Getting Started with RDF & SPARQL

The basics of RDF graphs and the SPARQL query language

Taught by:

Brendan Newlon
Solutions Architect

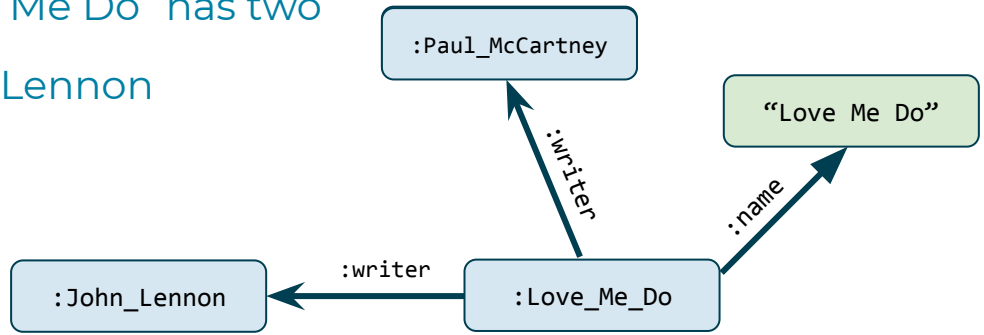
Learning Objectives

- Learn the fundamentals of RDF graphs
- Understand the core ideas of SPARQL queries
- Describe the common types of SPARQL queries
- Demonstrate the use of SPARQL to create or update RDF data
- Learn how to work with named graphs

Introduction to RDF

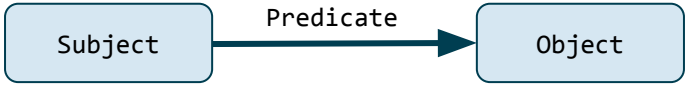
Why RDF?

- Resource Description Framework (RDF) provides a standardized universal model for representing data and its meaning
  - Support hybrid, varied, and changing data models with ease
  - Easy to represent any change in data or schema
  - Interoperable and composable
- Eg. The song with the name “Love Me Do” has two writers, Paul McCartney and John Lennon

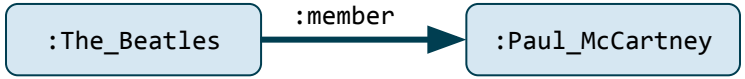


Key Terms 1: The Basic Idea

- Let's say the class (ie. category) "artist" includes both solo artists and bands, and a member of a band is a solo artist
- The RDF way to describe these relationships is based on how we would express it in speech:

