ORACLE

RDF Extension for Knowledge Graphs

RDFn: A Backward-Compatible RDF Extension for Unasserted and Named Triples

Souri Das, Ph.D.

Architect, Spatial and Graph,
Oracle America Inc.

Dec 16, 2023

Souripriya Das (Souri)

https://www.linkedin.com/in/souripriya-souri-das-ph-d-48801911/

Architect and Manager at Oracle

- Database
- RDF Knowledge Graph
- Property Graph

Education

- Ph.D., Rutgers University
- M.S., Vanderbilt University
- B.Tech., Indian Institute of Technology (IIT), Kharagpur

Standards Activity

- W3C RDB2RDF, Editor of R2RML
- W3C SPARQL 1.0 and 1.1
- W3C RDF 1.1

Publications in Database, Semantic Web, Knowledge Graphs

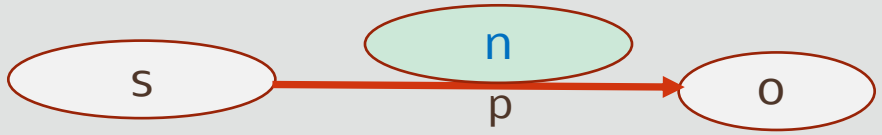
- ICDE, VLDB, EDBT, CIKM
- Patents in Database and Graph technologies

RDFn (RDF with naming): Core Concepts

RDF (default or named) graph → set of <s, p, o> tuples

s → subject, p → predicate, o → object.
(Note: Every tuple is **asserted**)

RDFn (def./named) graph → set of <s, p, o, n> tuples



- *tName* **n** → IRI in an exclusive namespace …/rdfn/ .
- isAsserted → true/false (an attribute of each tuple).

Types of tuples in RDFn: (asserted or unasserted) RDF triples (rdft:), custom-named (custom:), auto-named (auto:)

custom-named tuple

- The tName (tuple name), n, is an IRI in an exclusive namespace for custom names, …/rdfn/custom/ (We use custom: as the prefix in the examples here.)
- The RDF data creator provides the tName IRI.
- The same custom name may be used for multiple tuples, thus allowing making statements about a set of tuples possibly belonging to multiple graphs.

auto-named tuple

- The tName (tuple name), n, is an IRI in the exclusive namespace for auto-gen. names, …/rdfn/auto/. (We use auto: as the prefix in the examples here.)
- The tName for a triple in RDF (default or named) graph g is automatically generated using s, p, o.
- The generated tName is unique in the RDF graph.
- Users may refer to tName using locally unique **alias**.

RDFn: RDF (back.-compatible: asserted triple) + tName (edge-as-endpoint; custom-name for multi-edge) + unasserted

SPARQLn: SPARQL (back.-compat.) + tName IRI/var; isAsserted, isRDF/isAuto/isCustom (annotation; rdf/auto/custom? un/asserted?)

RDFn: Cheat Sheet for Data Loading

#	Input Triple[1]	Comments	Effect
1-a	:s :p :o .	# (same as RDF) # RDF triple # asserted	The target (default or named) graph g will contain a tuple <:s, :p, :o, rdft:…>, with isAsserted = true. The triple-name is generated using :s, :p, :o and is unique in the RDF graph.
2-a	:s :p :o | rdft:… .	# (same as above) # also, sets up an alias	Same as above. Additionally, the alias is bound to the generated, or (in case a matching triple was found), the pre-existing, triple-name.
3-a	:s :p :o | auto:… .	# auto-named # asserted # also, sets up an alias	The target (default or named) graph g will contain a tuple <:s, :p, :o, auto:…>, with isAsserted = true. Additionally, the alias is bound to the generated, or (in case a matching auto-named occurrence triple was found), the pre-existing, auto-name.
4-a	:s :p :o | custom:… .	# custom-named # asserted	The target graph will contain a tuple <:s, :p, :o, custom:…>, with isAsserted = true.
1-u	<< :s :p :o >> .	# RDF triple # unasserted	The target (default or named) graph g will contain a tuple <:s, :p, :o, rdft:…>. The value of its isAsserted attribute will be set to false, unless same tuple with isAsserted=true was already present in the graph. The triple-name is generated using :s, :p, :o and is unique in the RDF graph.
2-u	<< :s :p :o >> | rdft:… .	# (same as above) # also, sets up an alias	Same as above. Additionally, the alias is bound to the generated, or (in case a matching triple was found), the pre-existing, triple-name.
3-u	<< :s :p :o >> | auto:… .	# auto-named # unasserted # also, sets up alias	The target (default or named) graph g will contain a tuple <:s, :p, :o, auto:…>. The value of its isAsserted attribute will be set to false, unless same tuple with isAsserted=true was already present in the graph. The auto-name is generated using :s, :p, :o and is unique in the RDF graph.
4-u	<< :s :p :o >> | custom:… .	# custom-named # unasserted	The target graph will contain a tuple <:s, :p, :o, custom:…>. The value of its isAsserted attribute will be set to false, unless same tuple with isAsserted=true was already present in the graph.

