Enabling Completeness-aware Querying in SPARQL

Luis Galárraga, Katja Hose, Simon Razniewski

May 14th, 2017
WebDB, Chicago
Outline

• Completeness in RDF knowledge bases
• Completeness oracles
• Our vision
  – Representations for completeness oracles
  – Reasoning with completeness oracles
  – Enabling completeness in SPARQL
• Summary & conclusions
Outline

• Completeness in RDF knowledge bases
• Completeness oracles
• Our vision
  – Representations for completeness oracles
  – Reasoning with completeness oracles
  – Enabling completeness in SPARQL
• Summary & conclusions
Outline

• Completeness in RDF knowledge bases
• Completeness oracles
• Our vision
  – Representations for completeness oracles
  – Reasoning with completeness oracles
  – Enabling completeness in SPARQL
• Summary & conclusions
RDF Knowledge Bases (KBs)

Collection of structured knowledge

- Romance
  - Français
    - officialLanguage
  - Italiano
    - officialLanguage

- Switzerland
  - citizenOf

- Leonhard Euler
Plenty of KBs out there!
Plenty of KBs out there!
KBs in action

<table>
<thead>
<tr>
<th>Official Languages of Switzerland</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Switzerland › Official Languages</td>
<td></td>
</tr>
<tr>
<td>French</td>
<td>Romansh</td>
</tr>
<tr>
<td>German</td>
<td>Italian</td>
</tr>
</tbody>
</table>
Outline

- Completeness in RDF knowledge bases
- Completeness oracles
- Our vision
  - Representations for completeness oracles
  - Reasoning with completeness oracles
  - Enabling completeness in SPARQL
- Summary & conclusions
Completeness in RDF KBs

• KBs are highly incomplete
  – 1% of people have a citizenship in YAGO
Completeness in RDF KBs

• KBs are highly incomplete
  – 1% of people have a citizenship in YAGO
• We do not know where the incompleteness lies
Completeness in RDF KBs

• KBs are highly incomplete
  – 1% of people have a citizenship in YAGO

• We do not know where the incompleteness lies
  – A single person in the KB could be actually single or the KB may be incomplete
Completeness in RDF KBs

• KBs are highly incomplete
  – 1% of people have a citizenship in YAGO

• We do not know where the incompleteness lies
  – A single person in the KB could be actually single or the KB may be incomplete

• Problems for data producers and consumers
Completeness in RDF KBs

• KBs are highly incomplete
  – 1% of people have a citizenship in YAGO

• We do not know where the incompleteness lies
  – A single person in the KB could be actually single or the KB may be incomplete

• Problems for data producers and consumers
  – Consumers: no completeness guarantees for queries.
  – Producers: which parts of the KB need to be populated?
Completeness

- Defined with respect to a query $q$ via a complete hypothetical KB $K^*$. 

Completeness

• Defined with respect to a query $q$ via a complete hypothetical KB $K^*$.
  
  – A query $q$ is complete in $K$, iff $q(K^*) \subseteq q(K)$.
Completeness

- Defined with respect to a query $q$ via a complete hypothetical KB $K^*$.
  - A query $q$ is complete in $K$, iff $q(K^*) \subseteq q(K)$.

```
SELECT ?x WHERE { Switzerland officialLang ?x }
```
Completeness

- Defined with respect to a query $q$ via a complete hypothetical KB $K^*$.
  - A query $q$ is complete in $K$, iff $q(K^*) \subseteq q(K)$.

SELECT ?x WHERE { Switzerland officialLang ?x }
Completeness

- Defined with respect to a query \( q \) via a complete hypothetical KB \( K^* \).
- A query \( q \) is complete in \( K \), iff \( q(K^*) \subseteq q(K) \).