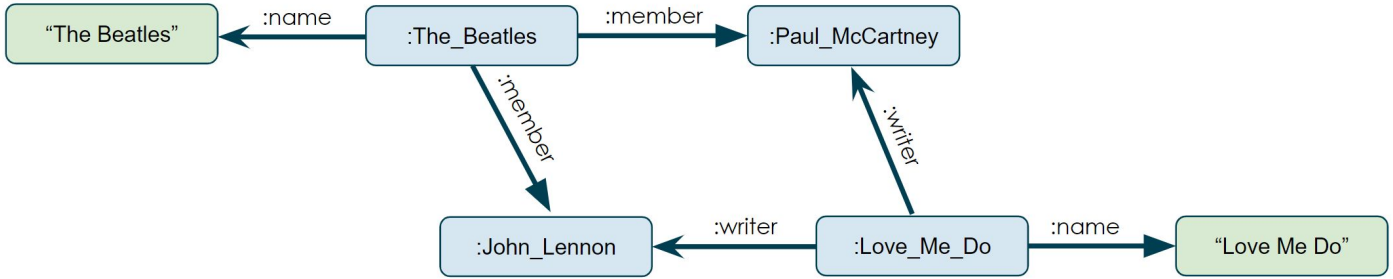


Eg. **The Beatles** has as a member **Paul McCartney**



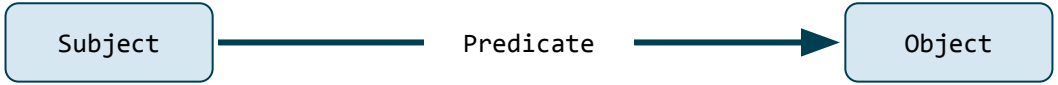
Key Terms 2: Objects

- By **class** we mean a type of thing (eg. band or artist)
- A class is made up of a set of **individuals** (eg. The Beatles or John Lennon), which can also be called instances or objects
- A class or individual can be the **subject** or the **object** in a 3-part RDF structure called an RDF triple

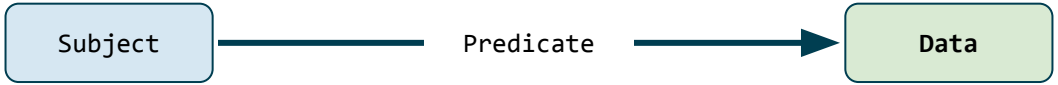


Key Terms 3: Properties

- The middle part of an RDF triple is the **predicate**, which is used in two ways. When it describes a **relationship** between two objects (classes or individuals) in our model, then it is called an **object property**

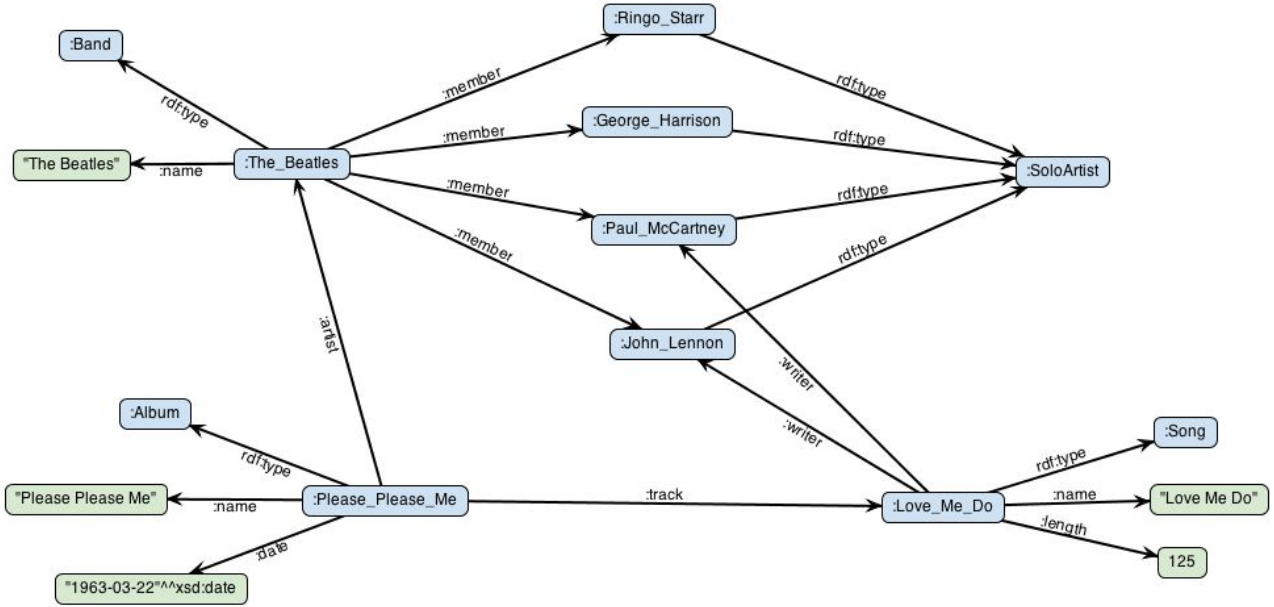


- If the predicate provides data (a number, date, string, etc.) about an object, it is called a **data property** describing an **attribute**