[1] Note: For RDF triples and auto-named triples, user's input only specifies the aliases (as placeholders: rdft:…, auto:…), not the actual names.

RDFn: Cheat Sheet for Query

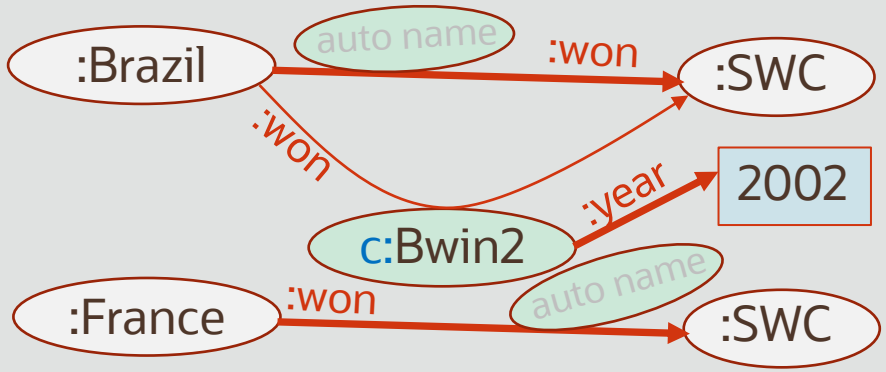
Triple-pattern	Comments	FILTER	Effect
?s ?p ?o .	# (same as SPARQL) # RDF triple # asserted		Looks at all **asserted RDF** triples in the target graph for a match.
?s ?p ?o | ?n .	# asserted # also, binds ?n to actual tName	isRDF(?n) isAuto(?n) isCustom(?n)	Looks at all **asserted** triples in the target graph for a match. Use of the isRDF/isAuto/isCustom functions in the FILTER limits the target to RDF triples, auto-named triples, or custom-named triples, respectively.
<< ?s ?p ?o >> .	# RDF triple # asserted or unasserted		Looks at all **RDF** triples – **asserted or unasserted** – in the target graph for a match.
<< ?s ?p ?o >> | ?n .	# (same as above) # also, binds ?n to actual tName	!isAsserted(?n) isRDF(?n) isAuto(?n) isCustom(?n)	Looks at all **asserted** or **unasserted** triples in the target graph for a match. Use of the !isAsserted() function in the FILTER limits the target to unasserted triples only. Use of the isRDF/isAuto/isCustom functions in the FILTER limits the target to RDF triples, auto-named triples, or custom-named triples, respectively.

RDFn Data Loading: in Batches, over Time

Load Batch #1

Brazil won soccer world cup *twice*. France won once. Brazil's 2nd win was in year 2002.

```
:Brazil :won :SWC | auto:Bwin .          # A1, auto-named
:Brazil :won :SWC | custom:Bwin2 .       # A2, custom-named
:France :won :SWC | auto:Fwin .          # A3, auto-named
custom:Bwin2 :year 2002 .                # A4, RDF triple
```

