Virtual Graphs

Defining and Applying Virtual Graphs

Taught by:

Al Baker

VP, Enterprise Solutions

Learning Objectives

- Defining and configuring a Virtual Graph
- Options available for virtual mapping to data sources
- How Virtual Graphs interact with other features

Overview

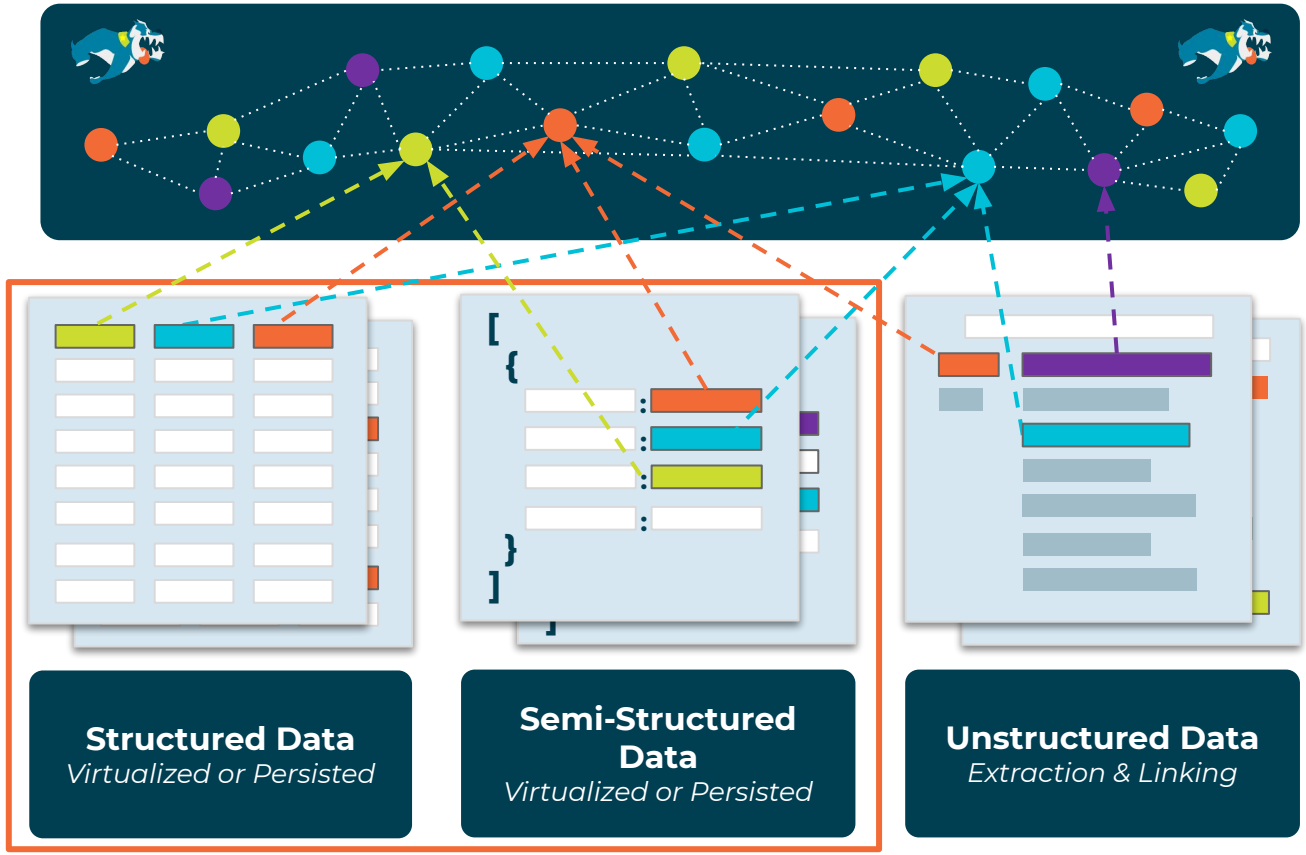
What is a Virtual Graph?

