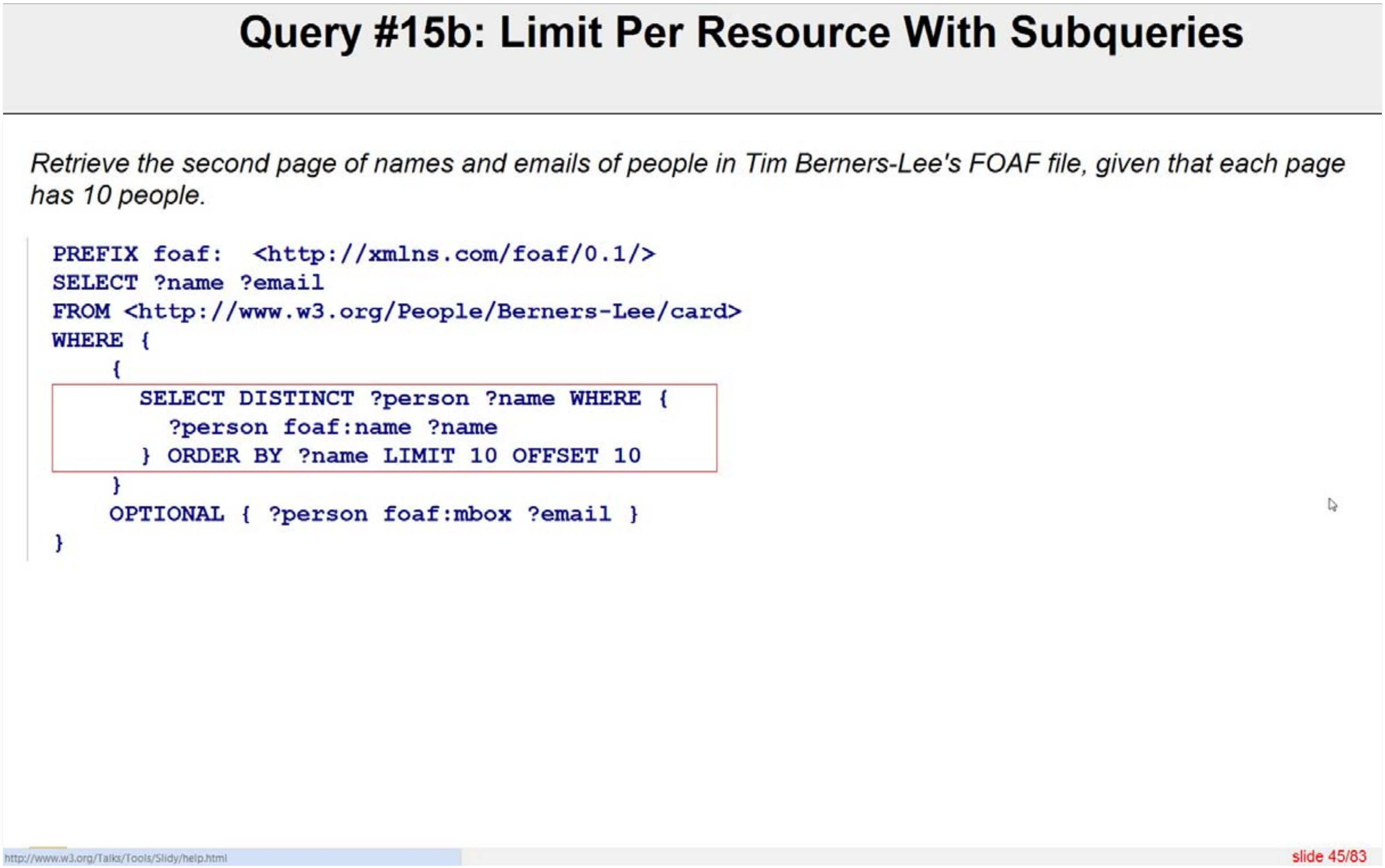


Query #15a: Limit Per Resource Without Subqueries

Retrieve the second page of names and emails of people in Tim Berners-Lee's FOAF file, given that each page has 10 people.

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
FROM <http://www.w3.org/People/Berners-Lee/card>
WHERE {
    ?person foaf:name ?name .
    OPTIONAL { ?person foaf:mbox ?email }
} ORDER BY ?name LIMIT 10 OFFSET 10
```

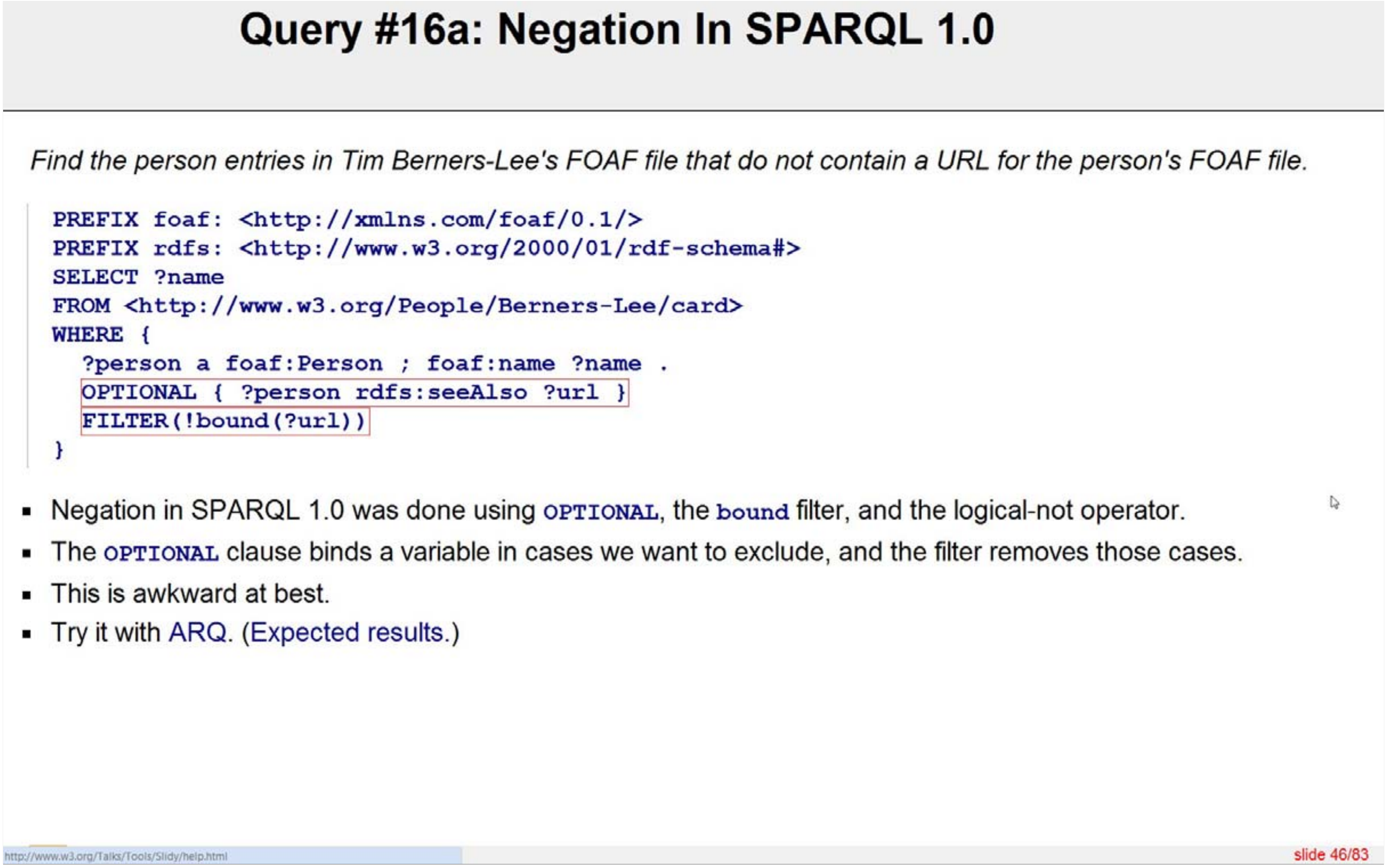
- Simple--just use the `LIMIT` and `OFFSET` clauses to get the second set of ten.
- Try it with ARQ. (Expected results.)
- How many *rows* are in the results? But how many *people* are in the results?



Query #15b: Limit Per Resource With Subqueries

Retrieve the second page of names and emails of people in Tim Berners-Lee's FOAF file, given that each page has 10 people.

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
FROM <http://www.w3.org/People/Berners-Lee/card>
WHERE {
    {
      SELECT DISTINCT ?person ?name WHERE {
        ?person foaf:name ?name
      } ORDER BY ?name LIMIT 10 OFFSET 10
    }
    OPTIONAL { ?person foaf:mbox ?email }
}
```