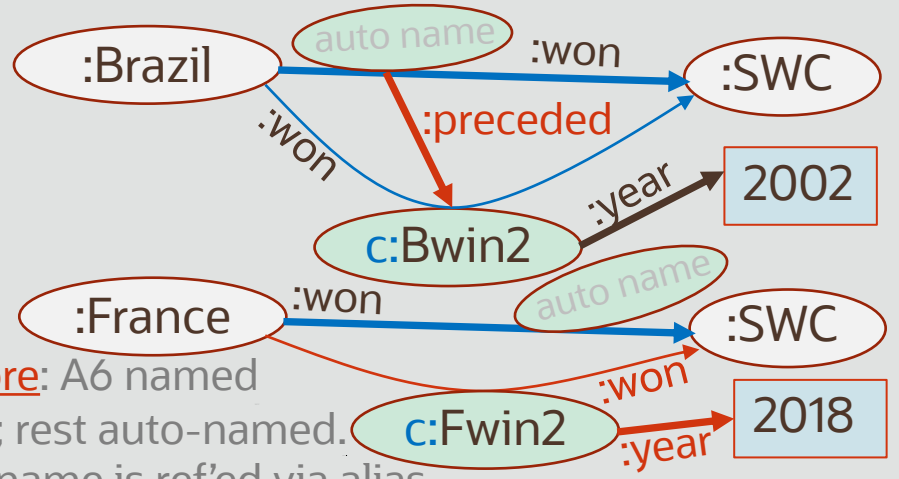


Four edges: A2 named manually; rest auto-named.

Load Batch #2

Brazil's 1st win preceded their 2nd win. France won *again* and that win came in year 2018.

```
:Brazil :won :SWC | auto:Bw1 .            # sets up alias to A1
auto:Bw1 :preceded custom:Bwin2 .         # A5, auto-named
:France :won :SWC | custom:Fwin2 .        # A6, custom-named
custom:Fwin2 :year 2018 .                 # A7, RDF triple
```



Three more: A6 named manually; rest auto-named. A1's auto name is ref'ed via alias.

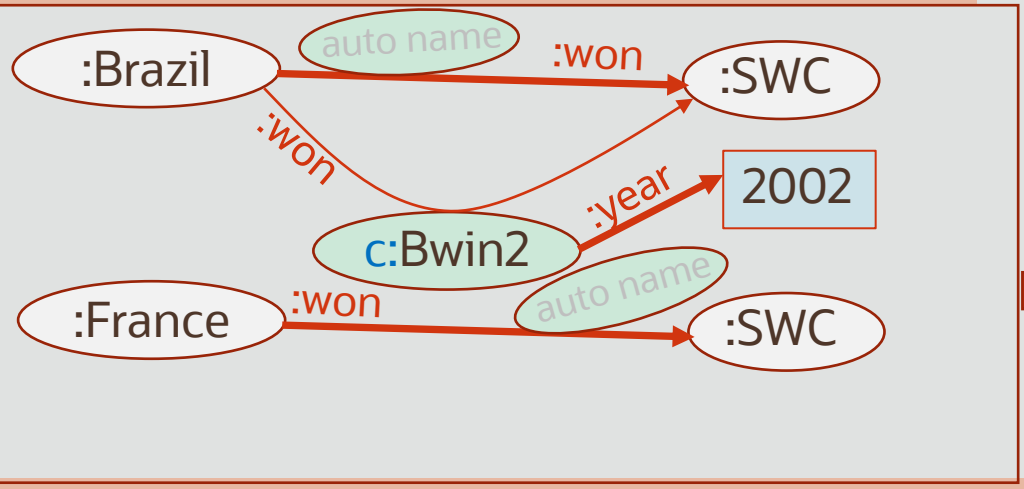
Note: The following *first-time changes in #2* were accommodated just by adding new edges. No deletions.

1. A1 is used as an endpoint – as *subject* in A5. Its auto-gen. tName (created during #1) is ref'ed via *alias*.
2. A3 became part of a multi-edge, { A3, A6 }. User supplied an *IRI* as unique tName for new triple A6.

