LDOW, WWW'09

April 20, 2009,

Madrid, Spain

MashQL

A Data Mashup Language for the Data Web

Dr. Mustafa Jarrar

mjarrar@cs.ucy.ac.cy

HPCLab, University of Cyprus

Published As:
Mustafa Jarrar and Marios D. Dikaiakos: A Data Mashup Language for the Data Web. Proceedings of LDOW, part of WWW'09. ACM. 2009.

Motivation (Querying the Data Web)

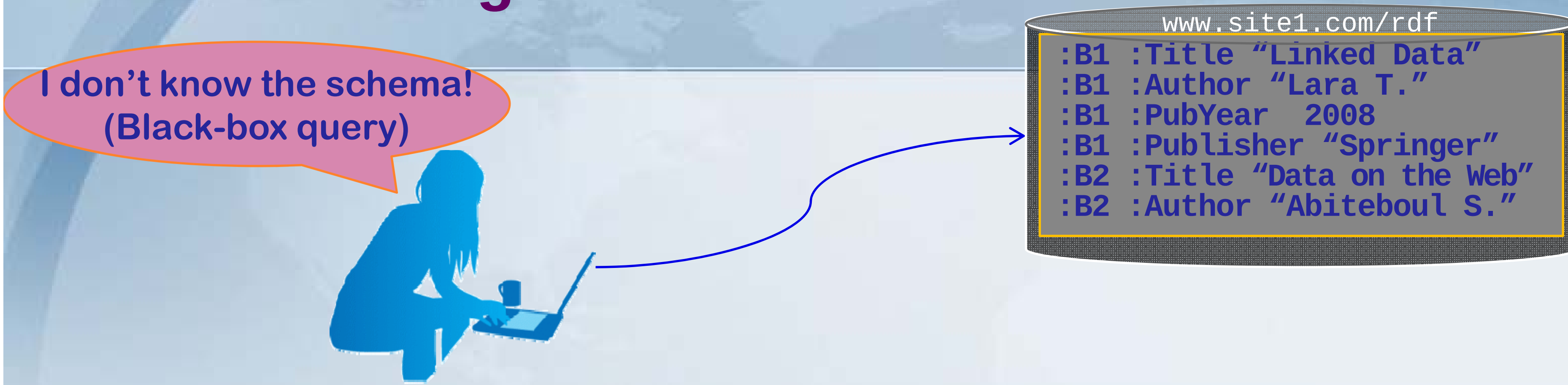
Can we query and mash up the Data Web **<u>as simple as</u>** filtering and piping Web Feeds?

- ➔ We *fundamentally* investigate this problem from a **Query Formulation** viewpoint.
- ➔ A "data mashup" is a query.

Outline

- Challenges
- Related Work
- MashQL (A Query Formulation Language)
- Evaluation
- Conclusions and Discussion

Challenges

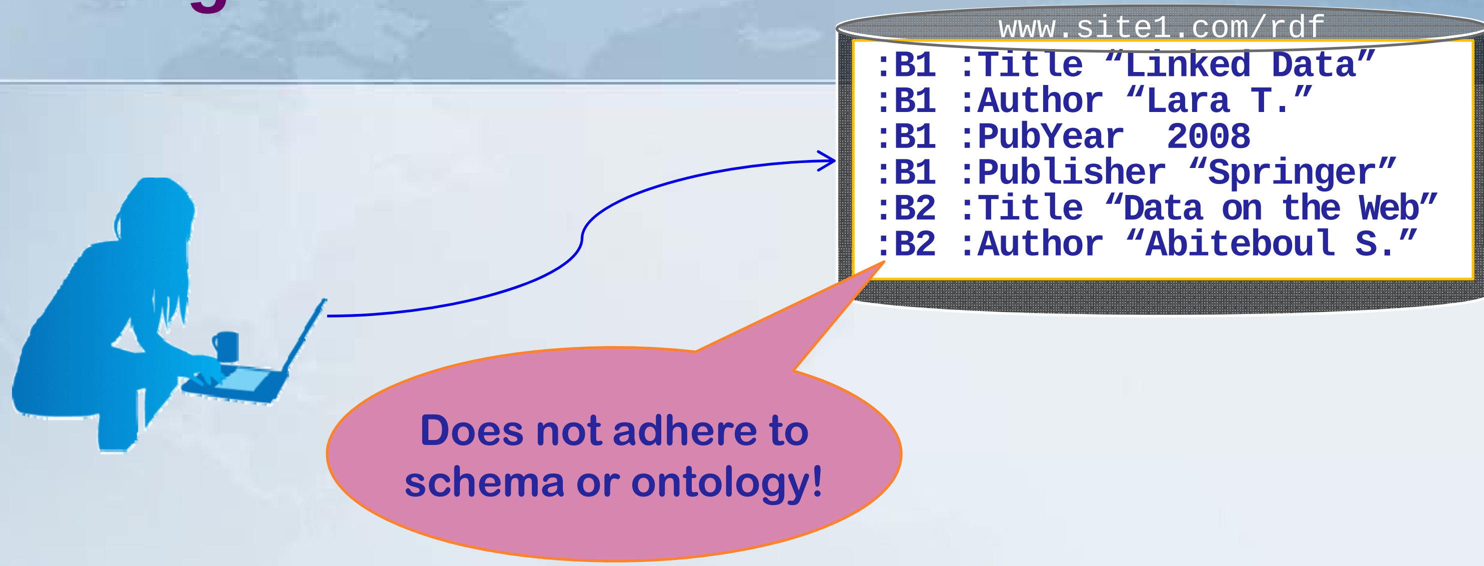


Problem Definition

We need a <u>way</u> to allow <u>*end-users*</u> formulate queries over structured data assuming that:

- *The user does not know the schema.* (1)

Challenges

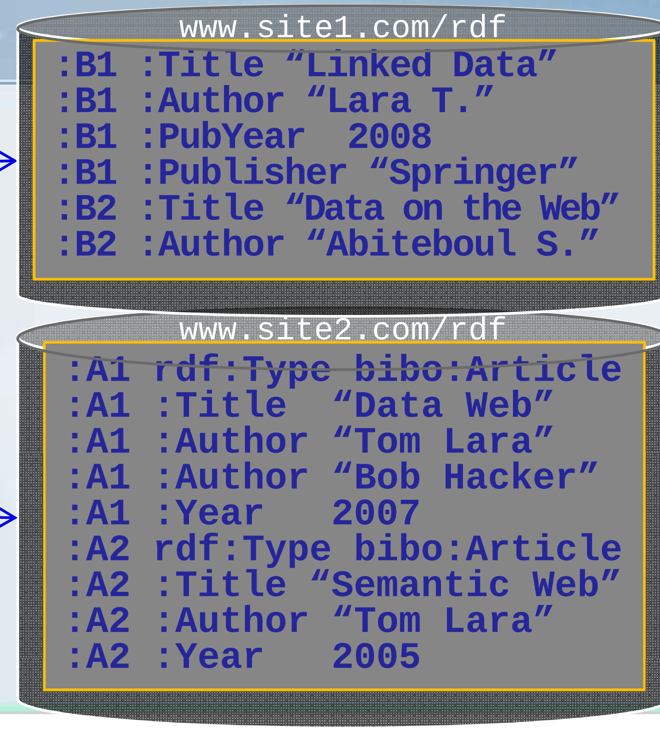


Problem Definition

How to allow *end-users* formulate queries over structured data assuming that:

- *The user does not know the schema.* (1)
- There is no offline or inline schema\ontology. (2)

Challenges



Problem Definition

How to allow *end-users* formulate queries over structured data assuming that:

- *The user does not know the schema.* (1)
- There is no offline or inline schema\ontology. (2)
- The query may involve multiple sources. (3)

Challenges

General Solution!

www.site1.com/rdf

```
:B1 :Title "Linked Data"
:B1 :Author "Lara T."
:B1 :PubYear  2008
:B1 :Publisher "Springer"
:B2 :Title "Data on the Web"
:B2 :Author "Abiteboul S."
```

www.site2.com/rdf

```
:A1 rdf:Type bibo:Article
:A1 :Title  "Data Web"
:A1 :Author "Tom Lara"
:A1 :Author "Bob Hacker"
:A1 :Year   2007
:A2 rdf:Type bibo:Article
:A2 :Title "Semantic Web"
:A2 :Author "Tom Lara"
:A2 :Year   2005
```

Problem Definition

How to allow *end-users* formulate queries over structured data assuming that:

- *The user does not know the schema.* (1)
- There is no offline or inline schema\ontology. (2)
- The query may involve multiple sources. (3)
- The query language is sufficiently expressive (not a single-purpose interface) (4)