Query #16a: Negation In SPARQL 1.0

Find the person entries in Tim Berners-Lee's FOAF file that do not contain a URL for the person's FOAF file.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?name
FROM <http://www.w3.org/People/Berners-Lee/card>
WHERE {
  ?person a foaf:Person ; foaf:name ?name .
  OPTIONAL { ?person rdfs:seeAlso ?url }
  FILTER(!bound(?url))
}
```

- Negation in SPARQL 1.0 was done using `OPTIONAL`, the `bound` filter, and the logical-not operator.
- The `OPTIONAL` clause binds a variable in cases we want to exclude, and the filter removes those cases.
- This is awkward at best.
- Try it with ARQ. (Expected results.)

Query #16b: Negation In SPARQL 1.1 (Part 1)

Find the person entries in Tim Berners-Lee's FOAF file that do not contain a URL for the person's FOAF file.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?name
FROM <http://www.w3.org/People/Berners-Lee/card>
WHERE {
  ?person a foaf:Person ; foaf:name ?name .
  MINUS { ?person rdfs:seeAlso ?url }
}
```

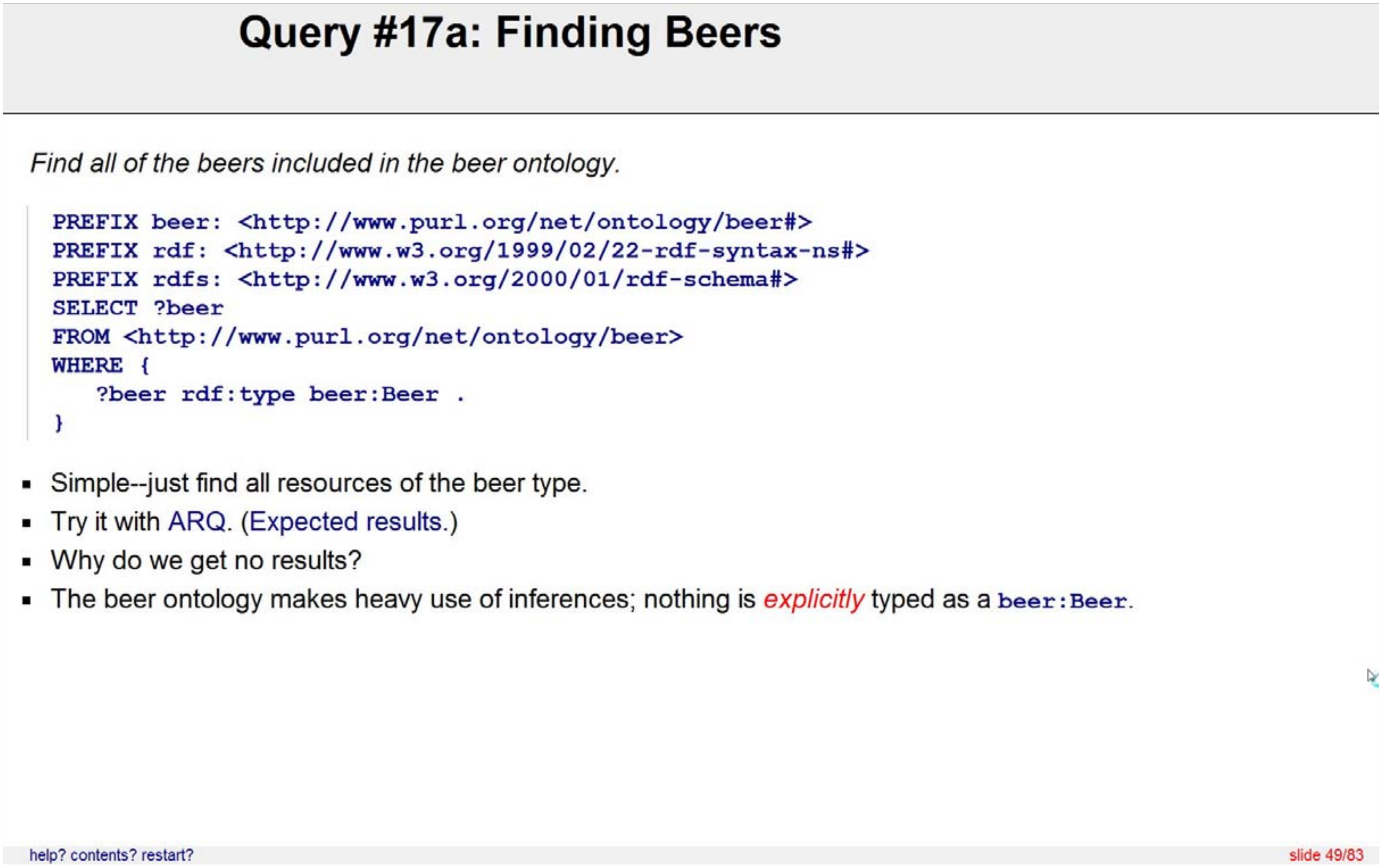
- SPARQL 1.1 includes a `MINUS` graph pattern clause: a binary operator that removes bindings that match the right-hand side.
- *No publicly deployed endpoints support* `MINUS`*, yet!*

Query #16c: Negation In SPARQL 1.1 (Part 2)