SELECT ?x WHERE { Switzerland officialLang ?x }
Completeness in RDF data

- Wikidata provides no value annotations
Completeness in RDF data

- Wikidata provides *no value annotations*

```
SELECT ?x WHERE { USA officialLang ?x }
```
Completeness in RDF data

- Wikidata provides no value annotations

SELECT ?x WHERE { USA officialLang ?x }
Completeness in RDF data

- Wikidata provides *no value annotations*
  
  ```sql
  SELECT ?x WHERE { USA officialLang ?x }
  ```

  ![Official Language](image)

  *Complete query*

- Not applicable if we know some official language
Completeness in RDF data

- Wikidata provides *no value annotations*

  ```sql
  SELECT ?x WHERE { USA officialLang ?x }
  ```

  ![Diagram of official language in RDF]

  [Complete query]

- Not applicable if we know some official language

  ![Diagram of official language in RDF for Switzerland]
Outline

- Completeness in RDF knowledge bases
- Completeness oracles
- Our vision
  - Representations for completeness oracles
  - Reasoning with completeness oracles
  - Enabling completeness in SPARQL
- Summary & conclusions
Completeness oracle

- Boolean function $\omega(q, K)$ that guesses the completeness of a query $q$ in a KB $K$. 
SR completeness oracle

- Function $\alpha$ that guesses the completeness of queries of the form [Galárraga et. al, 2017]:

  \[
  \text{SELECT } ?x \text{ WHERE } \{ \text{ subject relation } ?x \} 
  \]
SR completeness oracle

- Function $\omega$ that guesses the completeness of queries of the form [Galárraga et. al, 2017]:

  $$\text{SELECT } ?x \text{ WHERE } \{ \text{subject relation } ?x \}$$

- We use the notation $\omega(subject, relation)$
SR completeness oracle

- Function \( \omega \) that guesses the completeness of queries of the form [Galárraga et. al, 2017]:

  \[
  \text{SELECT } ?x \text{ WHERE } \{ \text{subject relation } ?x \} 
  \]

- We use the notation \( \omega(subject, relation) \)

- \( \omega = pca(s, r) = \text{partial completeness assumption} \)
SR completeness oracle

• Function $\omega$ that guesses the completeness of queries of the form [Galárraga et. al, 2017]:

$$\text{SELECT } ?x \text{ WHERE } \{ \text{subject} \text{ relation } ?x \}$$

• We use the notation $\omega(subject, relation)$

• $\omega = pca(s, r) =$ partial completeness assumption
  – Query is complete in KB if at least one answer is known
Evaluating SR oracles

\[ \omega = pca(s, r) = \text{partial completeness assumption} \]
Evaluating SR oracles

ω = pca(s, r) = partial completeness assumption

Gold standard:
Complete instances in the domain of official Language

PCA oracle

Français
Italiano
Dansk
Evaluating SR oracles

$\omega = \text{american-country-oracle}(s, r)$

**Gold standard:** Complete instances in the domain of official Language
Evaluating SR oracles

**PCA oracle**
- Precision = 3/5
- Recall = 3/4

**American country oracle**
- Precision = 1/2
- Recall = 1/4

**Gold standard:**
Complete instances in the domain of officialLanguage
SR completeness oracles

- Closed World Assumption: $\text{cwa}(s, r) = \text{true}$
- PCA: $\text{pca}(s, r) = \exists o : r(s, o)$
- Cardinality: $\text{card}(s, r) = \#(o : r(s, o)) \geq k$
- Popular entities: $\text{popularity}_{\text{pop}}(s, r) = \text{pop}(s)$
- No-chg over time: $\text{nochange}_{\text{chg}}(s, r) = \sim \text{chg}(s, r)$
- Star: $\text{star}_{r_1,\ldots,r_n}(s, r) = \forall i \in \{1,\ldots,n\} : \exists o : r_i(s, o)$
- Class: $\text{class}_c(s, r) = \text{type}(s, c)$
- Rule mining oracle
Rule mining SR oracle

• Based on completeness rules

\[
\text{notype}(x, \text{Adult}), \text{type}(x, \text{Person}) \Rightarrow \text{complete}(x, \text{hasChild})
\]

\[
\text{dateOfDeath}(x, y), \text{lessThan}_1(x, \text{placeOfDeath}) \Rightarrow \text{incomplete}(x, \text{placeOfDeath})
\]
Rule mining SR oracle