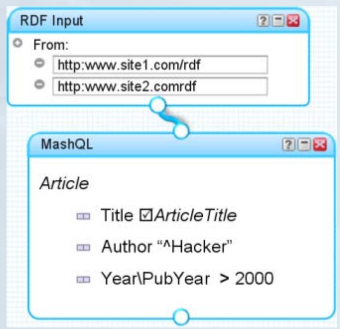
Related Work

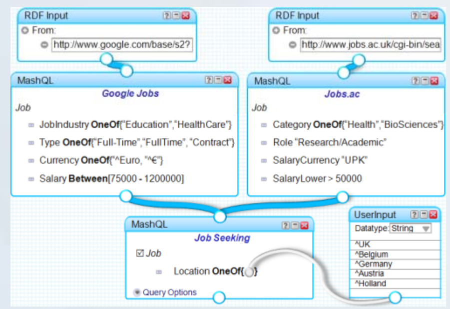
Query formulation is the art of accessing and consuming structured data *easily*. (interdisciplinary subject)

Assumption	Query by Form	Query by Example	Conceptual Queries (ConQuer)	NL Queries	Interactive Queries (Lorel)	Visual Scripting (DeriPipes)
Don't know the schema	✓	✗	✗	✓	✓	✗
Schema-free data	✗	✗	✗	✗	-	✓
Multiple sources	✗	✓	✗	✗	✗	✓
Expressive	✗	✓	✓	✓	-	✓
Intuitive	✓	✗	✗	✓	✓	✗

MashQL

A graphical query formulation Language.





all of these assumptions:

- ✓ *The user does not know the schema.*
- ✓ There is no offline or inline schema\ontology.
- ✓ The query may involve multiple sources.
- ✓ The query language is expressive.
  (not a single-purpose interface)

Challenging combination

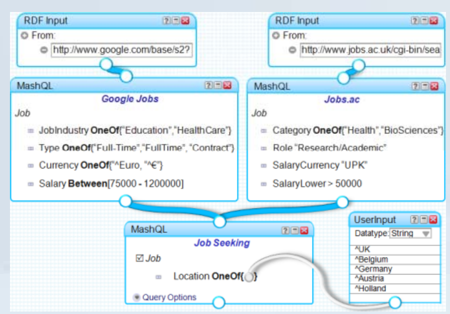
MashQL

A general structured-data retrieval solution.
(not merely an interface)

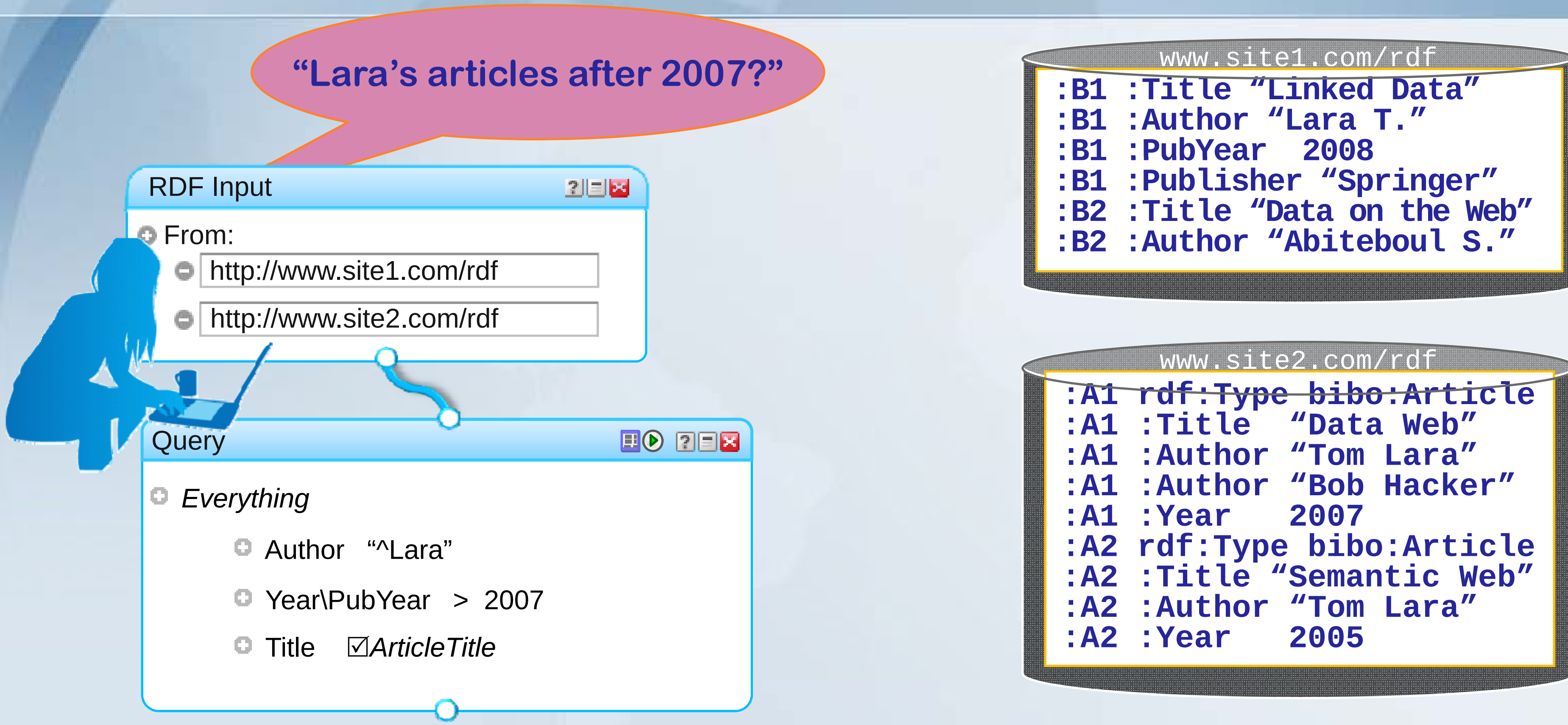
Without loosing this generality, we

- *Focus on RDF, as the most primitive query language, other data models can be mapped into it.*

- *Follow Yahoo Pipes' visualization, in order to illustrate that Data Web can be queried and mashed up as Web Feeds.*