Find the person entries in Tim Berners-Lee's FOAF file that do not contain a URL for the person's FOAF file.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?name
FROM <http://www.w3.org/People/Berners-Lee/card>
WHERE {
  ?person a foaf:Person ; foaf:name ?name .
  FILTER(NOT EXISTS { ?person rdfs:seeAlso ?url })
}
```

- SPARQL 1.1 includes a `NOT EXISTS` filter that uses the bindings from a solution to test whether or not a given graph pattern exists.
- In most cases, negation can be done with either `MINUS` or `NOT EXISTS` -- there are some differences in edge cases, though!
- Try it with ARQ. (Expected results.)

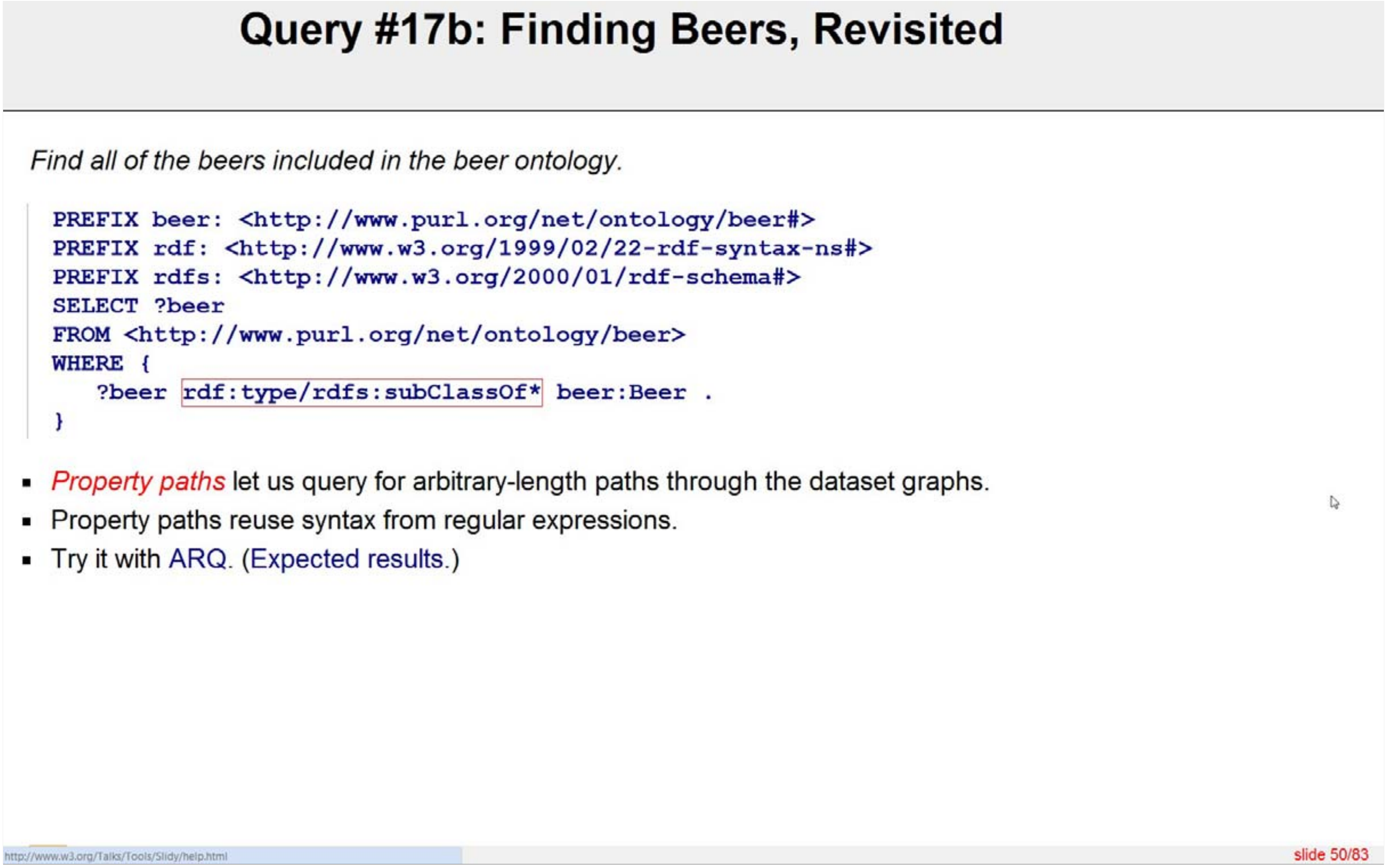


Query #17a: Finding Beers

Find all of the beers included in the beer ontology.

```
PREFIX beer: <http://www.purl.org/net/ontology/beer#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?beer
FROM <http://www.purl.org/net/ontology/beer>
WHERE {
   ?beer rdf:type beer:Beer .
}
```

- Simple--just find all resources of the beer type.
- Try it with ARQ. (Expected results.)
- Why do we get no results?
- The beer ontology makes heavy use of inferences; nothing is *explicitly* typed as a `beer:Beer`.

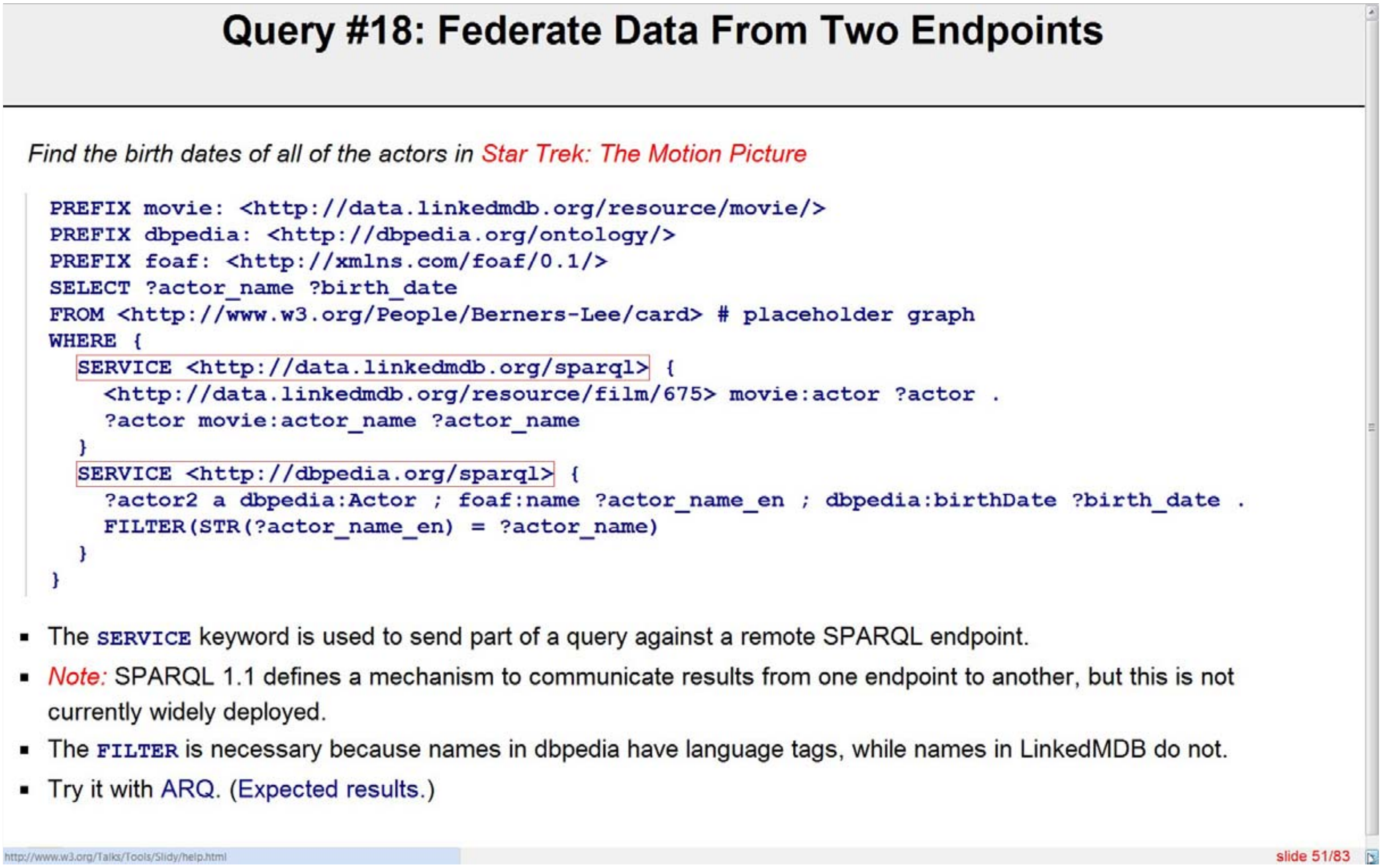


Query #17b: Finding Beers, Revisited

Find all of the beers included in the beer ontology.