- Based on completeness rules

\[
\text{notype}(x, \text{Adult}), \text{type}(x, \text{Person}) \Rightarrow \text{complete}(x, \text{hasChild})
\]

\[
\text{dateOfDeath}(x, y), \text{lessThan}_{1}(x, \text{placeOfDeath}) \Rightarrow \text{incomplete}(x, \text{placeOfDeath})
\]

- Learned using the AMIE [Galárraga et. al, 2013] rule mining system
  - On gold standard built via crowdsourcing
Rule mining SR oracle

• Based on completeness rules

\[
\text{notype}(x, \text{Adult}), \text{type}(x, \text{Person}) \Rightarrow \text{complete}(x, \text{hasChild}) \\
\text{dateOfDeath}(x, y), \text{lessThan}_1(x, \text{placeOfDeath}) \Rightarrow \text{incomplete}(x, \text{placeOfDeath})
\]

• Learned using the AMIE [Galárraga et. al, 2013] rule mining system
  - On gold standard built via crowdsourcing
  - 100% F1-measure for functional relations, quite good for relations \textit{hasChild, graduatedFrom}
Outline

- Completeness in RDF knowledge bases
- Completeness oracles
- Our vision
  - Representations for completeness oracles
  - Reasoning with completeness oracles
  - Enabling completeness in SPARQL
- Summary & conclusions
Representing completeness oracles

• Extensional approach [Darari, et. Al, 2013]
  - An oracle is a collection of completeness statements about queries
Representing completeness oracles

• Extensional approach [Darari, et. Al, 2013]
  – An oracle is a collection of completeness statements about queries

```
SELECT DISTINCT ?y WHERE { ?x hasOfficialLang ?y } is complete in the KB
```
Representing completeness oracles

- Extensional approach [Darari, et. Al, 2013]
  - An oracle is a collection of completeness statements about queries

SELECT DISTINCT ?y WHERE { ?x hasOfficialLang ?y } is complete in the KB
Representing completeness oracles

- Extensional approach [Darari, et. Al, 2013]
  - A call to the oracle asks for the existence of the query in the graph

\[
\text{SELECT DISTINCT } ?y \text{ WHERE } \{ ?x \text{ hasOfficialLang } ?y \} \text{ is complete in the KB}
\]

\[
\text{true} \text{ hasPattern pattern} \rightarrow \text{pattern} \text{ hasOfficialLang } ?x \rightarrow ?x \text{ subject} \rightarrow ?x \text{ hasProjectionVariable} \rightarrow \text{pattern} \text{ object} \rightarrow \text{pattern} \text{ hasPattern} \rightarrow \text{true}
\]

\[
?y \text{ hasOfficialLang} \rightarrow ?y \text{ hasPattern} \rightarrow ?y \text{ distinct}
\]
Representing completeness oracles

• Intensional approach
  – The oracle logic is embedded as a lambda function or a link to a program or resource
Representing completeness oracles

- Intensional approach
  - The oracle logic is embedded as a lambda function or a link to a program or resource
Providing completeness guarantees

List of results is complete according to oracle $\omega$ with confidence $X$
Providing completeness guarantees
Providing completeness guarantees

```
SELECT ?country WHERE {
  ?lang family Romance .
}
```
Providing completeness guarantees

SELECT ?country WHERE {
  ?lang family Romance .
}
Outline

- Completeness in RDF knowledge bases
- Completeness oracles
- Our vision
  - Representations for completeness oracles
  - Reasoning with completeness oracles
  - Enabling completeness in SPARQL
D completeness oracles

• Oracle $\omega_d$ for the completeness of queries:

  SELECT DISTINCT ?x WHERE { ?x relation ?y }
  SELECT DISTINCT ?y WHERE { ?x relation ?y }
D completeness oracles

• Oracle $\omega_d$ for the completeness of queries:

  SELECT DISTINCT ?x WHERE { ?x relation ?y }
  SELECT DISTINCT ?y WHERE { ?x relation ?y }