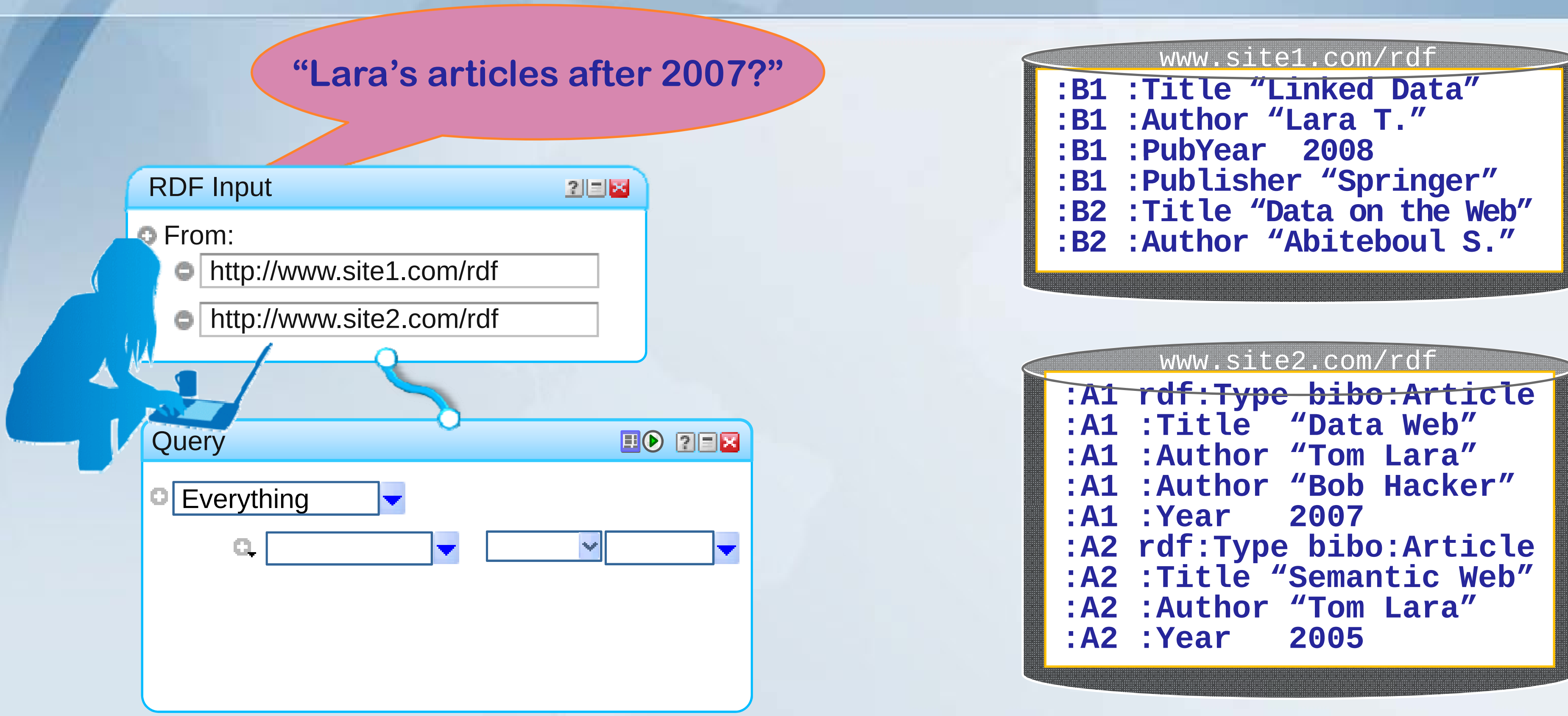


Example 1



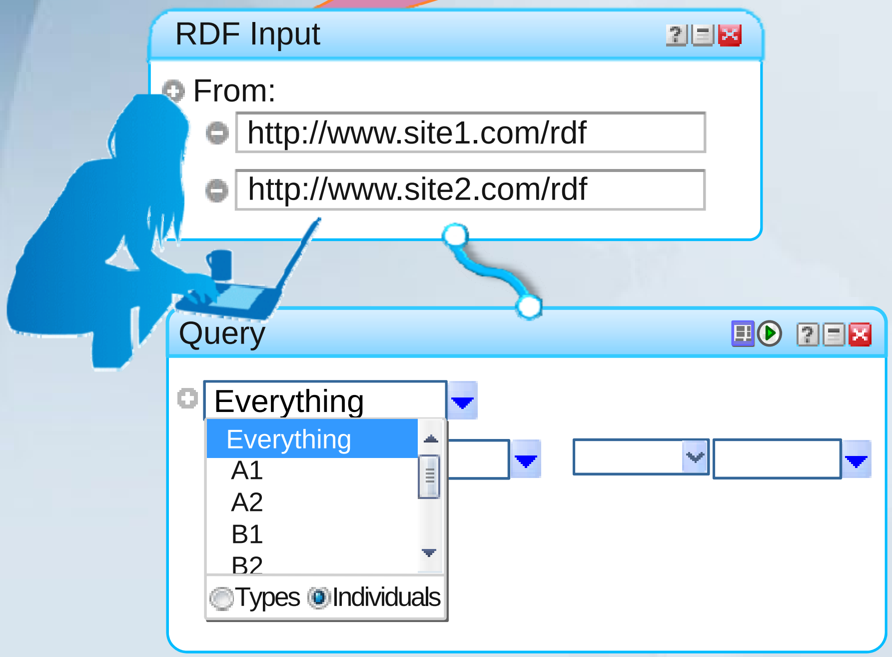
- Interactive Query Formulation.
- MashQL queries are translated into and executed as SPARQL.

Example 1


"Lara's articles after 2007?"
RDF Input
From:
http://www.site1.com/rdf
http://www.site2.com/rdf
Query
Everything
www.site1.com/rdf
:B1 :Title "Linked Data"
:B1 :Author "Lara T."
:B1 :PubYear 2008
:B1 :Publisher "Springer"
:B2 :Title "Data on the Web"
:B2 :Author "Abiteboul S."
www.site2.com/rdf
:A1 rdf:Type bibo:Article
:A1 :Title "Data Web"
:A1 :Author "Tom Lara"
:A1 :Author "Bob Hacker"
:A1 :Year 2007
:A2 rdf:Type bibo:Article
:A2 :Title "Semantic Web"
:A2 :Author "Tom Lara"
:A2 :Year 2005

Example 1

"Lara's articles after 2007?"



www.site1.com/rdf

```
:B1 :Title "Linked Data"
:B1 :Author "Lara T."
:B1 :PubYear  2008
:B1 :Publisher "Springer"
:B2 :Title "Data on the Web"
:B2 :Author "Abiteboul S."
```

www.site2.com/rdf

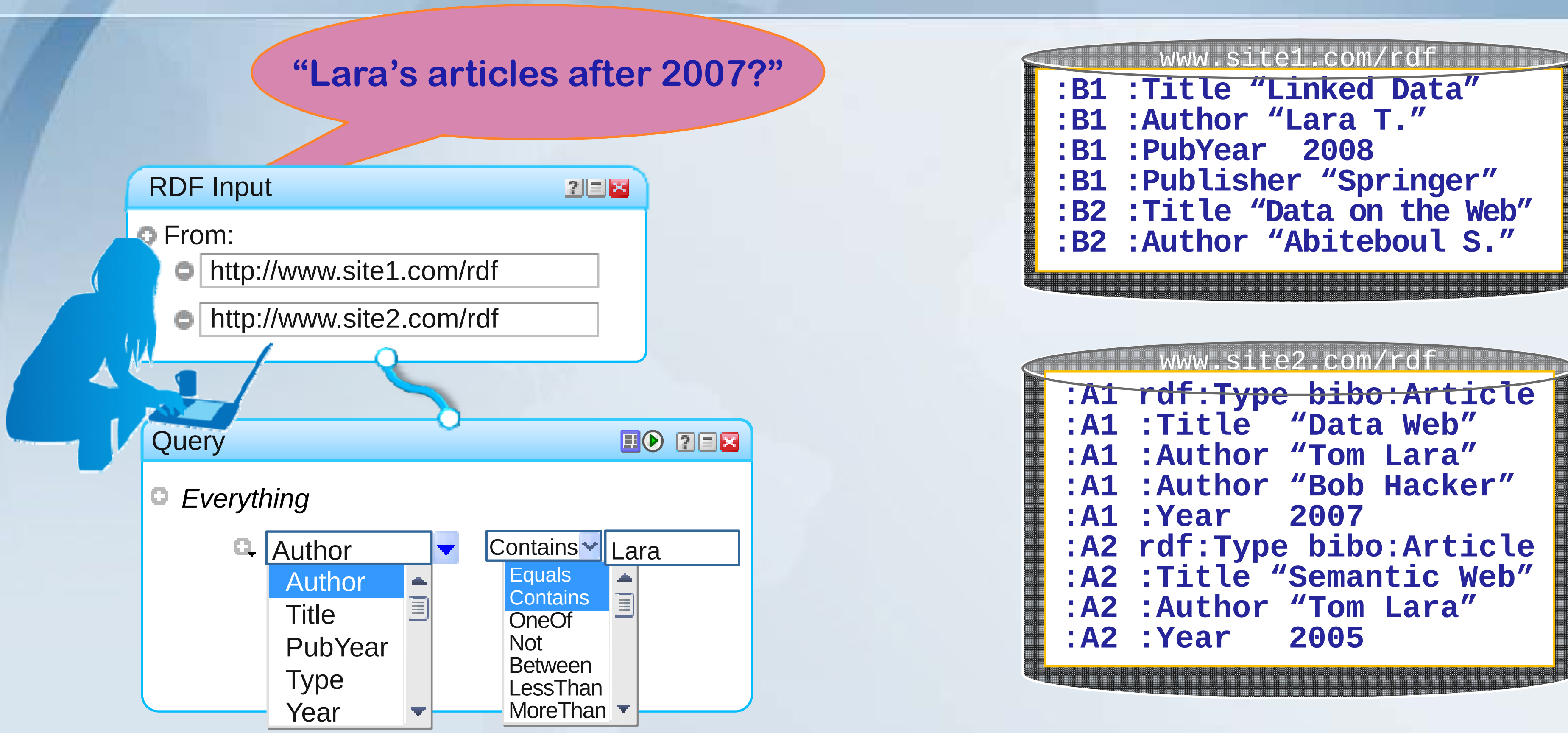
```
:A1 rdf:Type bibo:Article
:A1 :Title  "Data Web"
:A1 :Author "Tom Lara"
:A1 :Author "Bob Hacker"
:A1 :Year   2007
:A2 rdf:Type bibo:Article
:A2 :Title "Semantic Web"
:A2 :Author "Tom Lara"
:A2 :Year   2005
```

Background queries

```
SELECT X WHERE {?X ?P ?O}
Union
SELECT X WHERE {?S ?P ?X}
```

```
SELECT X WHERE {?S rdf:Type ?X}
```