SPARQLn Queries Remain Valid Throughout

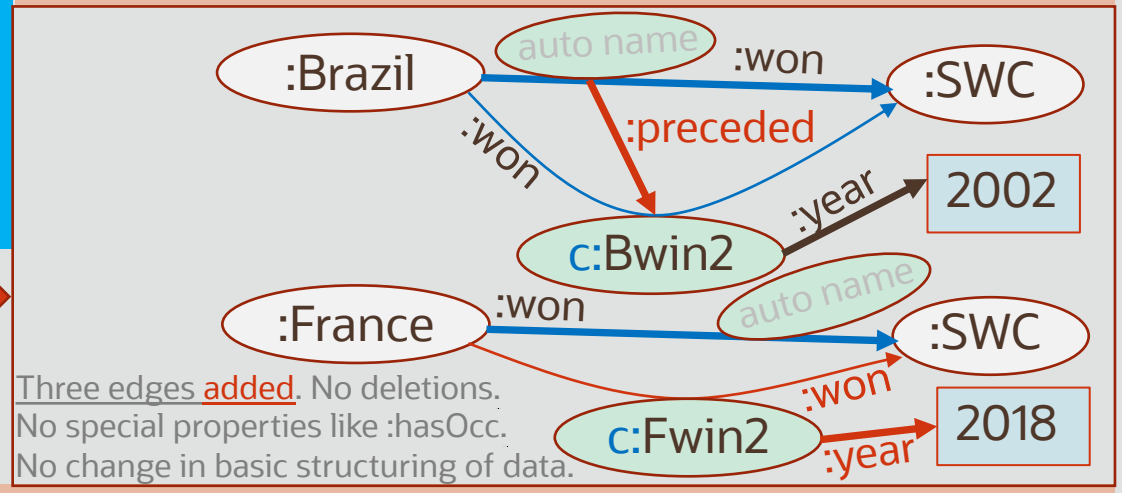
Load Batch #1

Brazil won soccer world cup *twice*. France won once. Brazil's 2nd win was in year 2002.



Load Batch #2

Brazil's 1st win preceded their 2nd win. France won *again* and that win came in year 2018.



SPARQLn

Query: For each world cup winner, find win count and last (known) year of winning.

```
SELECT ?c (COUNT(*) as ?cnt) (MAX(?yr) as ?last)
WHERE { ?c :won :SWC | ?win . OPTIONAL { ?win :year ?yr } } GROUP BY ?c
```

RESULT: [?c= :Brazil, ?cnt=2, ?last=2002], [?c= :France, ?cnt=1]

RESULT: [?c= :Brazil, ?cnt=2, ?last=2002], [?c= :France, ?cnt=2, ?last=2018]

Why pre-existing queries remain valid in spite of changes like *new edge-as-endpoint, new multi-edge?*

- Loading of batch #2 in this example was accommodated just by adding new triples. No deletions or replacements were needed.
- The addition did not affect the structuring of the data and no special properties were introduced.
- Adding a new edge parallel to an existing edge simply involved adding a triple with the same s-p-o, but a (new) custom-name.
- Adding a new edge-as-endpoint simply involved using the name of the triple as the subject (source) of the new triple (edge).

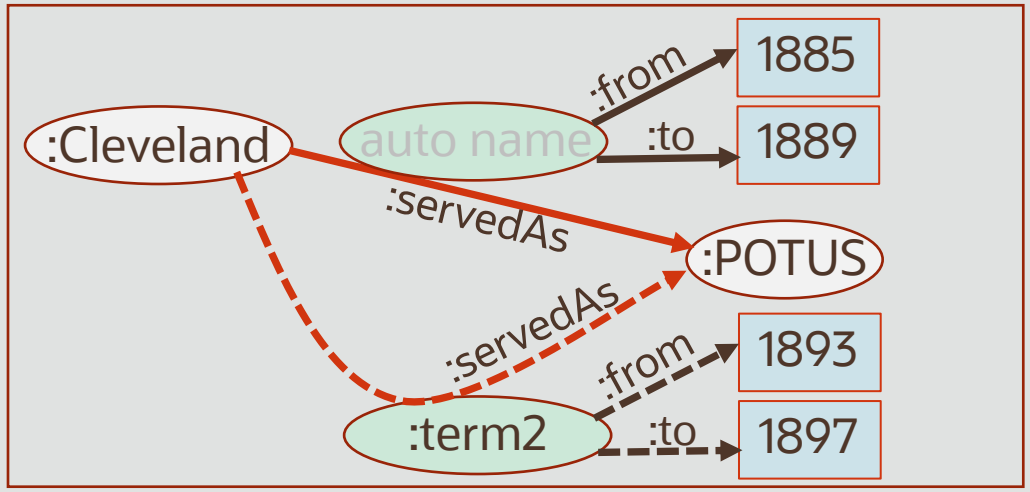
RDFn vs. RDF-star (1): Multi-Edge Handling