• We use the notation $\omega_d(relation)$ or $\omega_d(relation^{-1})$

  SELECT DISTINCT ?y WHERE { ?x officialLang ?y }
D completeness oracles

- Oracle $\omega_d$ for the completeness of queries:

  SELECT DISTINCT ?x WHERE { ?x relation ?y }
  SELECT DISTINCT ?y WHERE { ?x relation ?y }

- We use the notation $\omega_d(relation)$ or $\omega_d(relation^{-1})$

  SELECT DISTINCT ?y WHERE { ?x officialLang ?y }

- If $\omega_d$ returns true, $\omega_d$ states that the KB knows all languages that are official in some country
Completeness guarantees for arbitrary queries

- Write completeness annotations for every possible type of query
  - It requires a large amount of effort
Completeness guarantees for arbitrary queries

- Write completeness annotations for every possible type of query
  - It requires a large amount of effort
- Reuse existing SR and D oracles
Completeness guarantees for arbitrary queries

- Write completeness annotations for every possible type of query
  - It requires a large amount of effort
- Reuse existing SR and D oracles

```
SELECT ?country WHERE {
  ?lang family Romance .
}
```
Completeness guarantees for arbitrary queries

- Write completeness annotations for every possible type of query
  - It requires a large amount of effort

- Reuse existing SR and D oracles

SELECT ?country WHERE {
  ?lang family Romance .
}

ω’ =
Completeness guarantees for arbitrary queries

- Write completeness annotations for every possible type of query
  - It requires a large amount of effort
- Reuse existing SR and D oracles

\[
\omega' = \omega(\text{Romance, family}^{-1})
\]

```
SELECT ?country WHERE {
  ?lang family Romance .
}
```
Completeness guarantees for arbitrary queries

• Write completeness annotations for every possible type of query
  – It requires a large amount of effort
• Reuse existing SR and D oracles

\[
\omega' = \omega(\text{Romance}, \text{family}^{-1}) \land (\bigwedge_{l:\text{family}(l, \text{Romance})} \omega(l, \text{officialLang}^{-1}))
\]
Completeness guarantees for arbitrary queries

- Write completeness annotations for every possible type of query
  - It requires a large amount of effort
- Reuse existing SR and D oracles

```sql
SELECT ?country WHERE {
  ?country officialLang ?lang
  ?lang family Romance .
}
```

\[ \omega' = \omega(\text{Romance, family}^{-1}) \land (\land_{l: \text{family}(l, \text{Romance})} \omega(l, \text{officialLang}^{-1})) \]

It will generate false negatives
Completeness guarantees for arbitrary queries

- Write completeness annotations for every possible type of query
  - It requires a large amount of effort
- Reuse existing SR and D oracles

SELECT ?country WHERE {
  ?lang family Romance .
}

\( \varphi' = \varphi(\text{Romance, family}^{-1}) \land (\bigwedge_{l:\text{family}(l, \text{Romance})} \varphi(l, \text{officialLang}^{-1})) \)

If the KB misses Ligurian, this term returns false.
Completeness guarantees for arbitrary queries

- Write completeness annotations for every possible type of query
  - It requires a large amount of effort
- Reuse existing SR and D oracles

\[
\omega' = \omega(\text{Romance, family}^{-1}) \wedge (\bigwedge_{l:\text{family}(l, \text{Romance})} \omega(l, \text{officialLang}^{-1}))
\]

\[
\text{SELECT } ?\text{country} \text{ WHERE } \{ \\
?\text{country} \text{ officialLang } ?\text{lang} . \\
?\text{lang} \text{ family Romance} . \\
\}
\]
SELECT ?country WHERE {
    ?country monarch ?monarch .
    ?country locatedIn Europe .
    ?lang family Romance .
}
SELECT ?country WHERE {
    ?country monarch ?monarch .
    ?country locatedIn Europe .
    ?lang family Romance .
}

Automatic oracle composition
Automatic oracle composition

SELECT ?country WHERE {
    ?country monarch ?monarch .
    ?country locatedIn Europe .
    ?lang family Romance .
}
Automatic oracle composition