Key Terms 4: Graphs

- Taken together, these elements make up a **graph**
- In a graph, points representing objects or data are called **nodes** while the predicates that connect them (either object properties or data properties) are called **edges**



Key Terms: Review

- There are two kinds of **objects**: classes and individuals/instances
- **Classes** are sets, collections, types of objects, kinds of things
- **Individuals** (or instances) are what a class groups together
- Properties come in two types. An **object property** is a relationship between two things. **Datatype properties** are attributes of one thing
- In graph representation diagrams, classes & individuals are called **nodes** while properties are called **edges**
- In **RDF triples**, classes and individuals are the **subjects** or **objects**, while properties correspond to **predicates.** A set of RDF triples is called an **RDF graph**

RDF Concepts

- **IRI:** Nodes and edges with a unique identifier
- **Literal:** Nodes representing values like numbers and dates
- **Blank node:** Nodes without an explicit identifier

IRI

- Internationalized Resource Identifier

```
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
```

```
http://stardog.com/tutorial/The_Beatles
```

```
mailto:John_Doe@example.com
```

```
urn:isbn:9788026874256
```

```
tag:stardog.com,2018:product:stardog
```

Prefixed Name

- An IRI looks like this

```
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
```

- Using a prefix declaration for its namespace

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

- Can be shortened to a prefixed name

```
rdf:type
```

Literals

- Literals are written in quotes followed by their datatype IRI

```
"1963-03-22" ^^xsd:date
```

```
"1963-03-22T21:44:00Z"^^xsd:dateTime
```

- Datatype can be omitted for strings:

```
"The Beatles"
```

```
"The Beatles"^^xsd:string
```

- Datatype and quotes can be omitted for some datatypes

125	3.14	3.2E4	true
integer	decimal	double	boolean

RDF Serialization

- Declare prefixes

```
PREFIX :<http://stardog.com/tutorial/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

- Write subject, predicate, object followed by a '.'

```
:The_Beatles    rdf:type    :Band .
:The_Beatles    :name       "The Beatles" .
```

Turtle Syntax

```
PREFIX :<http://stardog.com/tutorial/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

:Love_Me_Do      rdf:type    :Song .
:Love_Me_Do      :name       “Love Me Do" .
:Love_Me_Do      :length     “125” ^^xsd:integer .
:Love_Me_Do      :writer     :John_Lennon .
:Love_Me_Do      :writer     :Paul_McCartney .
```

Literal Shorthand

```
PREFIX :<http://stardog.com/tutorial/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

:Love_Me_Do     rdf:type   :Song .
:Love_Me_Do     :name      “Love Me Do" .
:Love_Me_Do     :length    125 .
:Love_Me_Do     :writer    :John_Lennon .
:Love_Me_Do     :writer    :Paul_McCartney .
```

Shorthand for rdf:type

```
PREFIX :<http://stardog.com/tutorial/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

:Love_Me_Do     a          :Song .
:Love_Me_Do     :name      “Love Me Do" .
:Love_Me_Do     :length    125 .
:Love_Me_Do     :writer    :John_Lennon .
:Love_Me_Do     :writer    :Paul_McCartney .
```

Same Subject

```
PREFIX :<http://stardog.com/tutorial/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