Data: Cleveland served as the President for *two* (non-consecutive) terms: 1885-1889 and 1893-1897.

RDFn

:Cleveland :servedAs :POTUS | (auto:term1, custom:term2) .
auto:term1 :from 1885 ; :to 1889 .
custom:term2 :from 1893 ; :to 1897 .

6 triples



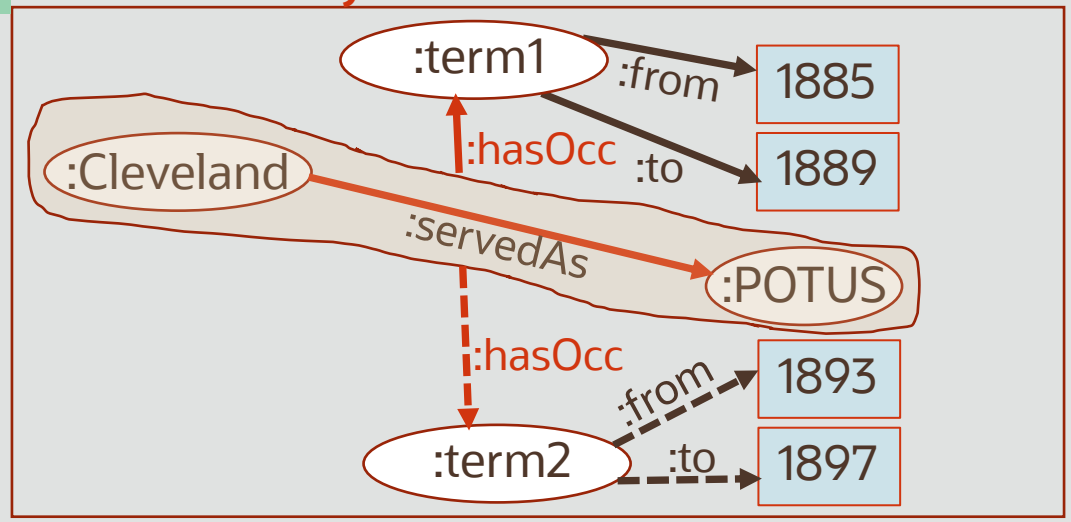
- When new data comes in about Cleveland's second term, a new custom-named :servedAs triple is created, and the custom-name is used as the subject for adding info about the from/to year for that term.
- Both minimizes triple count and maintains uniformity of representation.

RDF-star

:Cleveland :servedAs :POTUS {| :hasOccurrence :term1, :term2 |} .
:term1 :from 1885 ; :to 1889 .
:term2 :from 1893 ; :to 1897 .

7 triples

Scheme 1: Always Use :hasOccurrence



- Even info about Cleveland first term is modeled using the :hasOccurrence property. Thus, when new data comes in about Cleveland's second term, a new :hasOccurrence triple is created to represent this info and so on.
- Provides uniformity of representation but increases number of triples.