SELECT ?country WHERE {
  ?country monarch ?monarch .
  ?country locatedIn Europe .
  ?lang family Romance .
}
Automatic oracle composition

\[ \omega' = \omega(\text{Europe, locatedIn}^{-1}) \]

```
SELECT ?country WHERE {
  ?country monarch ?monarch .
  ?country locatedIn Europe .
  ?lang family Romance .
}
```
Automatic oracle composition

\[ \omega' = \omega(\text{Europe, locatedIn}^{-1}) \]

```
SELECT ?country WHERE {
  ?country monarch ?monarch .
  ?country locatedIn Europe .
  ?lang family Romance .
}
```
Automatic oracle composition

\[ \omega' = \omega(\text{Europe, locatedIn}^{-1}) \]

SELECT ?country WHERE {
  ?country monarch ?monarch .
  ?country locatedIn Europe .
  ?lang family Romance .
}
Automatic oracle composition

\[ \omega'' = \omega_d(\text{monarch}) \land \\
( \Lambda_{\text{country}} \omega(\text{country, monarch})) \]

\[ \omega' = \omega(\text{Europe, locatedIn}^{-1}) \]

SELECT ?country WHERE {
    ?country monarch ?monarch .
    ?country locatedIn Europe .
    ?lang family Romance .
}
Automatic oracle composition

\[ \omega'' = \omega_d(\text{monarch}) \land \\
(\land_{\text{country}} \omega(\text{country}, \text{monarch})) \\
\omega' = \omega(\text{Europe}, \text{locatedIn}^{-1}) \]

SELECT ?country WHERE {
  ?country monarch ?monarch .
  ?country locatedIn Europe .
  ?lang family Romance .
}
Automatic oracle composition

\[ \omega'' = \omega_d(\text{monarch}) \land (\land_{\text{country}} \omega(\text{country}, \text{monarch})) \]

\[ \omega' = \omega(\text{Europe}, \text{locatedIn}^{-1}) \]

SELECT ?country WHERE {
  ?country monarch ?monarch .
  ?country locatedIn Europe .
  ?lang family Romance .
}
SELECT ?country WHERE {
    ?country monarch ?monarch .
    ?country locatedIn Europe .
    ?lang family Romance .
}
Automatic oracle composition

\[ \omega'' = \omega_d(\text{monarch}) \land (\land_{\text{country}} \omega(\text{country}, \text{monarch})) \]

\[ \omega' = \omega(\text{Europe}, \text{locatedIn}^{-1}) \]

\[ \omega^* = \omega(\text{Romance}, \text{family}^{-1}) \]

SELECT ?country WHERE {
  ?country monarch ?monarch .
  ?country locatedIn Europe .
  ?lang family Romance .
}
Automatic oracle composition

\( \omega'' = \omega_d(\text{monarch}) \land (\Lambda_{\text{country}} \omega(\text{country}, \text{monarch})) \)

\( \omega' = \omega(\text{Europe}, \text{locatedIn}^{-1}) \)

\( \omega* = \omega(\text{Romance}, \text{family}^{-1}) \)

\( \omega** = \Lambda_{l : \text{family}(l, \text{Romance})} \omega(l, \text{officialLang}^{-1})) \)

SELECT ?country WHERE {
    ?country monarch ?monarch .
    ?country locatedIn Europe .
    ?lang family Romance .
}
Automatic oracle composition

\( \omega'' = \omega_d(\text{monarch}) \land (\Lambda \text{country} \; \omega(\text{country, monarch})) \)

\( \omega' = \omega(\text{Europe, locatedIn}^{-1}) \)

\( \omega^{**} = \Lambda l : \text{family}(l, \text{Romance}) \; \omega(l, \text{officialLang}^{-1})) \)

\( \omega^* = \omega(\text{Romance, family}^{-1}) \)

SELECT ?country WHERE {
  ?country monarch ?monarch .
  ?country locatedIn Europe .
  ?lang family Romance .
}