```
PREFIX beer: <http://www.purl.org/net/ontology/beer#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?beer
FROM <http://www.purl.org/net/ontology/beer>
WHERE {
   ?beer rdf:type/rdfs:subClassOf* beer:Beer .
}
```

- *Property paths* let us query for arbitrary-length paths through the dataset graphs.
- Property paths reuse syntax from regular expressions.
- Try it with ARQ. (Expected results.)



Query #18: Federate Data From Two Endpoints

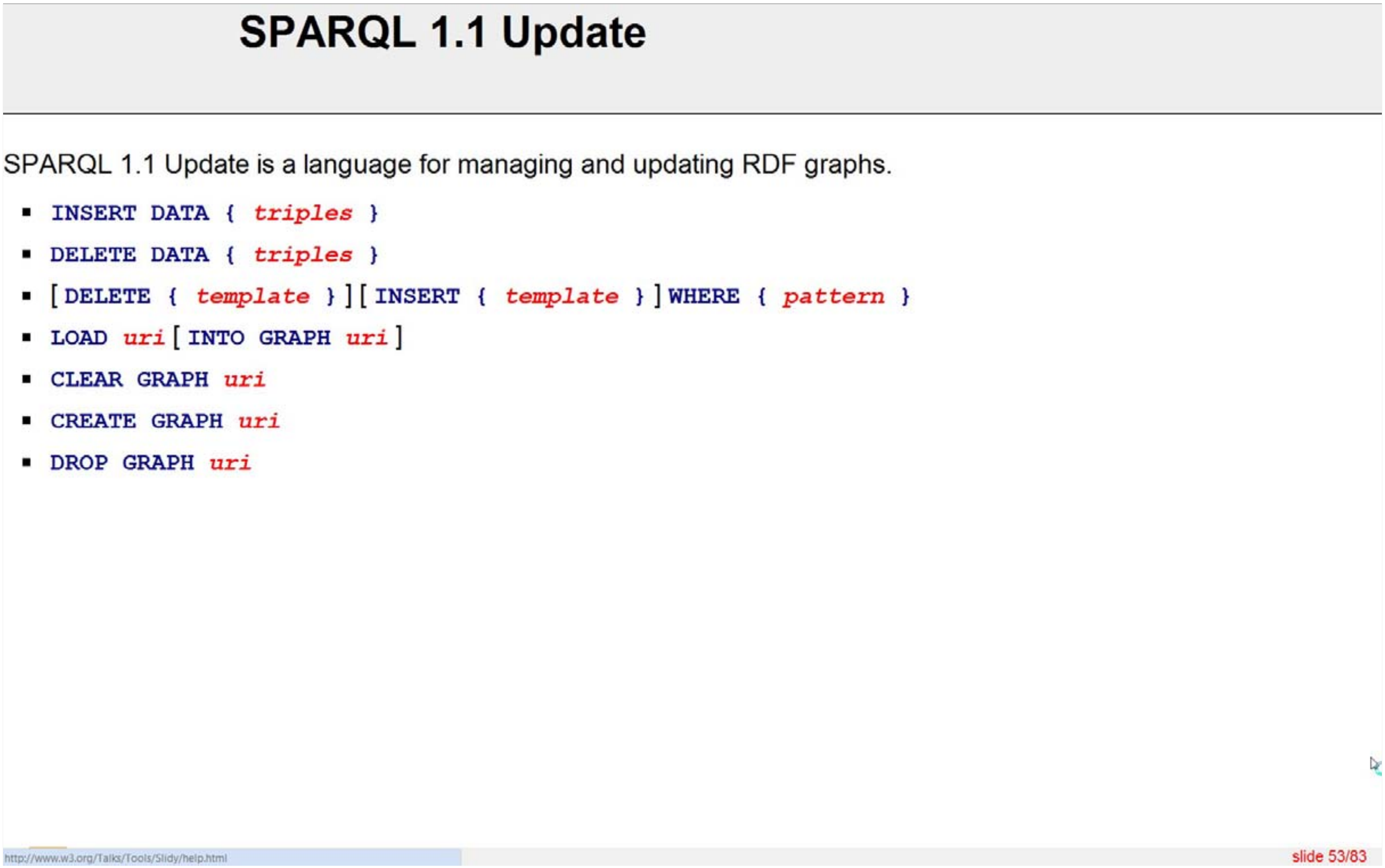
Find the birth dates of all of the actors in Star Trek: The Motion Picture