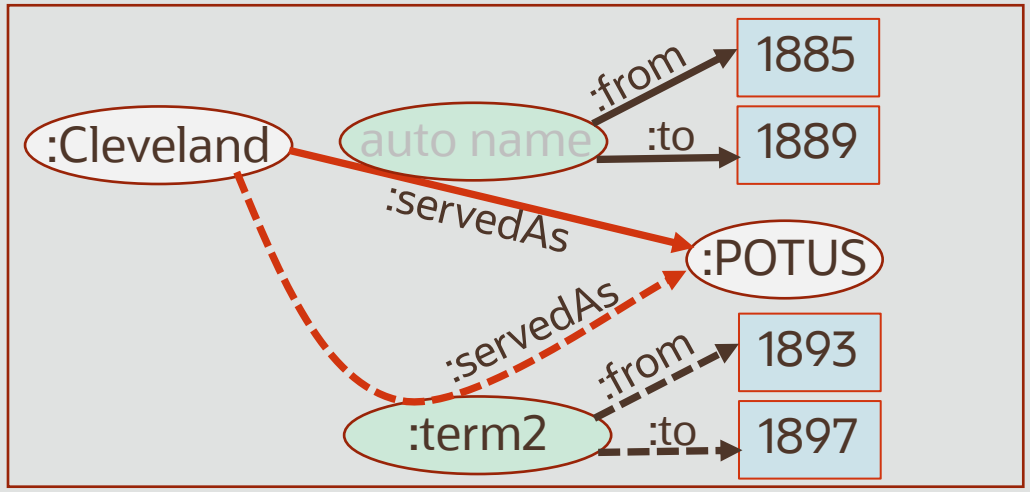
RDFn vs. RDF-star (2): Multi-Edge Handling

Data: Cleveland served as the President for *two* (non-consecutive) terms: 1885-1889 and 1893-1897.

RDFn

:Cleveland :servedAs :POTUS | (auto:term1, custom:term2) .
auto:term1 :from 1885 ; :to 1889 .
custom:term2 :from 1893 ; :to 1897 .

6 triples



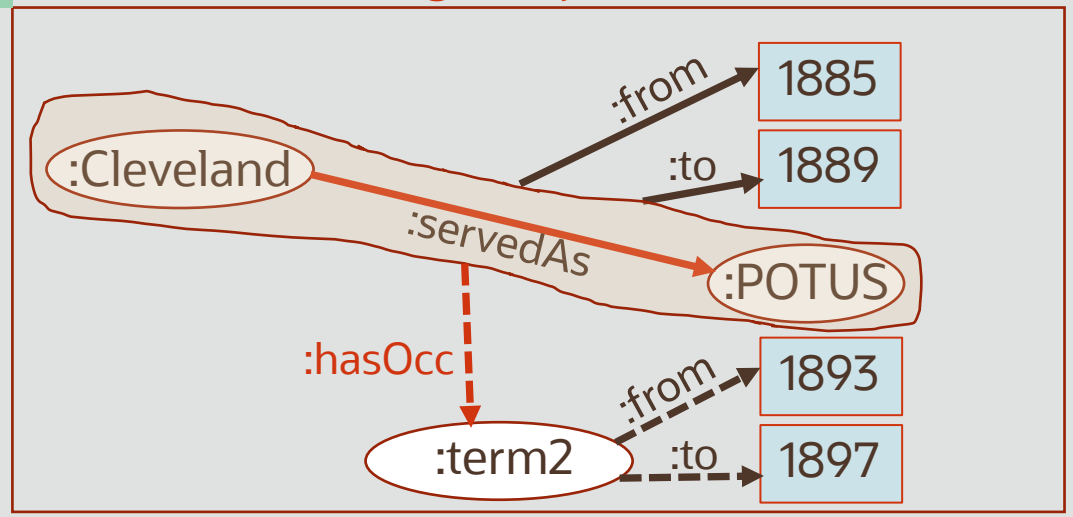
- When new data comes in about Cleveland's second term, a new custom-named :servedAs triple is created, and the custom-name is used as the subject for adding info about the from/to year for that term.
- Both minimizes triple count and maintains uniformity of representation.

RDF-star

:Cleveland :servedAs :POTUS {| :from 1885 ; :to 1889 ;
:hasOccurrence :term2 |} .
:term2 :from 1893 ; :to 1897 .

6 triples

Scheme 2: Multi-Edge-only Use of :hasOccurrence



- When new data comes in about Cleveland's second term, a :hasOccurrence triple is created to represent this info and the occurrence IRI is used as the subject for adding info about the from/to year for that term.
- Minimizes triple count but loses uniformity of representation.