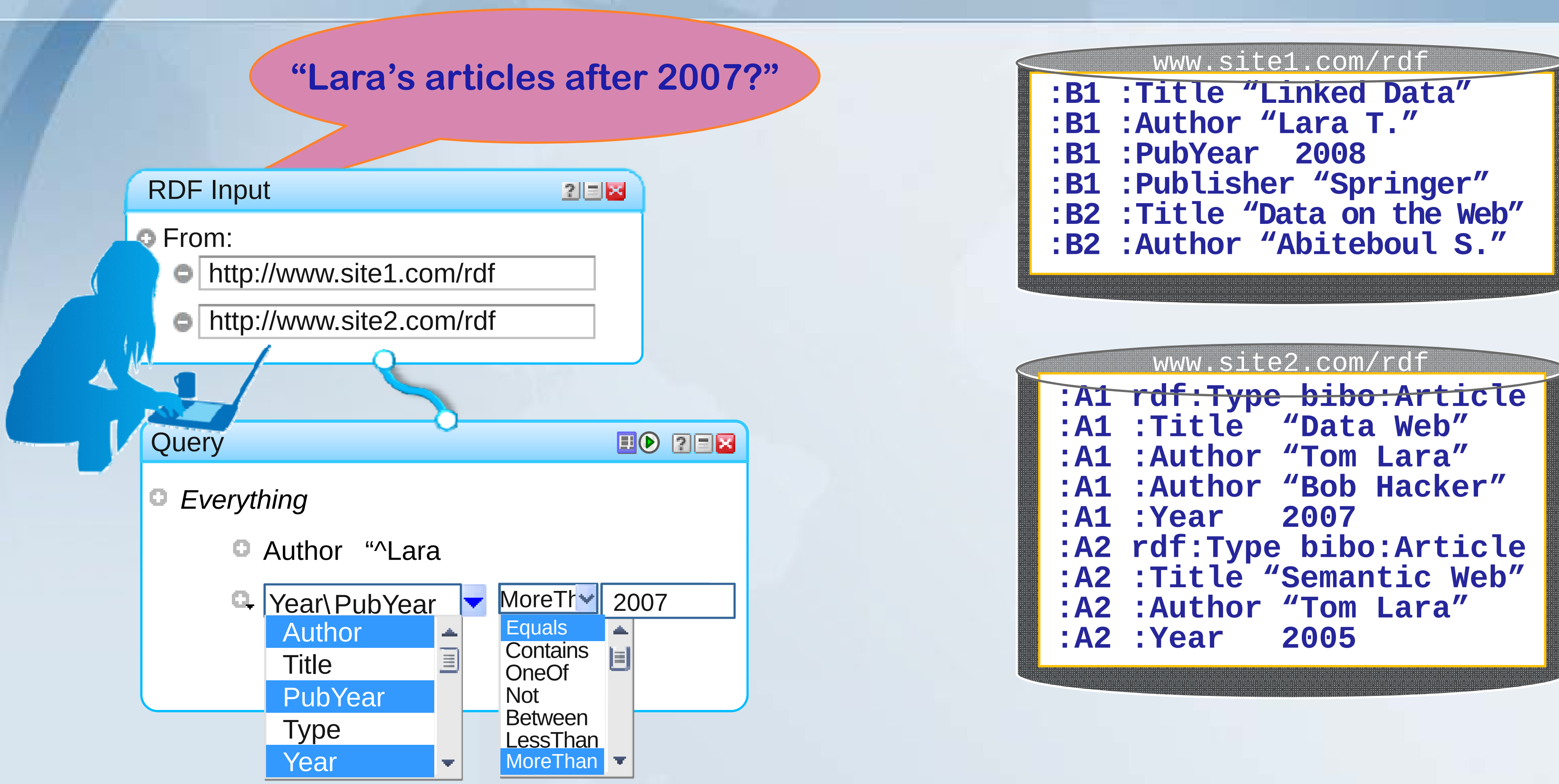
Example 1



Background query

```
SELECT P WHERE {?Everything ?P ?O}
```

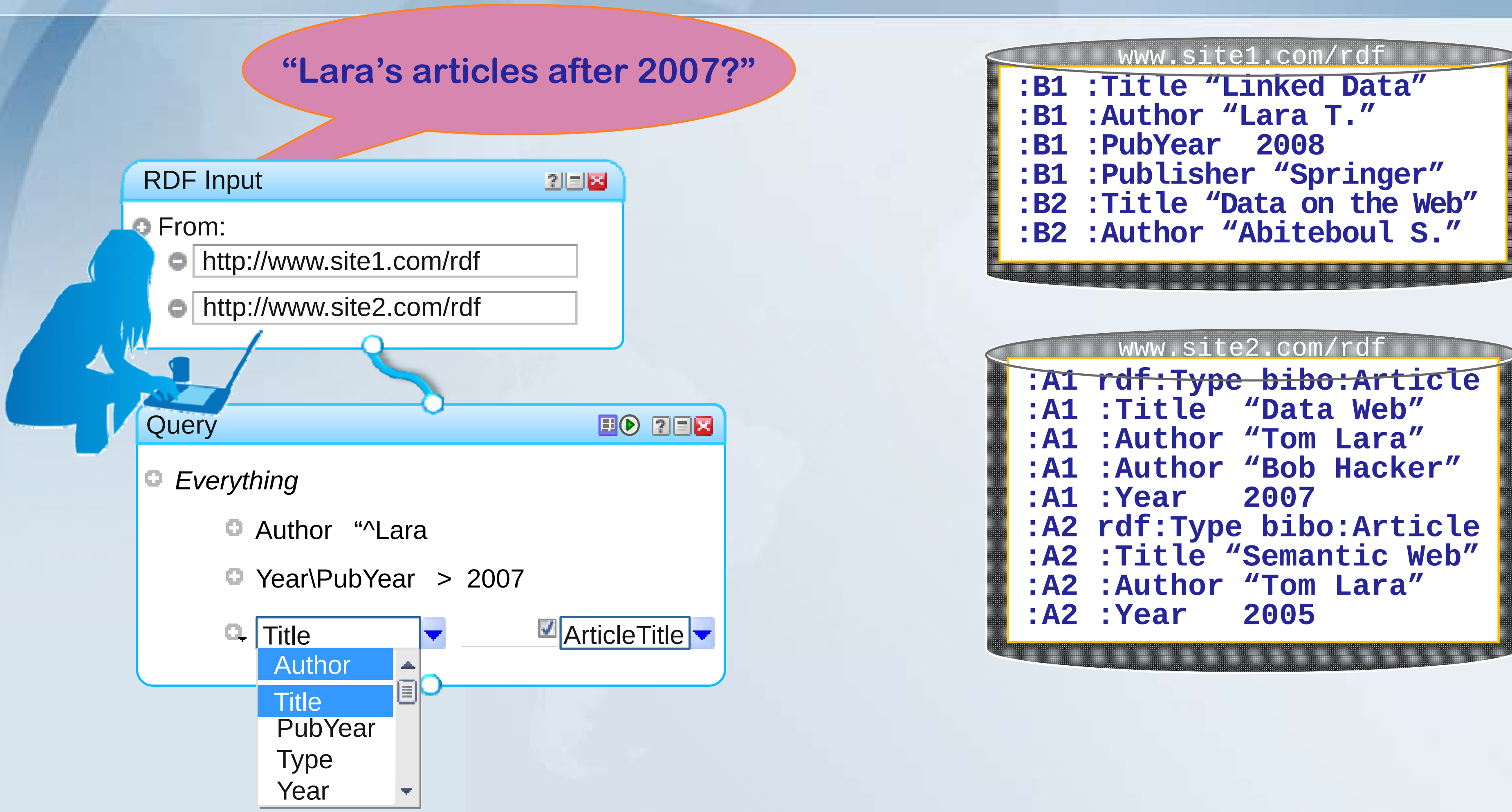
Example 1



Background query

```
SELECT P WHERE {?Everything ?P ?O}
```

Example 1

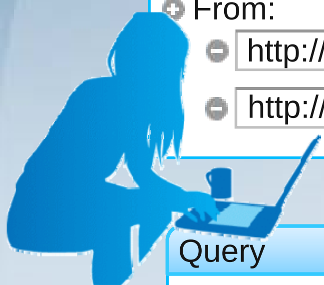


Background query

```
SELECT P WHERE {?Everything ?P ?O}
```

Example 1

"Lara's articles after 2007?"



RDF Input

From:
- http://www.site1.com/rdf
- http://www.site2.com/rdf

Query

Everything
- Author "^Lara
- Year\PubYear > 2007
- Title ☑*ArticleTitle*

www.site1.com/rdf

```
:B1 :Title "Linked Data"
:B1 :Author "Lara T."
:B1 :PubYear  2008
:B1 :Publisher "Springer"
:B2 :Title "Data on the Web"
:B2 :Author "Abiteboul S."
```

www.site2.com/rdf

```
:A1 rdf:Type bibo:Article
:A1 :Title  "Data Web"
:A1 :Author "Tom Lara"
:A1 :Author "Bob Hacker"
:A1 :Year   2007
:A2 rdf:Type bibo:Article
:A2 :Title "Semantic Web"
:A2 :Author "Tom Lara"
:A2 :Year   2005
```

```
PREFIX S1: <http://site1.com/rdf>
PREFIX S2: <http://site1.com/rdf>
SELECT ?ArticleTitle
FROM <http://site1.com/rdf>
FROM <http://site2.com/rdf>
WHERE {
 {?X S1:Author ?X1} UNION {?X S2:Author ?X1}
 {?X S1:PubYear ?X2} UNION {?X S2:Year ?X2}
 {{?X S1:Title ?ArticleTitle} UNION {?X S2:Title
?ArticleTitle}}
 FILTER regex(?X1, "^Hacker")
 FILTER (?X2 > 2000)}
```

Example 2

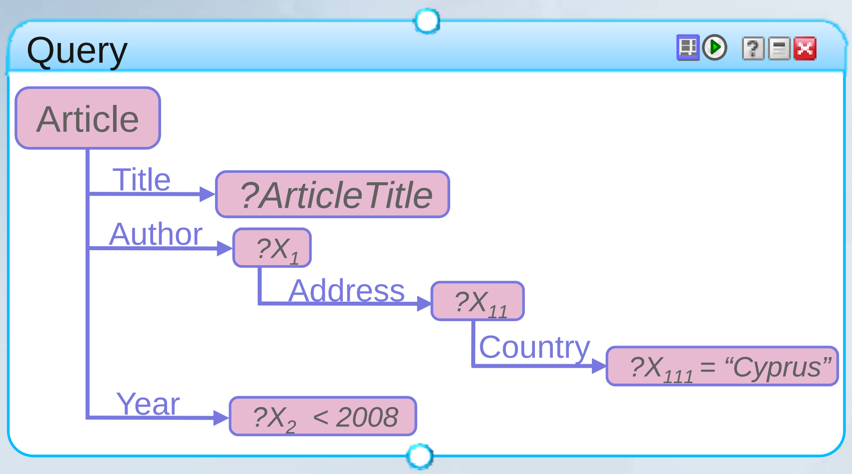
“Recent articles from Cyprus?”