```
PREFIX movie: <http://data.linkedmdb.org/resource/movie/>
PREFIX dbpedia: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?actor_name ?birth_date
FROM <http://www.w3.org/People/Berners-Lee/card> # placeholder graph
WHERE {
  SERVICE <http://data.linkedmdb.org/sparql> {
    <http://data.linkedmdb.org/resource/film/675> movie:actor ?actor .
    ?actor movie:actor_name ?actor_name
  }
  SERVICE <http://dbpedia.org/sparql> {
    ?actor2 a dbpedia:Actor ; foaf:name ?actor_name_en ; dbpedia:birthDate ?birth_date .
    FILTER(STR(?actor_name_en) = ?actor_name)
  }
}
```

- The `SERVICE` keyword is used to send part of a query against a remote SPARQL endpoint.
- *Note:* SPARQL 1.1 defines a mechanism to communicate results from one endpoint to another, but this is not currently widely deployed.
- The `FILTER` is necessary because names in dbpedia have language tags, while names in LinkedMDB do not.
- Try it with ARQ. (Expected results.)

What else is new in SPARQL 1.1?

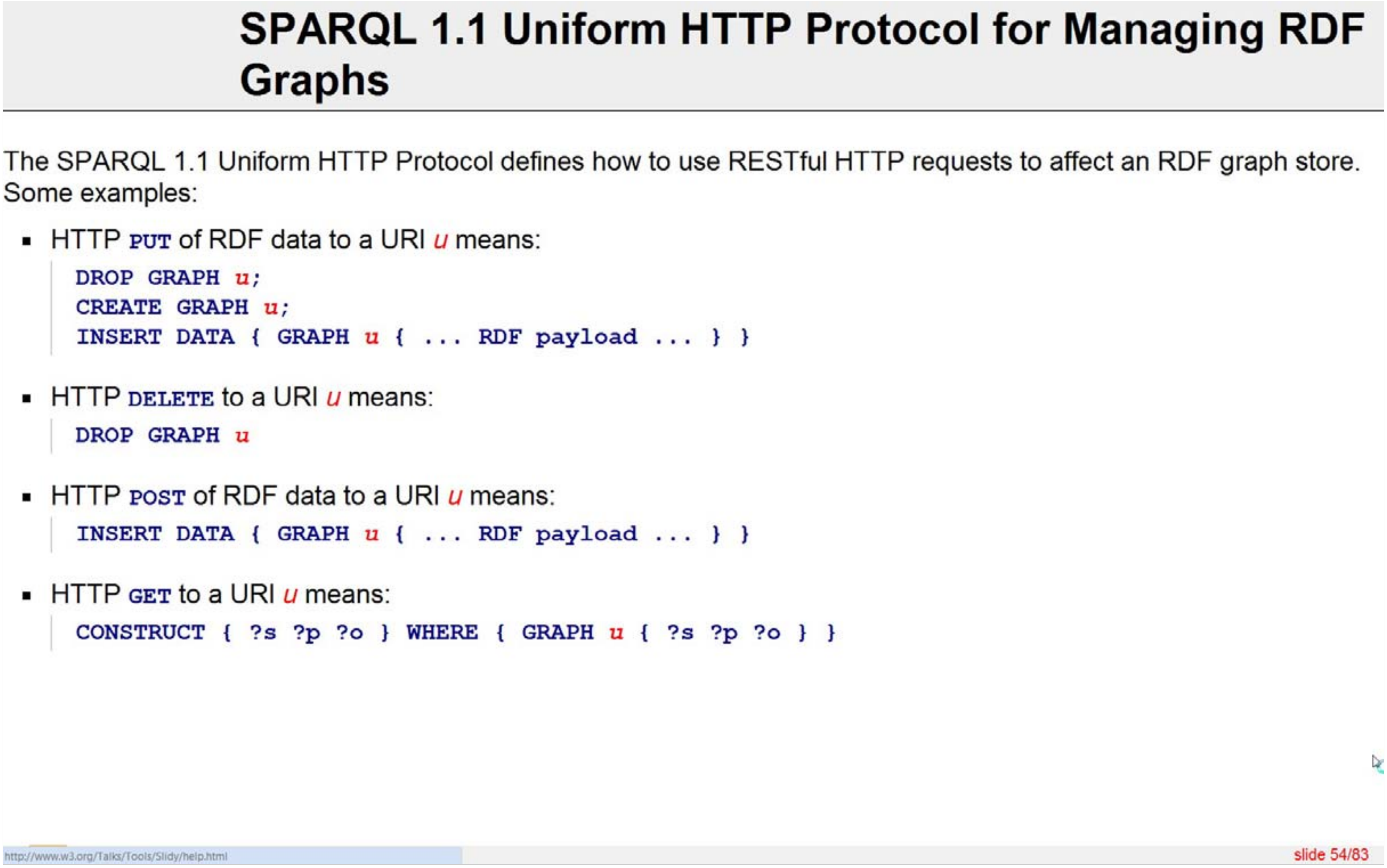
The new SPARQL WG is also extending the SPARQL landscape with:



SPARQL 1.1 Update

SPARQL 1.1 Update is a language for managing and updating RDF graphs.

- `INSERT DATA { triples }`
- `DELETE DATA { triples }`
- [`DELETE { template }`] [`INSERT { template }`] `WHERE { pattern }`
- `LOAD uri` [`INTO GRAPH uri`]
- `CLEAR GRAPH uri`
- `CREATE GRAPH uri`
- `DROP GRAPH uri`



SPARQL 1.1 Uniform HTTP Protocol for Managing RDF Graphs

The SPARQL 1.1 Uniform HTTP Protocol defines how to use RESTful HTTP requests to affect an RDF graph store. Some examples:

- HTTP `PUT` of RDF data to a URI *u* means:

  ```
  DROP GRAPH u;
  CREATE GRAPH u;
  INSERT DATA { GRAPH u { ... RDF payload ... } }
  ```

- HTTP `DELETE` to a URI *u* means:

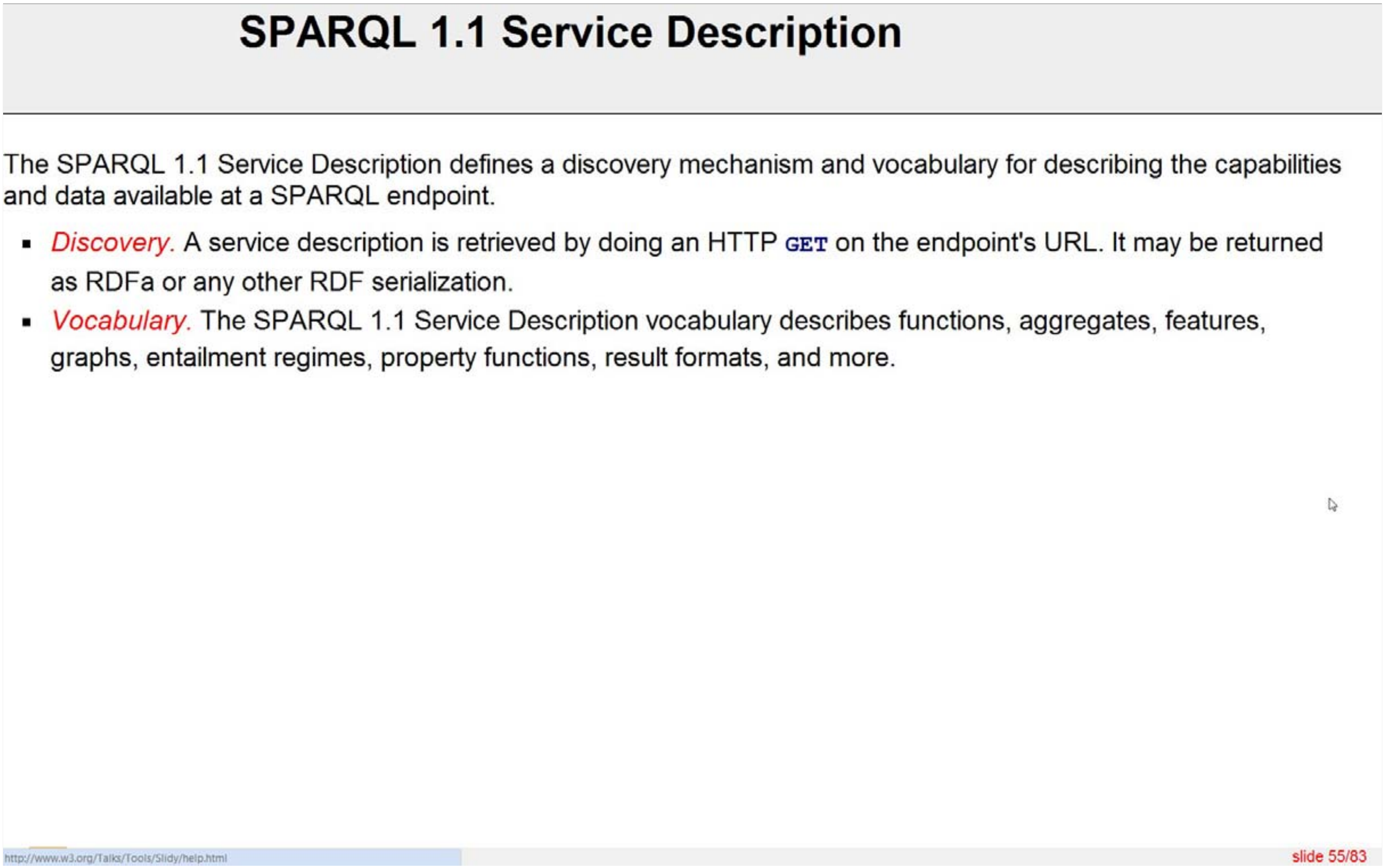
  ```
  DROP GRAPH u
  ```

- HTTP `POST` of RDF data to a URI *u* means:

  ```
  INSERT DATA { GRAPH u { ... RDF payload ... } }
  ```

- HTTP `GET` to a URI *u* means:

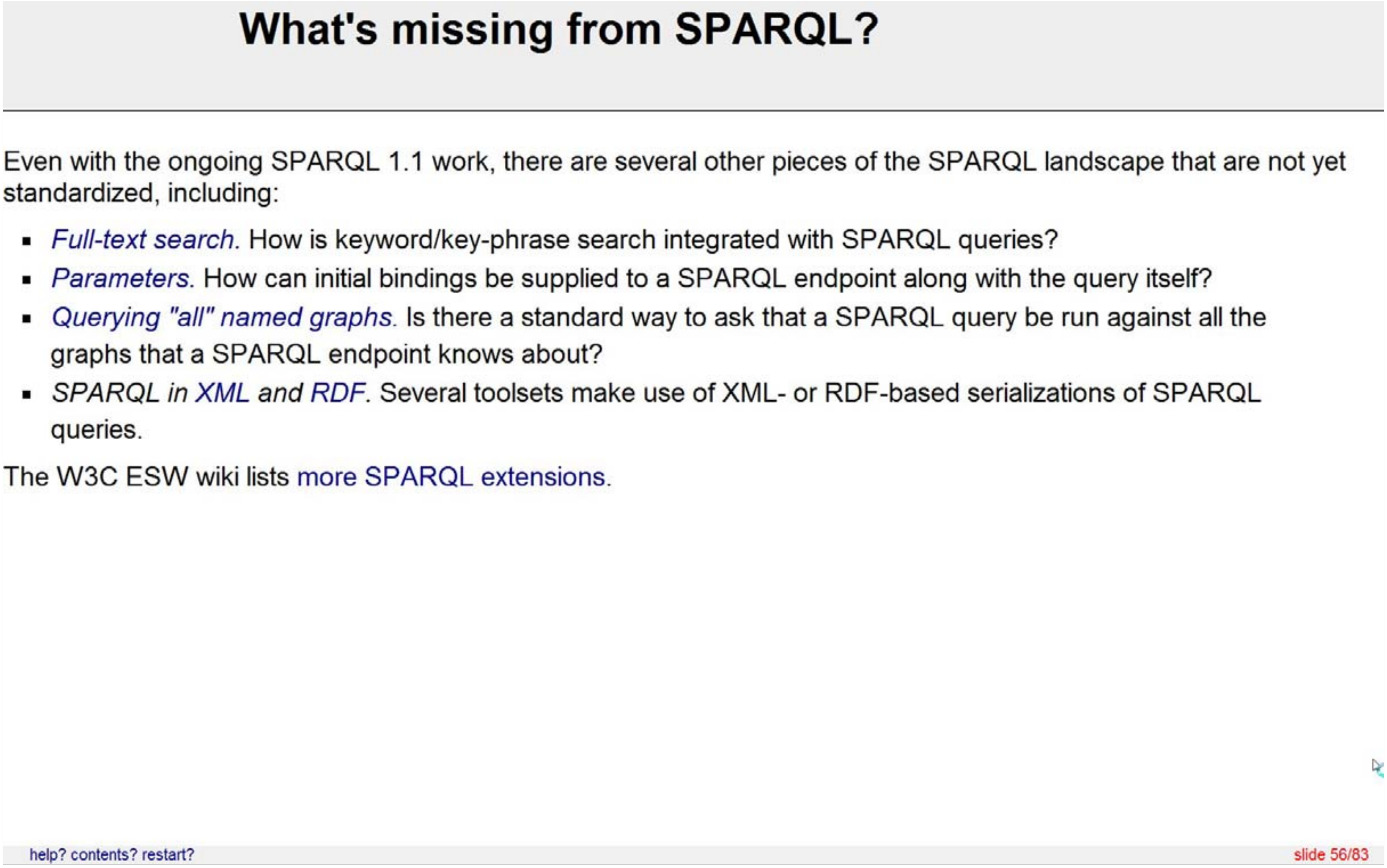
  ```
  CONSTRUCT { ?s ?p ?o } WHERE { GRAPH u { ?s ?p ?o } }
  ```



SPARQL 1.1 Service Description

The SPARQL 1.1 Service Description defines a discovery mechanism and vocabulary for describing the capabilities and data available at a SPARQL endpoint.

- *Discovery.* A service description is retrieved by doing an HTTP `GET` on the endpoint's URL. It may be returned as RDFa or any other RDF serialization.
- *Vocabulary.* The SPARQL 1.1 Service Description vocabulary describes functions, aggregates, features, graphs, entailment regimes, property functions, result formats, and more.

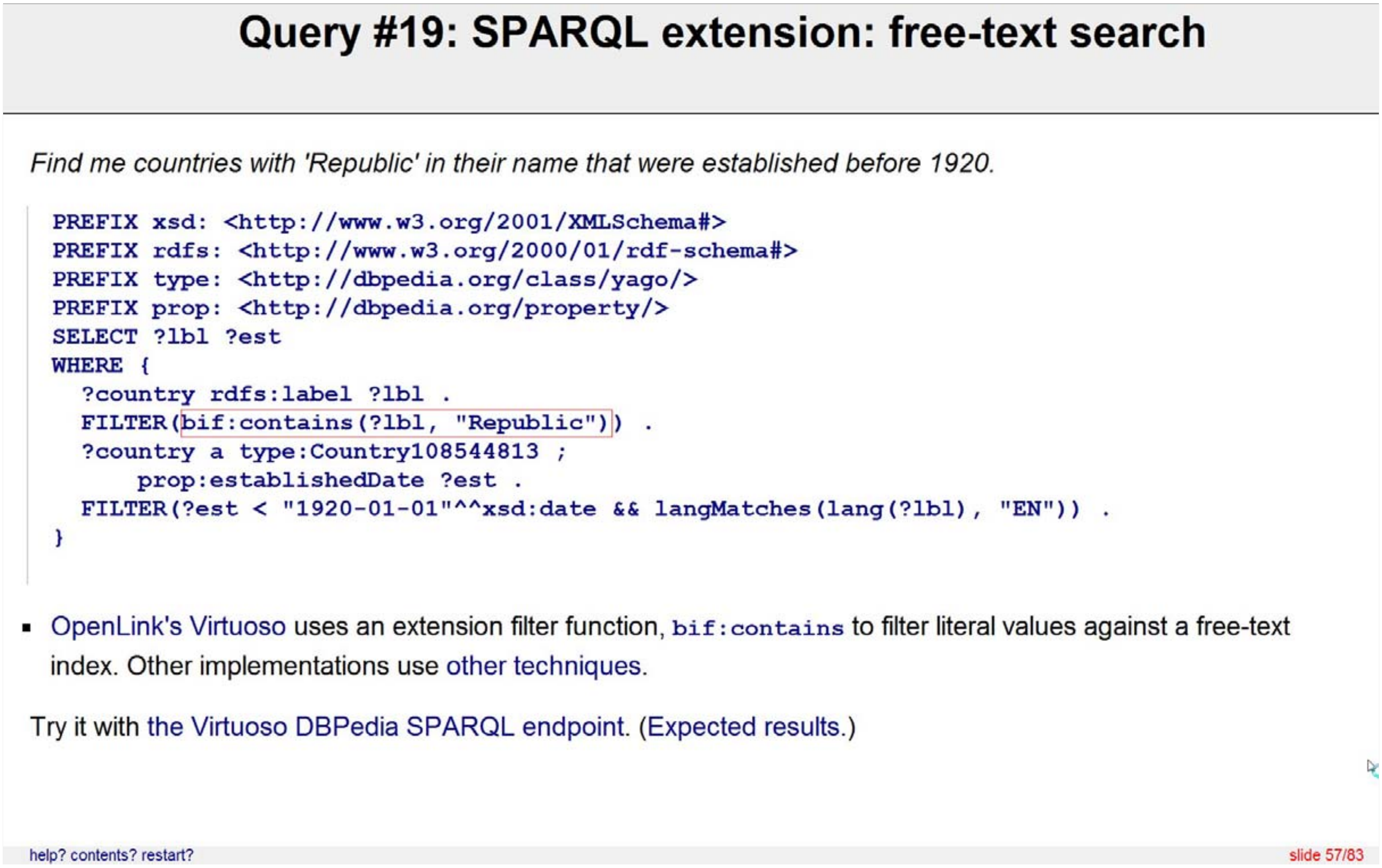


What's missing from SPARQL?

Even with the ongoing SPARQL 1.1 work, there are several other pieces of the SPARQL landscape that are not yet standardized, including:

- *Full-text search.* How is keyword/key-phrase search integrated with SPARQL queries?