:Love_Me_Do     a          :Song ;
                :name      “Love Me Do" ;
                :length    125 ;
                :writer    :John_Lennon ;
                :writer    :Paul_McCartney .
```

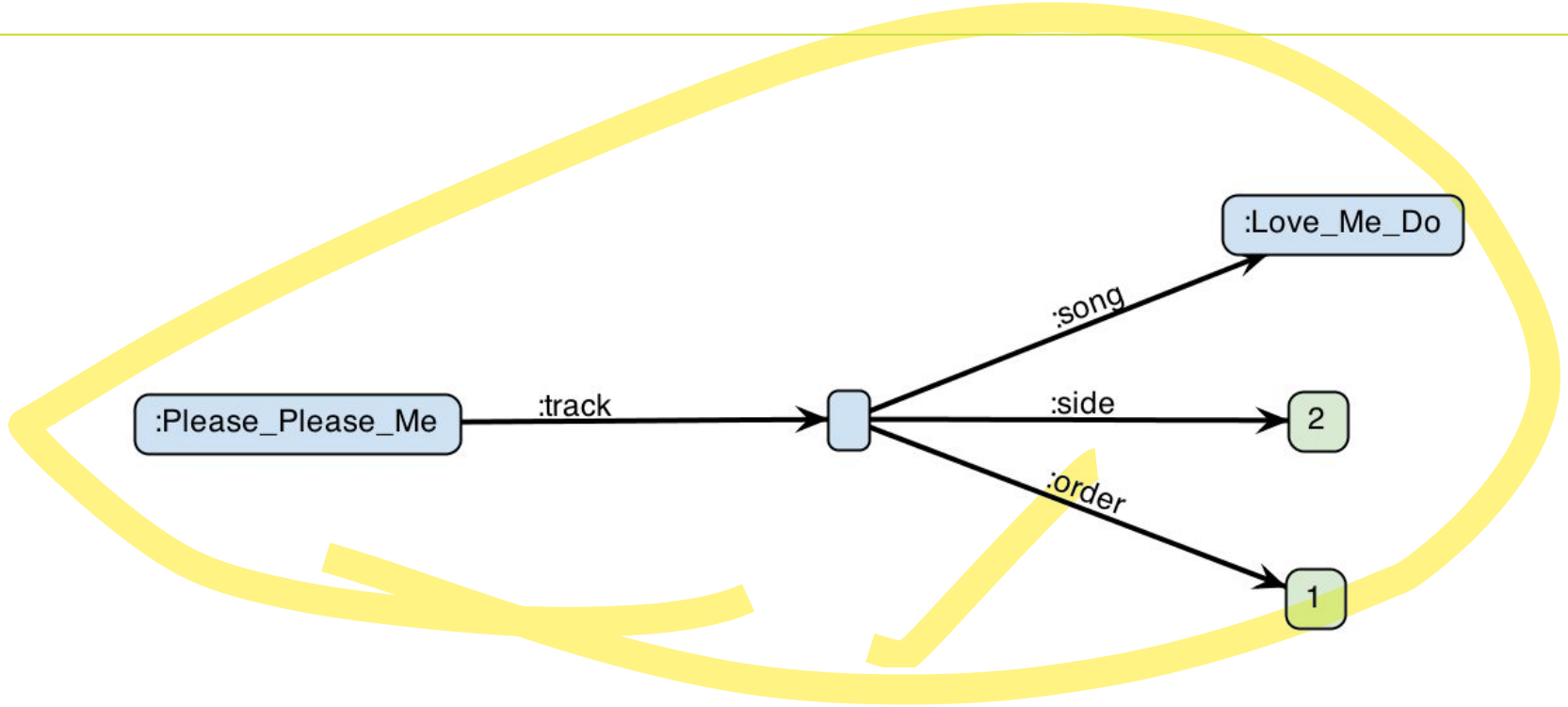
Same Subject and Predicate

```
PREFIX :<http://stardog.com/tutorial/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

:Love_Me_Do     a          :Song ;
                :name      “Love Me Do" ;
                :length    125 ;
                :writer    :John_Lennon ,
                           :Paul_McCartney .
```

Ignore Whitespace

```
PREFIX :<http://stardog.com/tutorial/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
:Love_Me_Do a :Song ;
          :name “Love Me Do" ;
          :length 125 ;
          :writer :John_Lennon , :Paul_McCartney .
```

Blank Nodes



B-Node Serialization

```
:Please_Please_Me :track _:s2n1 .
_:s2n1   :side 2 ;
         :order 1 ;
         :song :Love_Me_Do .
```