Query

- *Article*
  - Title ☑*ArticleTitle*
  - Author
    - Address
      - Country “Cyprus”
  - Year > 2008

→ Retrieve every Article that: has a title, written by author, who has address, this address has a country called Cyprus, and the article is published after 2008.

The Intuition of MashQL



A query is a tree

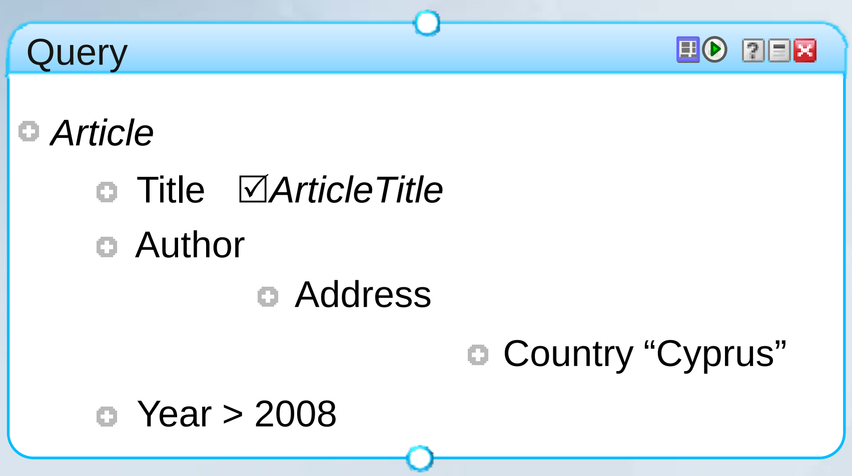
- The root is called the query *subject*.
- Each branch is a *restriction*.
- Branches can be expanded, (information path)
- Object value filters

*Def. A **Query** Q with a subject S, denoted by Q(S), is a set of restrictions on S. Q(S) = R_1 AND … AND R_n.*

*Dif. A **Subject** S $\in (I \cup V)$, where I is an identifier and V is a variable.*

*Dif. A **Restriction** R = <R_x , P, O_f>, where R_x is an optional restriction prefix that can be (maybe | without), P is a predicate (P $\in I \cup V$), and O_f is an object filter.*

The Intuition of MashQL



An Object filter is one of :

- Equals
- Contains
- MoreThan
- LessThan
- Between
- one of
- Not(f)
- Information Path (sub query)

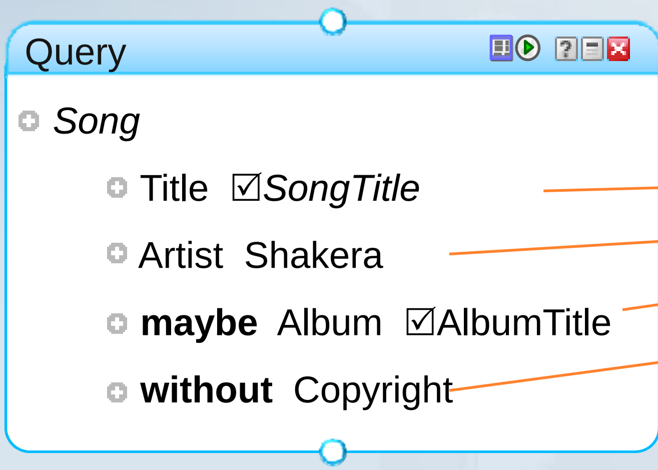
*Def. An **object filter** O_f = <O, f>, where O is an object and f is a filtering function one of :*

O_f = <O>, where O is an object, $O \in V \cup I$.
O_f = <O, Equals(X, T, L_t)>, where X can be a variable or a constant, T is a datatype, and L_t is a language tag.
O_f = <O, Contains(X, T, L_t)>, where O is an object variable, X is a regex literal, T is a data type, and L_t is a language.
O_f = <O, MoreThan(X, T)>, where O is an object variable, X is a variable or a constant, T is a datatype.
O_f = <O, LessThan(X, T)>, where O is an object variable, X is a variable or a constant, T is a datatype identifier.
O_f = <O, Between(X, Y, T)>, where X and Y are variables or constants, T is a datatype identifier.
O_f = <O, OneOf(V)>, where O is an object variable, and V is a set of values $\{v_1, \ldots, v_n\}$, v_i is a variable or constant.
O_f = <O, Not(f)>, where f is one of the functions defined above.
O_f = <O, Q_i(O)>, where O is an object ($O \in V \cup I$), and Q_i(O) is a sub-query with O being the query subject.

More MashQL Constructs

- Resection Operators {Required, Maybe, or Without}

  All restriction are required (i.e. AND), unless they are prefixed with "maybe" or "without"




```
SELECT ?SongTitle, AlbumTitle
WHERE {
 {?X :Title ?SongTitle.
  ?X :Artist :Shakera
  Optional{?X :Album ?AlbumTitle}
  Optional{?X :Copyrihgt ?X2}
  Filter FILTER (!Bound(?X2)).
```

More MashQL Constructs

➤ Union operator (denoted as "\") between

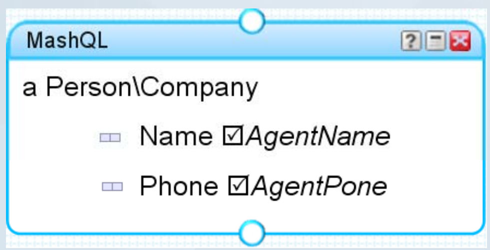
Objects


```
SELECT ?Person
WHERE {
  ?Person :WorkFor :Google
   UNION
  ?Person WorkFor :Yahoo}
```

Predicates

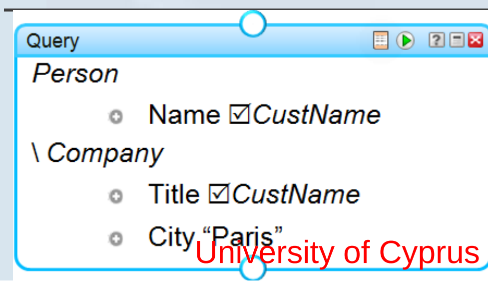

```
SELECT ?FName
WHERE {
 ?Person :Surname ?FName
  UNION
 ?Person :Firstname ?FName}
```

Subjects




```
SELECT ?AgentName, ?AgentPhone
WHERE {
 {?Person rdf:type :Person.
  ?Person :Name ?AgentName.
  ?Person :Phone ?AgentPhone}
UNION
 {?Company rdf:type :Company.
  ?Company :Name ?AgentName.
  ?Company :Phone ?AgentPhone}}
```

Queries

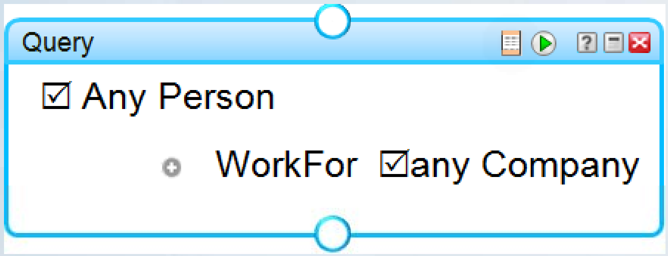



```
SELECT ?CustName,
WHERE {
   ?Person :Name ?CustName.
  UNION
   {?Company :Title ?CustName.
    ?Company :City ?X1.
    FILTER regex(?X1, "Paris")}}
```

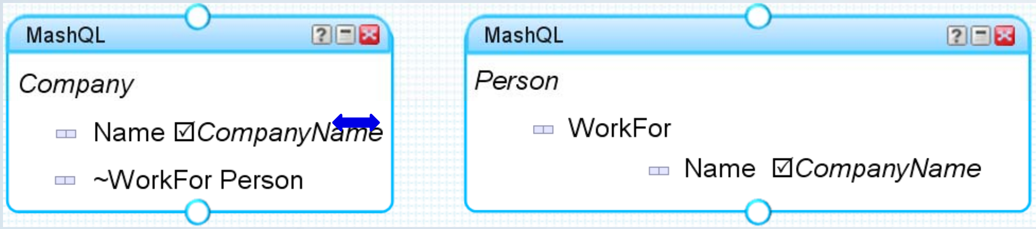
More MashQL Constructs

And several other constructs, including:

- Types



- and Reverse Predicates



- Datatypes and Language Tags
- ….