Completeness veredict = \( \omega^{**} \land \omega^* \land \omega' \land \omega'' \)
Automatic oracle composition

\[ \omega'' = \omega_d(\text{monarch}) \land (\land_{\text{country}} \omega(\text{country}, \text{monarch})) \]
\[ \omega' = \omega(\text{Europe}, \text{locatedIn}^{-1}) \]

\[ \omega^{**} = \land_{l : \text{family}(l, \text{Romance})} \omega(l, \text{officialLang}^{-1})) \]
\[ \omega^* = \omega(\text{Romance}, \text{family}^{-1}) \]

SELECT ?country WHERE {
   ?country monarch ?monarch .
   ?country locatedIn Europe .
   ?lang family Romance .
}

Completeness verdict = \( \omega^{**} \land \omega^* \land \omega' \land \omega'' \)
Confidence = \( \text{prec}(\omega^{**}) \times \text{prec}(\omega^*) \times \text{prec}(\omega') \times \text{prec}(\omega'') \)
Automatic oracle composition

\[ \omega'' = \omega_d(\text{monarch}) \land (\land_{\text{country}} \omega(\text{country}, \text{monarch})) \]

\[ \omega' = \omega(\text{Europe}, \text{locatedIn}^{-1}) \]

\[ \omega** = \land_{l: \text{family}(l, \text{Romance})} \omega(l, \text{officialLang}^{-1}) \]

\[ \omega* = \omega(\text{Romance}, \text{family}^{-1}) \]

\[ \text{Completeness verdict} = \omega** \land \omega* \land \omega' \land \omega'' \]

Confidence = \text{prec}(\omega**) \times \text{prec}(\omega*) \times \text{prec}(\omega') \times \text{prec}(\omega'')

It could easily lead to false negatives
Automatic oracle composition

\[ \omega'' = \omega_d(\text{monarch}) \land \\
( \land_{\text{country}} \omega(\text{country}, \text{monarch})) \]

\[ \omega' = \omega(\text{Europe}, \land_{\text{locatedIn}}) \]

\[ \omega^{**} = \land_{l: \text{family}(l, \text{Romance})} \omega(l, \land_{\text{family}}) \land_{\text{officialLang}} \]

\[ \omega^* = \omega(\text{Romance}, \land_{\text{family}}) \land_{\text{family}} \]

SELECT ?country WHERE {
    ?country monarch ?monarch .
    ?country locatedIn Europe .
    ?lang family Romance .
}

Completeness verdict = \(\omega^{**} \land \omega^* \land \omega' \land \omega''\)

Confidence = prec(\(\omega^{**}\)) \times prec(\(\omega^*\)) \times prec(\(\omega'\)) \times prec(\(\omega''\))
Automatic oracle composition

\[ \omega'' = \omega_d(\text{monarch}) \land (\bigwedge_{\text{country}} \omega(\text{country, monarch})) \]

\[ \omega' = \omega(\text{Europe, locatedIn}^{-1}) \]

\[ \omega^{**} = \bigwedge_{l : \text{family}(l, \text{Romance})} \omega(l, \text{officialLang}^{-1})) \]

\[ \omega^* = \omega(\text{Romance, family}^{-1}) \]

Use more complex oracles that cover larger parts of the query graph at once

Confidence = \text{prec}(\omega^{**}) \times \text{prec}(\omega^*) \times \text{prec}(\omega') \times \text{prec}(\omega'')

We would like to minimize the number of used oracles

Completeness verdict = \omega^{**} \land \omega^* \land \omega' \land \omega''
Outline

- Completeness in RDF knowledge bases
- Completeness oracles
- Our vision
  - Representations for completeness oracles
  - Reasoning with completeness oracles
  - Enabling completeness in SPARQL
- Summary & conclusions
Enabling completeness in SPARQL

- Calls to completeness oracles could be embedded in the query language
Enabling completeness in SPARQL

- Calls to completeness oracles could be embedded in the query language
  - Example: aggregated number of Spanish speakers in a county per state, only for those states with complete information
Enabling completeness in SPARQL