```
:Please_Please_Me :track _:s2n1 [
         :side 2 ;
         :order 1 ;
         :song :Love_Me_Do ] .
```

SPARQL

Triple Patterns

- A **triple pattern** is a triple with zero or more variables

```
?band rdf:type :Band .
```

```
?album :artist ?artist .
```

```
:Love_Me_Do :writer ?writer .
```

```
:The_Beatles ?p ?o .
```

- Triple patterns match the triples in the graph
- Each matching triple produces one result

SELECT Query: The Main Query Form in SPARQL

- It has two basic components:
  1. A list of selected variables
  2. Triple patterns to match
- Results are returned as a table where each selected variable is a column and each pattern match is a row

```
SELECT ?band
WHERE {
  ?band rdf:type :Band .
}
```

```
SELECT ?album ?artist
WHERE {
  ?album rdf:type :Album .
  ?album :artist ?artist .
  ?artist rdf:type :SoloArtist .
}
```

Single Triple Pattern

```
SELECT ?album
WHERE {
  ?album rdf:type :Album .
}
```

```
SELECT *
{
  ?album a :Album
}
```

Joins

```
SELECT * {
  ?album a :Album .
  ?album :artist ?artist .
}
```

```
SELECT * {
  ?album a :Album .
  ?album :artist ?artist .
  ?artist a :SoloArtist .
}
```

Optional Join

```
SELECT ?song ?length {
   ?song a :Song .
   OPTIONAL {
       ?song :length ?length .
   }
}
```

Subqueries

```
SELECT (avg(?count) AS ?avgCount)
{
    SELECT ?year (count(?album) AS ?count)
    {
        ?album a :Album ;
            :date ?date .
        BIND (year(?date) AS ?year)
    }
    GROUP BY ?year
}
```

Alternatives

```
SELECT ?name
{
   { ?artist a :SoloArtist }
   UNION
   { ?artist a :Band }
   ?artist :name ?name
}
```

Negation

```
SELECT ?song {
   ?song a :Song .
   FILTER (
      NOT EXISTS {
        ?song :length ?length .
      }
   )
}
```

Sort Results

```
SELECT *
{
  ?album a :Album ;
         :artist ?artist ;
         :date ?date
}
ORDER BY ?date
```

Limit Results

```
SELECT *
{
  ?album a :Album ;
         :artist ?artist ;
         :date ?date
}
ORDER BY ?date
LIMIT 2
```

Offset Results

```
SELECT *
{
  ?album a :Album ;
         :artist ?artist ;
         :date ?date
}
ORDER BY ?date
LIMIT 2
OFFSET 2
```

Filtering Results

```
SELECT *
{
  ?album a :Album ;
         :artist ?artist ;
         :date ?date
  FILTER (year(?date) >= 1970)
}
ORDER BY ?date
```

Binding Variables

```
SELECT *
{
  ?album a :Album ;
         :artist ?artist ;
         :date ?date
  BIND (year(?date) AS ?year)
  FILTER (?year >= 1970)
}
ORDER BY ?date
```

Removing Duplicates

```
SELECT DISTINCT ?year
{
  ?album a :Album ;
         :artist ?artist ;
         :date ?date
  BIND (year(?date) AS ?year)
}
ORDER BY ?year
```

Aggregation

```
SELECT (min(?date) as ?minDate) (max(?date) as ?maxDate)
{
   ?album a :Album ;
           :date ?date
}
```

Grouping Results

```
SELECT ?year (count(?album) AS ?count)
{
   ?album a :Album ;
           :date ?date ;
   BIND (year(?date) AS ?year)
}
GROUP BY ?year
ORDER BY desc(?count)
```

Property Paths

```
select distinct ?cowriter
{
   :Paul_McCartney ^:writer/:writer ?cowriter
   FILTER (?cowriter != :Paul_McCartney)
}
order by ?cowriter
```

Recursive Paths