Formal Syntax and Semantics

Def.1 (Dataset): A dataset D is a set of triples, each triple t is formed as <S, P, O>, where S ∈ I, P ∈ I, and O ∈ I ∪ L.
Def.2 (Typed Literals): Every object literal must have a datatype D: If O ∈ L then O ∈ D.
Def.3 (Language Tags): *An object literal (O ∈ L) may have a language tag* L_t.
Def. 4 (Query): *A Query Q with a subject S, denoted by Q(S), is a set of restrictions on S. Q(S) =* R_1 *AND … AND* R_n.
Def. 5 (Subject): *A subject S ∈ (I ∪ V), where I is an identifier and V is a variable.*
Def. 6 (Restriction): *A restriction R = <*R_x*, P,* O_f*>, where* R_x *is an optional restriction prefix that can be (maybe | without), P is a predicate (P ∈ I ∪ V), and* O_f *is an object.*
Def.7 (Object Filter): *An object filter* O_f *= <O, f>, where O is an object and f is a filtering function. An object filter can have one of the following nine forms:*

1. O_f *= <O>, where O is an object, O ∈ V ∪ I.* This is the simplest object filter, i.e., it does not add any restriction on the object value of the retrieved triples.
2. O_f *= <O, Equals(X, T,* L_t*)>, where X can be a variable or a constant, T is a datatype, and* L_t *is a language tag.* This filter restricts the retrieved results, such that, the object value *O* should be equal to *X*, with datatype *T*, and with language L_t.
3. $_{Of}$ *= <O, Contains(X, T,* $_{Lt}$*)>, where O is an object variable, X is a regex literal, T is a data type, and* $_{Lt}$ *is a language.* This filter restricts the retrieved results, such that, the object value *O* should be equal to regex(*X*), with datatype *T*, and with language $_{Lt}$. A regex literal is a literal that contains a regular expression matching pattern.
4. O_f *= <O, MoreThan(X, T)>, where O is an object variable, X is a variable or a constant, T is a datatype.* This filter restricts the retrieved results, such that, the object value *O* should be more than *X* and with datatype *T*.
5. O_f *= <O, LessThan(X, T)>, where O is an object variable, X is a variable or a constant, T is a datatype identifier.* This filter restricts the retrieved results, such that, the object value *O* should be less than *X* and with datatype *T* (see rule-9).
6. O_f *= <O, Between(X, Y, T)>, where X and Y are variables or constants, T is a datatype identifier.* This filter restricts the retrieved results, such that, the object value *O* should be more than or equals *X*, less than or equals *Y*, and with datatype *T.*
7. O_f *= <O, OneOf(V)>, where O is an object variable, and V is a set of values {*v_1*, ... ,* v_n*},* v_i *is a variable or constant.* This filter restricts the retrieved results, such that, the object value *O* should be equal to one of the values in *V*.
8. O_f *= <O, Not(f)>, where f is one of the functions defined above.* This filter extends all of the above functions with simple negation. The filter is same as the *Equals* filter but with negation, i.e., *Not Equal*.
9. O_f *= <O,* Q_i*(O)>, where O is an object (O ∈ V ∪ I), and* Q_i*(O) is a sub-query with O being the query subject.* The restrictions defined in the sub-query Q_i*(O)* should be satisfied as well. Notice that this definition is recursive; however, this does not mean the query itself is recursive.

Def.8 (Types): *A subject (S ∈ I) or an object (O ∈ I) can be prefixed with "a" or "an" to mean the instances of this subject/object type, instead of the subject/object itself*.
Def.9 (Union): A union can be declared between objects, predicates, subjects and/or queries, in the following forms:

1. O_n = <O_1\\$O_2$\ . . . \\$O_n$>, to indicate unions between objects, where O_i ∈ I.
2. P_n = <P_1\\$P_2$\ . . . \\$P_n$>, to indicate unions between predicates, where P_i ∈ I.
3. S_n = <S_1\\$S_2$\ . . . \\$S_n$>, to indicate unions between subjects, where S_i ∈ I.
4. Q_n *= <*Q_1**Q_2*\ . . . *Q_n*>, to indicate unions between queries*.

Def.10 (Reverse): *<~P> indicates the reverse of the predicate P. Let* R_1 *be a restriction on S such that <S P O>, and* R_2 *be <O ~P S>,* R_1 *and* R_2 *have the same meaning.*

Query Formulation Algorithm

Formalization of the background queries

Select Subject S

(1) $S \in S_T : \pi_O(\sigma_{P=':Type'}(D))$

(1') `O1:{(?S1 <:Type> ?O1)}`

(2) $S \in S_I : \pi_S(D) \cup \pi_O(\sigma_{O\in}(D))$

(2') `S1:{(?S1 ?P1 ?O1)} UNION O1:{(?S1 ?P1?O1). Filter isURI(?O1)}`

(3) $S \in V$

Select a property P

(4) $(S \in S_T) \rightarrow P \in \pi_{P2}(\sigma_{P1=':Type' \wedge O1=Subject}(D) \rtimes_{S1=S2} \sigma(D))$

(4') `P2:{(?S1 <:Type> <S>)(?S2 ?P2 ?O2)}`

(5) $(S \in S_I) \rightarrow P \in \pi_P(\sigma_{S=Subject}(D))$

(5') `P1:{(<S> ?P1 ?O1)}`

(6) $(S \in V) \rightarrow P \in \pi_P(\sigma(D))$

(6') `P1:{(?S1 ?P1 ?O1)}`

(7) $P \in V$

Add a filter on P

(8) $(S \in S_I) \wedge (P \in V) \rightarrow O \in \pi_{O1}(\sigma_{S1=S \wedge O1\in}(D))$

(8') `O1:{(<S> ?P1 ?O1) Filter isURI(?O1)}`

(9) $(S \in S_I) \wedge (P \notin V) \rightarrow O \in \pi_{O1}(\sigma_{S1=S \wedge P1=P \wedge O1\in}(D))$

(9') `O1:{(<S> <P> ?O1) Filter isURI(?O1)}`

(10) $(S \in S_T) \wedge (P \in V) \rightarrow O \in \pi_{O2}(\sigma_{P1=':Type' \wedge O1=S}(D) \rtimes_{S1=S2} \sigma(D))$

(10') `O1:{(?S1 <:Type> <S>)(?S1 ?P2 ?O2)}`

(11) $(S \in S_T) \wedge (P \notin V) \rightarrow O \in \pi_{O2}(\sigma_{P1='rdf:Type' \wedge O1=S}(D) \rtimes_{S1=S2} \sigma_{P2=P}(D))$