- Calls to completeness oracles could be embedded in the query language
  - Example: aggregated number of Spanish speakers in a county per state, only for those states with complete information

```
SELECT ?state sum(?nspeak) WHERE {
} GROUP BY ?state HAVING (complete(?nspeak))
```
Enabling completeness in SPARQL

- Calls to completeness oracles could be embedded in the query language
  - Example: aggregated number of Spanish speakers in a county per state, only for those states with complete information

```
SELECT ?state sum(?nspeak) WHERE {
} GROUP BY ?state HAVING (complete(?nspeak))
```
Enabling completeness in SPARQL

For each value of \( ?\text{state} \) check if the bindings for \( ?\text{nspeak} \) are complete.

<table>
<thead>
<tr>
<th>?\text{state}</th>
<th>?\text{county}</th>
<th>?\text{nspeak}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delaware</td>
<td>New Castle Kent Sussex</td>
<td>2000 4300 1200</td>
</tr>
<tr>
<td>Hawaii</td>
<td>Hawaii Kalawao</td>
<td>30000 1200</td>
</tr>
</tbody>
</table>

SELECT \( ?\text{state} \) \text{sum}(?\text{nspeak}) \text{WHERE} \{  
    ?\text{county inState} ?\text{state} .
    ?\text{county spanishSpeakers} ?\text{nspeak} .
\} \text{GROUP BY} ?\text{state} \text{HAVING (complete(?\text{nspeak}))}
Enabling completeness in SPARQL

- For each value of ?state check if the bindings for ?nspeak are complete.

<table>
<thead>
<tr>
<th>?state</th>
<th>?county</th>
<th>?nspeak</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delaware</td>
<td>New Castle, Kent</td>
<td>2000</td>
</tr>
<tr>
<td></td>
<td>Sussex</td>
<td>4300, 1200</td>
</tr>
<tr>
<td>Hawaii</td>
<td>Hawaii, Kalawao</td>
<td>30000, 1200</td>
</tr>
</tbody>
</table>

```
SELECT complete(?nspeak) WHERE {
    ?county inState Delaware .
} }
```

```
SELECT ?state sum(?nspeak) WHERE {
} GROUP BY ?state HAVING (complete(?nspeak))
```
Enabling completeness in SPARQL

- For each value of ?state check if bindings for ?nspeak are complete

<table>
<thead>
<tr>
<th>?state</th>
<th>?county</th>
<th>?nspeak</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delaware</td>
<td>New Castle</td>
<td>2000</td>
</tr>
<tr>
<td></td>
<td>Kent</td>
<td>4300</td>
</tr>
<tr>
<td></td>
<td>Sussex</td>
<td>1200</td>
</tr>
<tr>
<td>Hawaii</td>
<td>Hawaii</td>
<td>30000</td>
</tr>
<tr>
<td></td>
<td>Kalawao</td>
<td>1200</td>
</tr>
</tbody>
</table>

SELECT `complete`(?nspeak) WHERE {
  ?county inState Delaware .
}

SELECT ?state sum(?nspeak) WHERE {
} GROUP BY ?state HAVING (`complete`(?nspeak))

Completeness oracles to the rescue!
Outline

- Completeness in RDF knowledge bases
- Completeness oracles
- Our vision
  - Representations for completeness oracles
  - Reasoning with completeness oracles
  - Enabling completeness in SPARQL
- Summary & conclusions
Summary

• Completeness is a dimension of data quality
  – It determines the value and reliability of the data
  – Existing work provides only completeness statements and oracles for simple queries

• Semantic Web is not completeness-aware
  – Vision
    • Use completeness oracles for simpler queries to infer completeness for arbitrary queries
    • Embed completeness in the SPARQL query language
  – **Goal:** Increase the value of the results delivered by queries
Future work

• Augment existing RDF data with completeness statements and oracles

• Implement reasoning with completeness oracles in SPARQL query engines
  – Extend the SPARQL query language to support the *complete* aggregation function