RDFn vs. *always-using* :hasOcc in RDF-star

Data: Consecutive term info for 44 US Presidents. (Total terms = 45: Cleveland had *two* non-consecutive terms.)

RDFn

```
:Washington :servedAs :POTUS | auto:pres1 .
auto:pres1 :from 1789 ; :to 1797 .
...
:Cleveland :servedAs :POTUS | (auto:pres22, custom:pres24) .
auto:pres22 :from 1885 ; :to 1889 .
custom:pres24 :from 1893 ; :to 1897.
...
:Trump :servedAs :POTUS | auto:pres45 .
auto:pres45 :from 2017 ; :to 2021 .
```

RDF-star

```
:Washington :servedAs :POTUS {| :hasOccurrence :pres1 |} .
:pres1 :from 1789 ; :to 1797 .
...
:Cleveland :servedAs :POTUS {| :hasOccurrence :pres22, :pres24 |} .
:pres22 :from 1885 ; :to 1889 .
:pres24 :from 1893 ; :to 1897 .
...
:Trump :servedAs :POTUS {| :hasOccurrence :pres45 |} .
:pres45 :from 2017; :to 2021 .
```

Data	RDFn	RDF-star	Remarks
Triple Count	(45*1+1) + (45+1)*2 = 138	(45*2 + 1) + (45+1)*2 = 183	• In RDF-star, an occurrence triple, and hence an IRI (or blank node), is created for each presidential term. • In RDFn, an IRI is created only for the 2nd term of Cleveland. No special occ. triples.
IRIs (or blank nodes) for president terms	1 (rest are locally unique aliases for the auto-generated tNames)	45	

Query: For each US President, find the start and end year of the term(s) he served as president.

SPARQLn

```
Select ?who ?start ?end
{ ?who :servedAs :POTUS | ?term .
  ?term :from ?start ; :to ?end }   3 triple-patterns
```

SPARQL-star

```
Select ?who ?start ?end
{ ?who :servedAs :POTUS {| :hasOccurrence ?term |}
  ?term :from ?start ; :to ?end }   4 triple-patterns
```

RDFn vs. *multi-edge-only* :hasOcc in RDF-star

Data: Consecutive term info for 44 US Presidents. (Total terms = 45: Cleveland had *two* non-consecutive terms.)

RDFn

```
:Washington :servedAs :POTUS | auto:pres1 .
auto:pres1 :from 1789 ; :to 1797 .
...
:Cleveland :servedAs :POTUS | (auto:pres22, custom:pres24) .
auto:pres22 :from 1885 ; :to 1889 .
custom:pres24 :from 1893 ; :to 1897.
...
:Trump :servedAs :POTUS | auto:pres45 .
auto:pres45 :from 2017 ; :to 2021 .
```

RDF-star

```
:Washington :servedAs :POTUS {| :from 1789 ; :to 1797 |} .
...
:Cleveland :servedAs :POTUS {| :from 1885 ; :to 1889;
                               :hasOccurrence :pres24 |} .
:pres24 :from 1893 ; :to 1897 .
...
:Trump :servedAs :POTUS {| :from 2017; :to 2021 |} .
```

Data	RDFn	RDF-star	Remarks
Triple Count	(45*1+1) + (45+1)*2 = 138	(45*1 + 1) + (45+1)*2 = 138	• In RDF-star, an occurrence triple, and hence an IRI, is created only for Cleveland 2nd term. • In RDFn, an IRI is created only for the 2nd term of Cleveland. No special occ. triples.
IRIs (or blank nodes) for president terms	1 (rest are locally unique aliases for the auto-generated tNames)	1	

Query: For each US President, find the start and end year of the term(s) he served as president.

SPARQLn

```
Select ?who ?start ?end                                   Simple
{ ?who :servedAs :POTUS | ?term .
  ?term :from ?start ; :to ?end }   3 triple-patterns
```

SPARQL-star

```
Select ?who ?start ?end                                   Complex
{ ?who :servedAs :POTUS {| :from ?start ; :to ?end |} }
UNION
{ ?who :servedAs :POTUS {| :hasOccurrence ?term |}
  ?term :from ?start ; :to ?end }}   7 triple-patterns
```

ORACLE