(11') `O:{(?S <rdf:Type> <S>)(?S <P> ?O)}`

(12) $(S \in V) \wedge (P \in V) \rightarrow O \in \pi_O(\sigma(D))$

(12') `O1:{(?S1 ?P1 ?O1)}`

(13) $(S \in V) \wedge (P \notin V) \rightarrow O \in \pi_O(\sigma_{P=P}(D))$

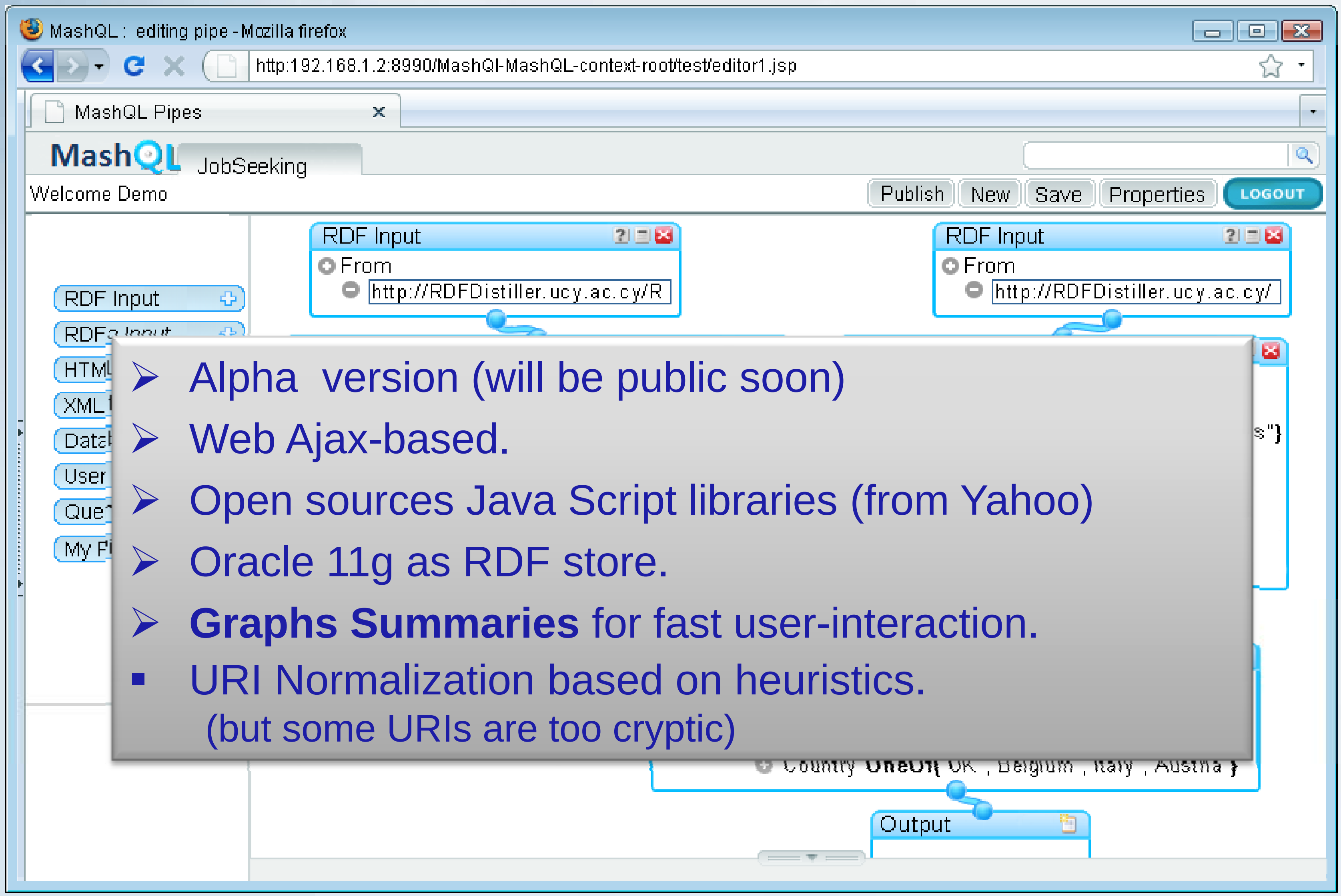
(13') `O1:{(?S1 <P> ?O1)}`

MashQL-SPARQL Mapping Rules

Rule-1: The symbol ☑ before a variable means that it will be returned in the results; i.e., included in the SELECT part of in SPARQL. If the output of the query is input to another, use "CONSTRUCT *".
Rule-2: In any of the following rules, if a subject, predicate, or object is italicized: it is seen as a SPARQL variable, i.e. prefixed with "?".
Rule-3: If S is a subject and R = < , P, Of>, the mapping is: {S P O}.
Rule-4: If S is a subject and R = <maybe, P, Of>, the mapping is: {OPTIONAL{S P O}}.
Rule-5: If S is a subject and R = < without, P, Of>, the mapping is: {S P O. FILTER (!bound(?O))}.
Rule 6. If Of = <O, Equals(X, T, Lt)>:
  Append the mapping with: FILTER(?O = X)
  If T ≠ Null: Append the mapping with: FILTER(datatype(?O)=T)
  If Lt ≠ Null: Append the mapping with: FILTER(lang(?O) = Lt)
Rule 7. If Of = Cor
  Append the ma
  If T ≠ Null: App
  If Lt ≠ Null: App
Rule 8. If Of = <O,
  Append the ma
  If T ≠ Null: App
Rule 9. If Of = <O,
  Append the mapping with: FILTER(?O < X)
  If T ≠ Null: Append the mapping with: FILTER(datatype(?O=T)
Rule 10. If Of = <O, Between(X, Y, T)>:
  Append the mapping with: FILTER(?O >=X)&& FILTER(?O<=Y)
  If T ≠ Null: Append the mapping with: FILTER(datatype(?O)=T)
Rule 11. If Of = <O, OneOf (V)>:
  Append the mapping with: {FILTER(?O = V1)|| . . . || FILTER(?O = Vn)}
  If Vi is a regex-ed literal, the ith filter above should be replaced with: FILTER Regex(?O, Vi)
Rule 12. If Of = <O, Not(f)>: The f filter will be generated as above, but with a negation.
Rule 13. If Of = <O, Qi(O)>: Repeat all mapping rules to generate Qi(O).
Rule 14. If a subject S is prefixed with "a" or "an": Append the mapping with: {?S rdf:type :S}
Rule 15. If an object O is prefixed with "a" or "an": Append the mapping with: {?O rdf:type :O}
Rule 16. Given On , If n >1 and Oi ∈ I : The mapping in rules 3-4 will be:{{S P :O1} UNION . . . UNION {S P :On}}
Rule 17. Given Pn , If n >1 and Pi ∈ I : The mapping in rules 3-4 will be: {{S :P1 O} UNION . . . UNION {S :Pn O}}
Rule 18. Given Sn , If n >1 and Si ∈ I : Regenerate the query n times, each time with Si as a root, and with a UNION between the queries.
Rule 19. Given Qn , If n >1 : Add UNION between the n queries.
Rule 20. If S is a subject and R = <~P, O>, the mapping is: {O P S}.

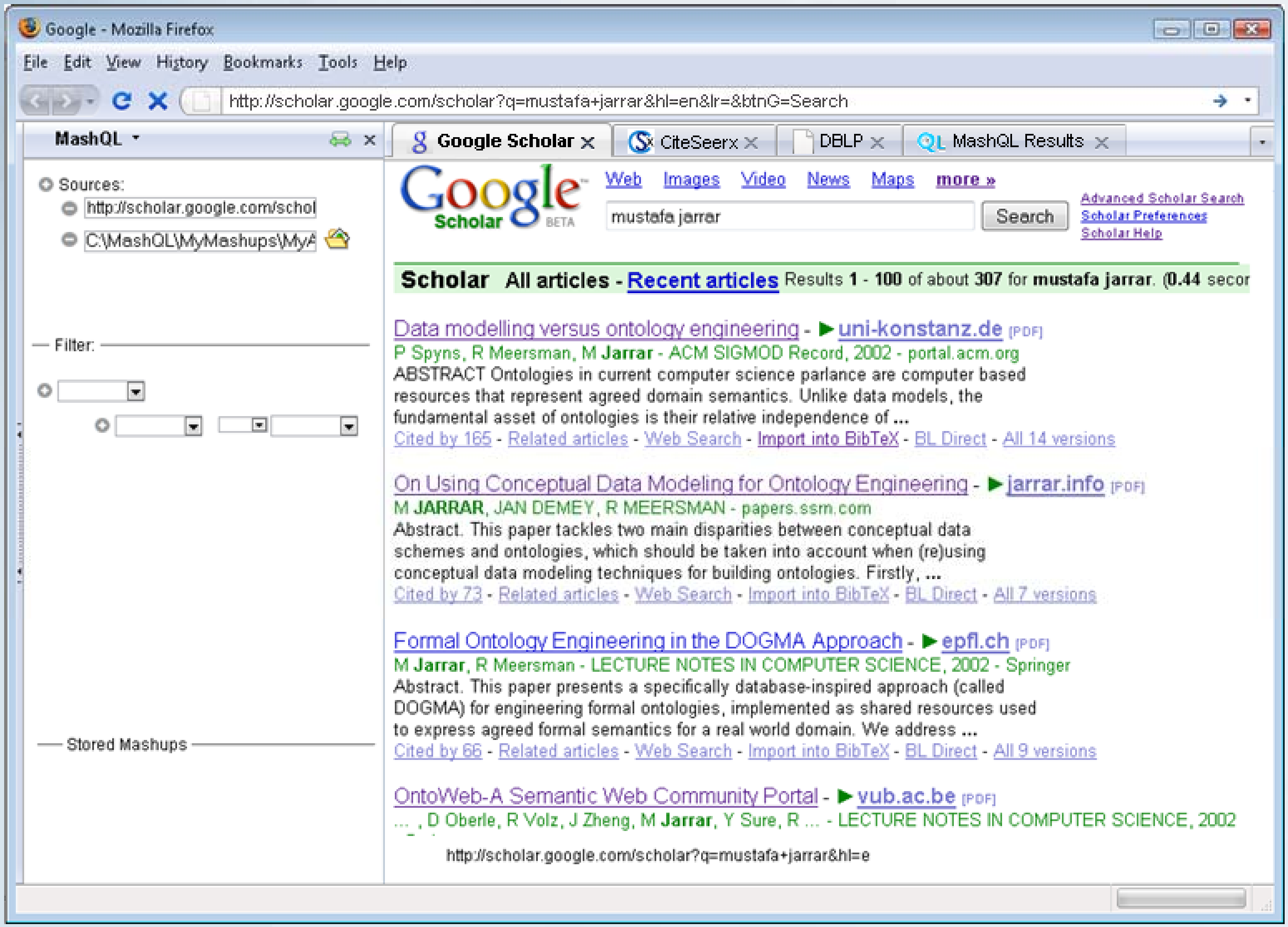
➔Also mapped into Oracle's SPARQL

MashQL Editor



- Alpha version (will be public soon)
- Web Ajax-based.
- Open sources Java Script libraries (from Yahoo)
- Oracle 11g as RDF store.
- **Graphs Summaries** for fast user-interaction.
- URI Normalization based on heuristics.
  (but some URIs are too cryptic)