- *Parameters.* How can initial bindings be supplied to a SPARQL endpoint along with the query itself?
- *Querying "all" named graphs.* Is there a standard way to ask that a SPARQL query be run against all the graphs that a SPARQL endpoint knows about?
- *SPARQL in XML and RDF.* Several toolsets make use of XML- or RDF-based serializations of SPARQL queries.

The W3C ESW wiki lists more SPARQL extensions.



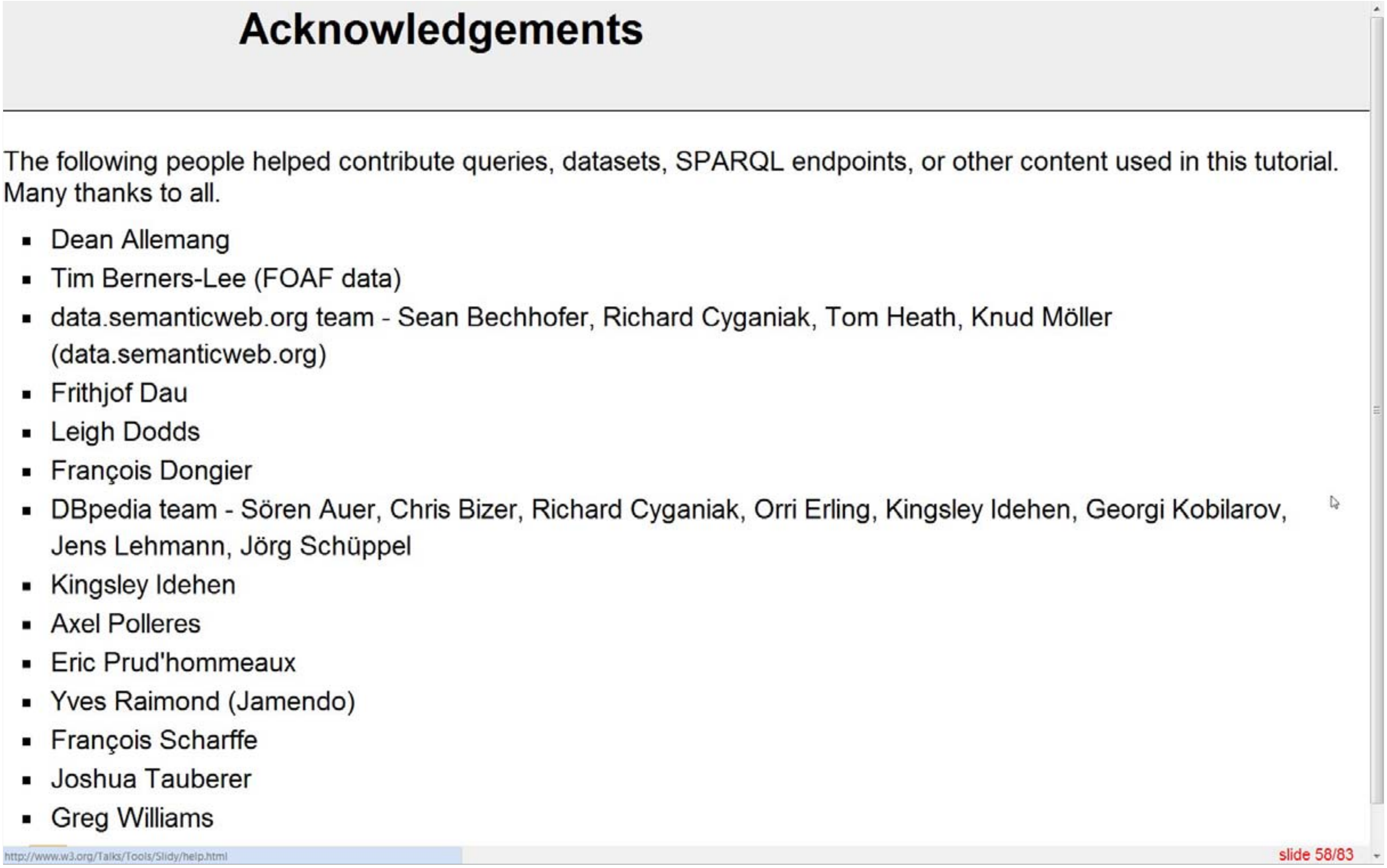
Query #19: SPARQL extension: free-text search

Find me countries with 'Republic' in their name that were established before 1920.

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX type: <http://dbpedia.org/class/yago/>
PREFIX prop: <http://dbpedia.org/property/>
SELECT ?lbl ?est
WHERE {
  ?country rdfs:label ?lbl .
  FILTER(bif:contains(?lbl, "Republic")) .
  ?country a type:Country108544813 ;
      prop:establishedDate ?est .
  FILTER(?est < "1920-01-01"^^xsd:date && langMatches(lang(?lbl), "EN")) .
}
```