```
select distinct ?cowriter
{
   :Paul_McCartney (^:writer/:writer)+ ?cowriter
   FILTER (?cowriter != :Paul_McCartney)
}
order by ?cowriter
```

Query Types

ASK Query

```
ASK {
    ?band a :Band .
    ?song :writer ?band .
}
```

DESCRIBE Query

```
DESCRIBE :The_Beatles
```

```
DESCRIBE ?band
WHERE {
   ?band a :Band ;
         :name ?name
   FILTER(contains(?name, "The"))
}
```

CONSTRUCT Query

```
CONSTRUCT WHERE {
   ?band a :Band ;
         :member ?member
}
```

```
CONSTRUCT {
   ?member a :BandMember
}
WHERE {
   ?band a :Band ;
         :member ?member
}
```

Updates

INSERT/DELETE Triples

```
INSERT DATA {
  :Love_Me_Do a :Song ;
      :name "Love Me Do" ;
      :length 125 ;
      :writer :John_Lennon , :Paul_McCartney .
};

DELETE DATA {
  :Love_Me_Do a :Song ;
      :name "Love Me Do" ;
};
```

INSERT Query

```
INSERT {
   ?member a :BandMember
}
WHERE {
   ?band a :Band ;
         :member ?member
}
```

INSERT/DELETE Query

```
DELETE {
   ?song :length ?seconds
}
INSERT {
   ?song :length ?duration
}
WHERE {
   ?song a :Song ;
        :length ?seconds
   BIND(?seconds * "PT1S"^^xsd:dayTimeDuration AS ?duration)
}
```

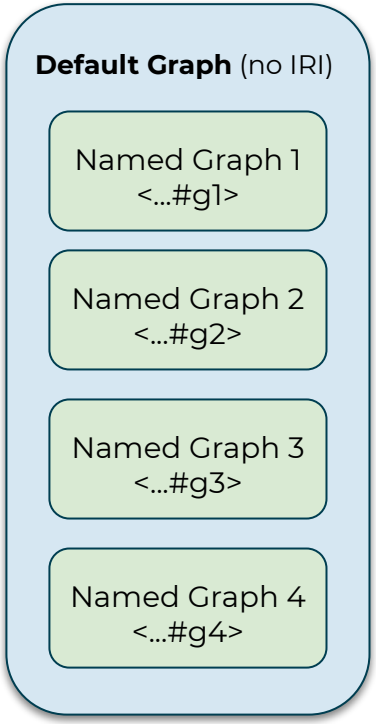
Graph Management

```
LOAD <http://...> TO :targetGraph          # load triples into graph

CLEAR :targetGraph                         # remove triples from graph

ADD :sourceGraph TO :targetGraph           # add triples from one graph to another

COPY :sourceGraph TO :targetGraph          # like ADD but CLEAR target graph first

MOVE :sourceGraph TO :targetGraph          # like COPY but CLEAR source graph
                                           # afterwards
```

Named Graphs

RDF Datasets

- An RDF dataset is a collection of RDF graphs:
  - There is exactly one default graph
    - It does not have a name
    - May be empty or contain RDF triples
  - Zero or more named graphs
    - A named graph is an RDF graph identified by an IRI
    - Graph names are unique within an RDF dataset



RDF Data in TriG: Turtle with Named Graphs

- RDF data for the default graph and zero or more named graphs can be serialized in a single document
- Use GRAPH to specify a named graph followed by its triples

```
GRAPH :Artist {
    :The_Beatles a :Band .
    ...
}
GRAPH :Album {
    :Please_Please_Me rdf:type :Album .
    ...
}
}
```

Specifying SPARQL Dataset