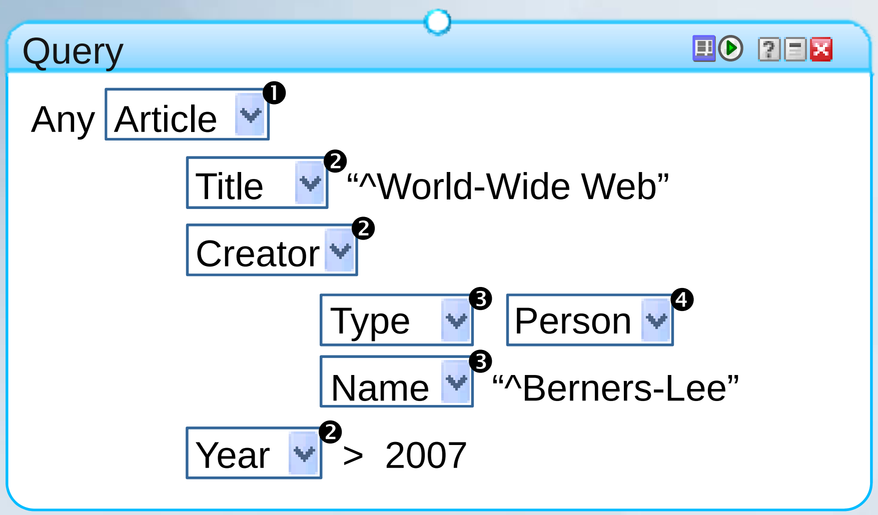
MashQL Firefox Add-On (Light-mashups @ your browser)



Evaluation (DBLP, Experiment 1)

(User Interaction Response Time)

How long it takes to generate the next list?



❶ `O:(?S Type ?O)`

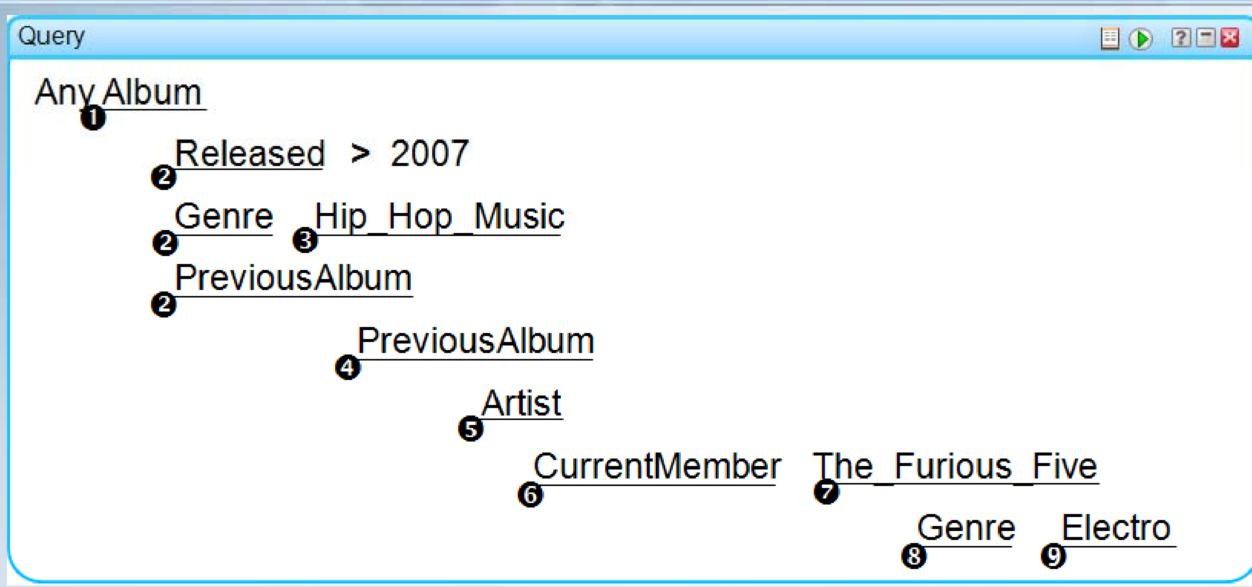
❷ `P:(?S Type Article)(?S ?P ?O1)`

❸ `P:(?S Type Article)`
`(?S Creator ?O1) (?O1 ?P ?O2)`

❹ `O:(?S Type Article)`
`(?S Creator ?O1)(?O1 Type ?O)`

Query	DBLP (9M triples)		DBLP (4M triples)		DBLP (2M triples)	
	GS	Oracle	GS	Oracle	GS	Oracle
Q_1	0.003	0.005	0.003	0.004	0.003	0.003
Q_2	0.001	0.136	0.001	0.148	0.001	0.108
Q_3	0.001	0.187	0.001	0.846	0.001	0.671
Q_4	0.001	1.208	0.001	0.835	0.001	0.650

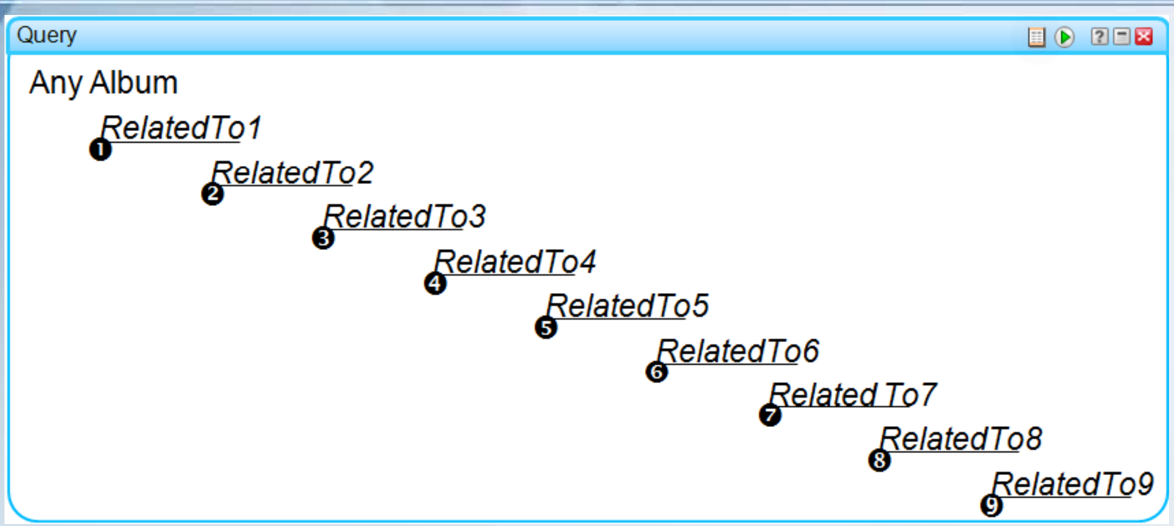
Evaluation (DBPedia, Experiment 2)



❹ P:(?S :Type :Album)
(?S :PreviousAlbum ?01)
(?01 ?P ? 02)

Query	DBPedia (32 M)	DBPedia (16 M)	DBPedia (8 M)	DBPedia (4 M)
Q_1	0.003	0.003	0.003	0.003
Q_2	0.002	0.002	0.002	0.002
Q_3	0.005	0.004	0.003	0.003
Q_4	0.005	0.004	0.004	0.004
Q_5	0.005	0.004	0.004	0.004
Q_6	0.005	0.005	0.005	0.005
Q_7	0.007	0.007	0.007	0.007
Q_8	0.005	0.005	0.005	0.005
Q_9	0.007	0.007	0.007	0.006

Evaluation (DBPedia, Experiment 3)



❸ P3:(?S Type ?Album)
(?S ?RelatedTo1 ?O1)
(?O1 ?RelatedTO2 ?O2)
(?O2 ?P3 ?O4)

Query	(B32) 32 M		DBPedia (16 M)	DBPedia (8 M)	DBPedia (4 M)
	GS	Oracle			
Q_1	0.001	0.027	0.001	0.001	0.001
Q_2	0.026	17.450	0.010	0.006	0.005
Q_3	0.048	49.656	0.010	0.011	0.010
Q_4	0.087	5348.7	0.022	0.017	0.011
Q_5	0.135	-	0.036	0.032	0.011
Q_6	0.185	-	0.055	0.047	0.016
Q_7	0.234	-	0.076	0.061	0.023
Q_8	0.265	-	0.098	0.077	0.027
Q_9	0.280	-	0.110	0.092	0.034

Conclusions

- “Query and mash up the Data Web as simple as filtering up Web Feeds” is a query formulation problem.

- End-users can navigate, query, and mash up unknown graphs.

  without knowing the schema. Data is schema-free. Multiple sources.

- MashQL is expressive as SPARQL.

  Except NAMED GRAPH.

- MashQL is not merely a SPARQL interface, or limited RDF.

  It has its own path-pattern intuition (can be similarly used for XML and DB).

Future Work

- Reasoning, Keyword Search, Aggregation Functions, etc.
- Results Presentation (Should be tacked fundamentally).
- Firefox add-on Mashup/query editor.
- RDF summaries for SPARQL optimization.
- ..

Thank You

Dr. Mustafa Jarrar

mjarrar@cs.ucy.ac.cy

HPCLab, University of Cyprus