- OpenLink's Virtuoso uses an extension filter function, `bif:contains` to filter literal values against a free-text index. Other implementations use other techniques.

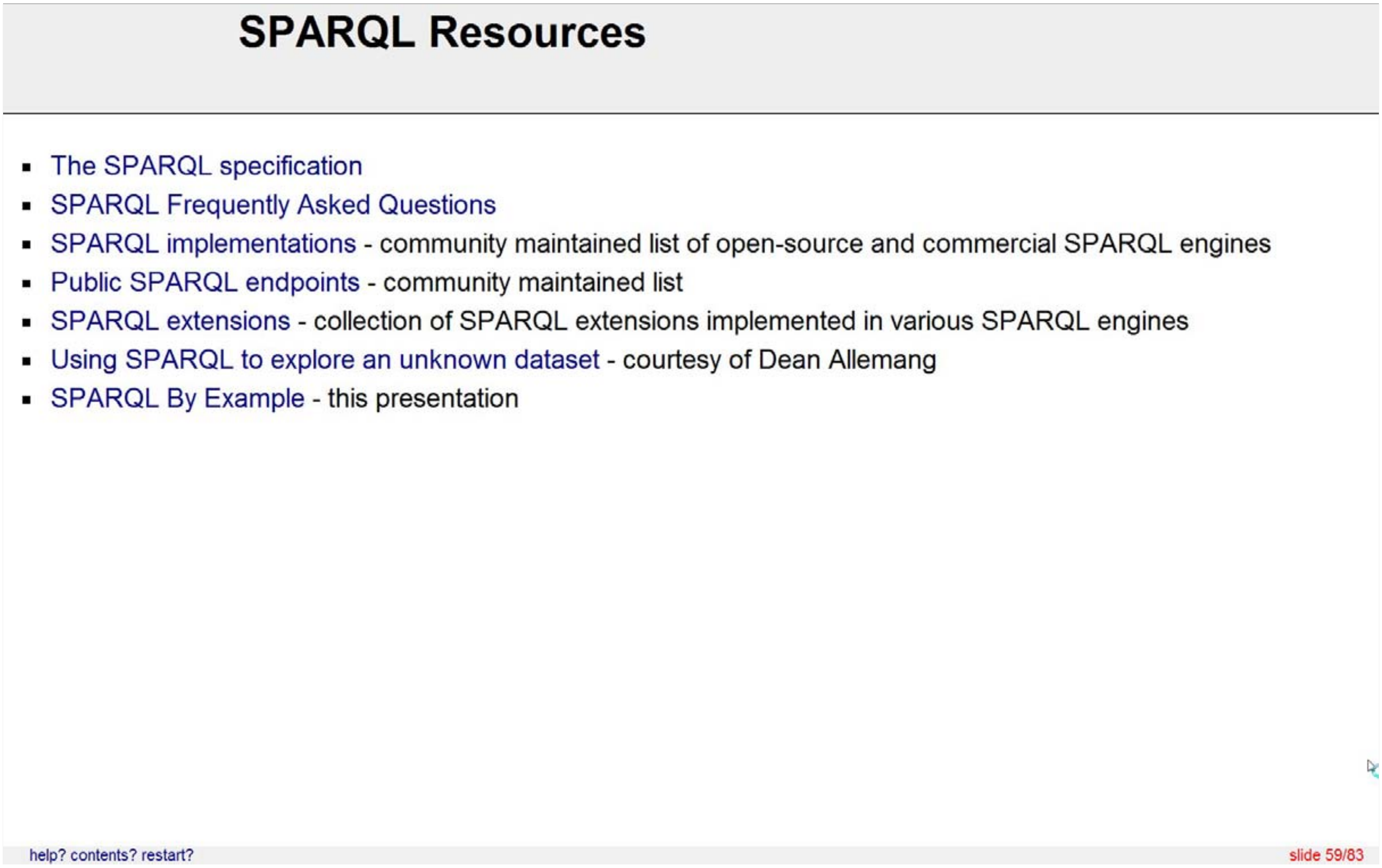
Try it with the Virtuoso DBPedia SPARQL endpoint. (Expected results.)



Acknowledgements

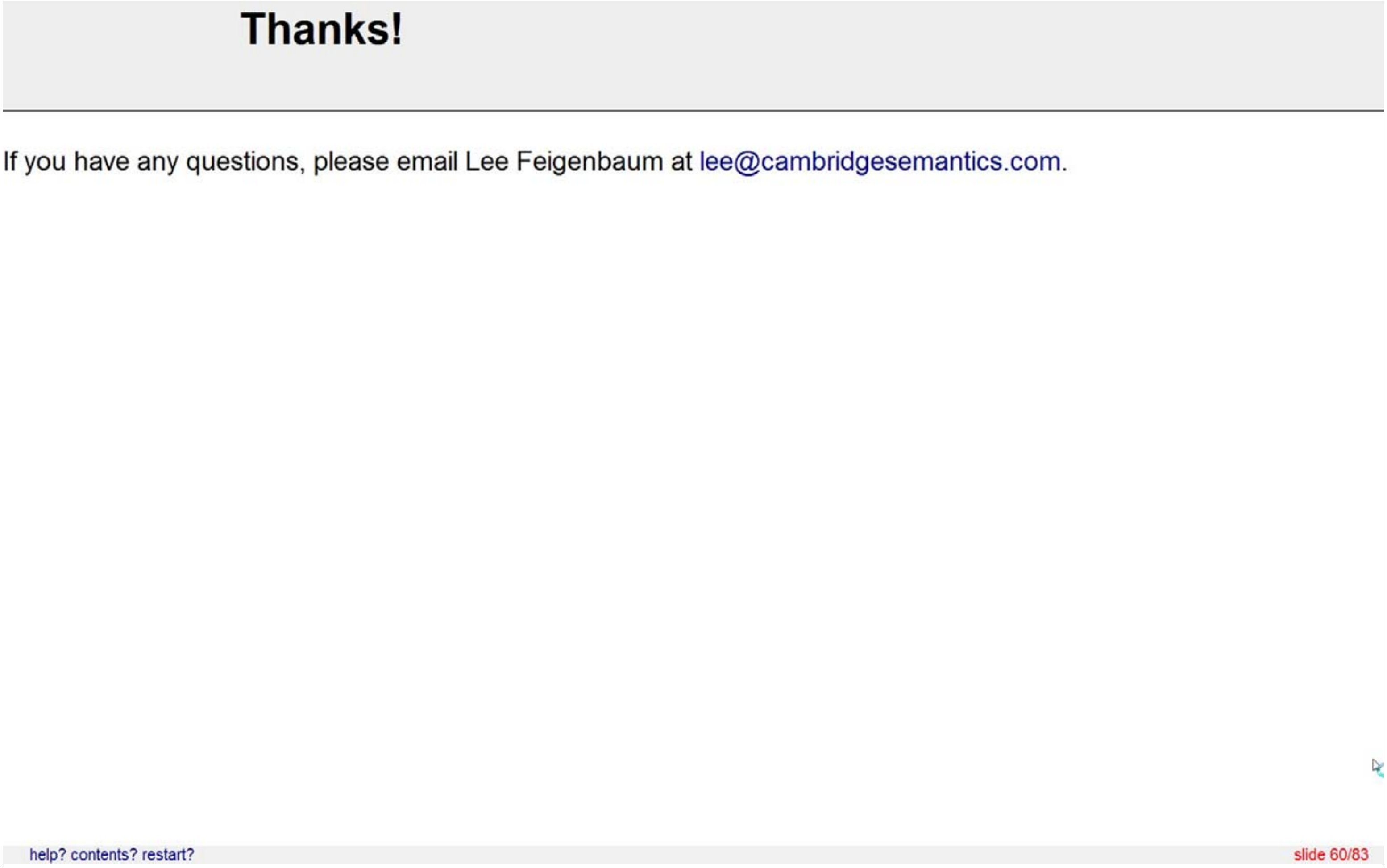
The following people helped contribute queries, datasets, SPARQL endpoints, or other content used in this tutorial. Many thanks to all.

- Dean Allemang
- Tim Berners-Lee (FOAF data)
- data.semanticweb.org team - Sean Bechhofer, Richard Cyganiak, Tom Heath, Knud Möller (data.semanticweb.org)
- Frithjof Dau
- Leigh Dodds
- François Dongier
- DBpedia team - Sören Auer, Chris Bizer, Richard Cyganiak, Orri Erling, Kingsley Idehen, Georgi Kobilarov, Jens Lehmann, Jörg Schüppel
- Kingsley Idehen
- Axel Polleres
- Eric Prud'hommeaux
- Yves Raimond (Jamendo)
- François Scharffe
- Joshua Tauberer
- Greg Williams



SPARQL Resources

- The SPARQL specification
- SPARQL Frequently Asked Questions
- SPARQL implementations - community maintained list of open-source and commercial SPARQL engines
- Public SPARQL endpoints - community maintained list
- SPARQL extensions - collection of SPARQL extensions implemented in various SPARQL engines
- Using SPARQL to explore an unknown dataset - courtesy of Dean Allemang
- SPARQL By Example - this presentation



Thanks!

If you have any questions, please email Lee Feigenbaum at lee@cambridgesemantics.com.