```
PREFIX ex: <...>
SELECT *
FROM ex:g1
FROM ex:g4
FROM NAMED ex:g1
FROM NAMED ex:g2
FROM NAMED ex:g3
WHERE {
        ...Pattern A...
     GRAPH ex:g3 {
        ...Pattern B...
     }
     GRAPH ?graph {
        ...Pattern C...
     }
}
```

- A query can use **FROM** to override the default graph and temporarily treat the merge of one or more graphs as though they are the default graph
- **FROM NAMED** determines which graphs can be available as named graphs in the query
- **GRAPH** directs a query either to a particular named graph, or to any of the available named graphs

Based on: http://www.slideshare.net/LeeFeigenbaum/sparql-cheat-sheet

Specifying SPARQL Dataset

```
PREFIX ex: <...>
SELECT *
FROM ex:g1
FROM ex:g4
FROM NAMED ex:g1
FROM NAMED ex:g2
FROM NAMED ex:g3
WHERE {
        ...Pattern A...
     GRAPH ex:g3 {
        ...Pattern B...
     }
     GRAPH ?graph {
        ...Pattern C...
     }
}
```

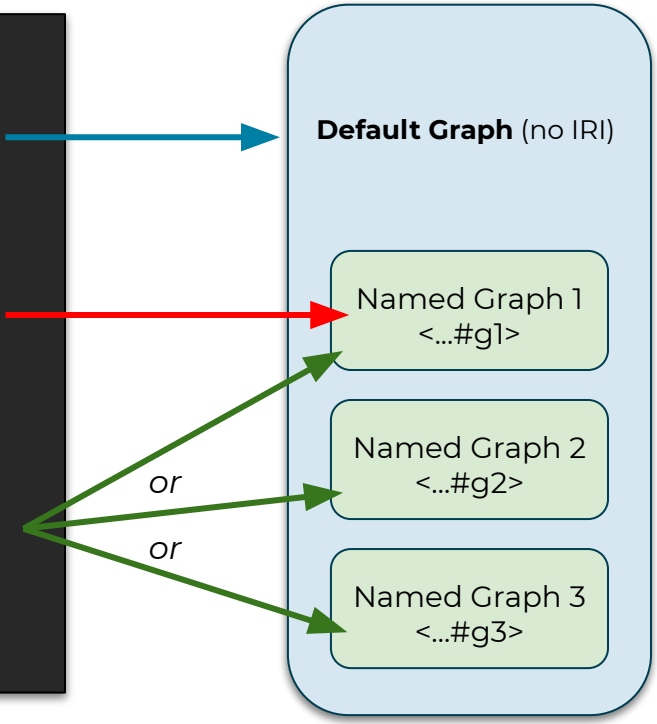
In this example...

- **Pattern A** results come from the merge of g1 and g4 which temporarily act as the default graph
- **Pattern B** results can only come from the named graph g3
- **Pattern C** results may come from any of the named graphs available to this query: g1, g2, or g3. The ?graph variable will specify the source(s) of any results

Based on: http://www.slideshare.net/LeeFeigenbaum/sparql-cheat-sheet

Querying a Specific Dataset

```
SELECT * {
    ...graph pattern...                 #   default graph
                                            and any named
                                            graphs
    GRAPH :g1 {
        ...graph pattern...             # a specific graph
    }
    GRAPH ?graph {
        ...graph pattern...             # any named graph
    }
}
```



Based on: http://www.slideshare.net/LeeFeigenbaum/sparql-cheat-sheet

Named Graph Query

```
SELECT * {
 GRAPH :Album {
   ?album a :Album .
   ?album :artist ?artist .
 }
 GRAPH :Artist {
   ?artist a :SoloArtist .
 }
}
```

Override Default Graph

```
SELECT *
FROM :Album
{
  ?album a :Album .
  ?album :artist ?artist .
  ?album :date ?date .
}
```

Learning Objectives

Learning Objectives

- Learn the fundamentals of RDF graphs
- Understand the core ideas of SPARQL queries
- Describe the common types of SPARQL queries
- Demonstrate the use of SPARQL to create or update RDF data
- Learn how to work with named graphs

Thank you