- Enables extending the graph over existing data sources without moving data
- Virtual Graphs managed as named graphs in the knowledge graph
- Three key components:
  - Unique name
  - Data source administration
  - Virtual Graph mappings

Virtual Graph Sources

- Relational databases
  - Traditional (ex. Oracle, Postgresql, MySQL)
  - Big Data (ex. Apache Hive, AWS Athena, Google BigQuery, SparkSQL)
- NoSQL databases (ex. MongoDB, Cassandra, DataStax, ElasticSearch, CosmosDB)
- SPARQL engine / sources
- Cloud services
- Files / unstructured data
- CRM
- Data analytics

Example Knowledge Graph

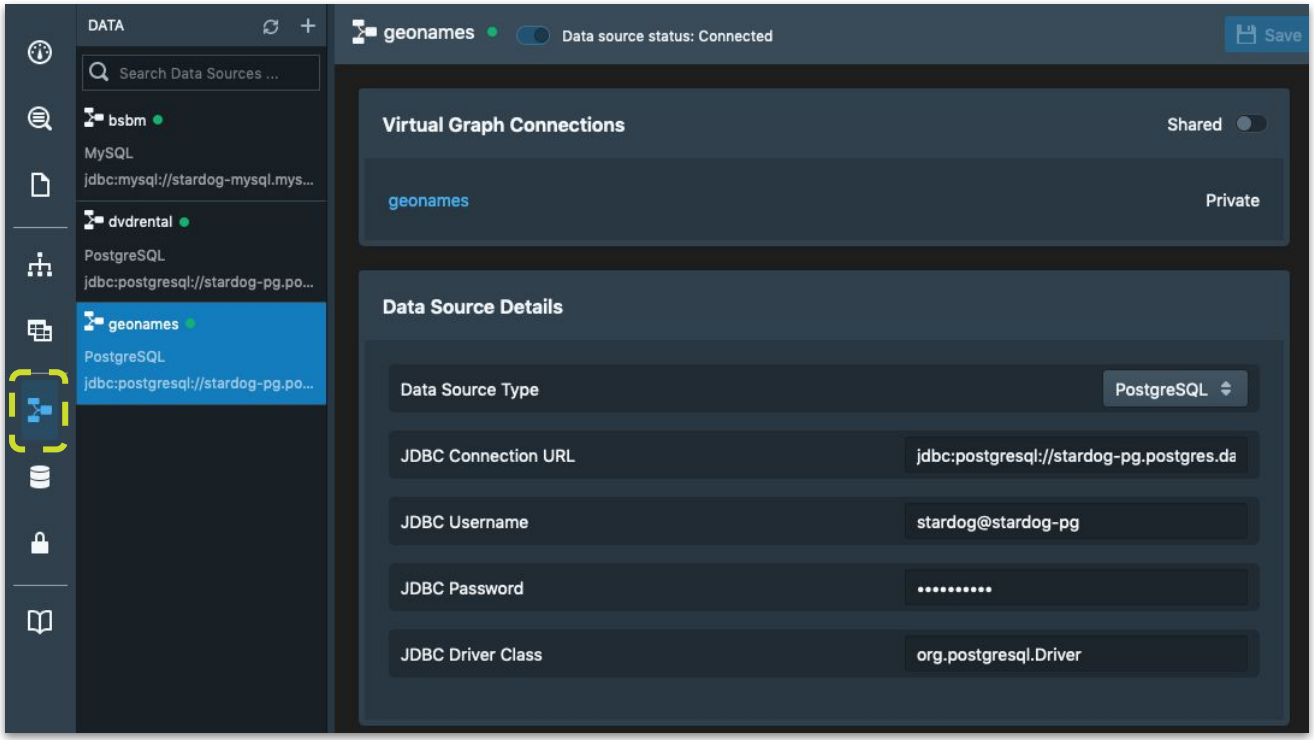


Data Sources

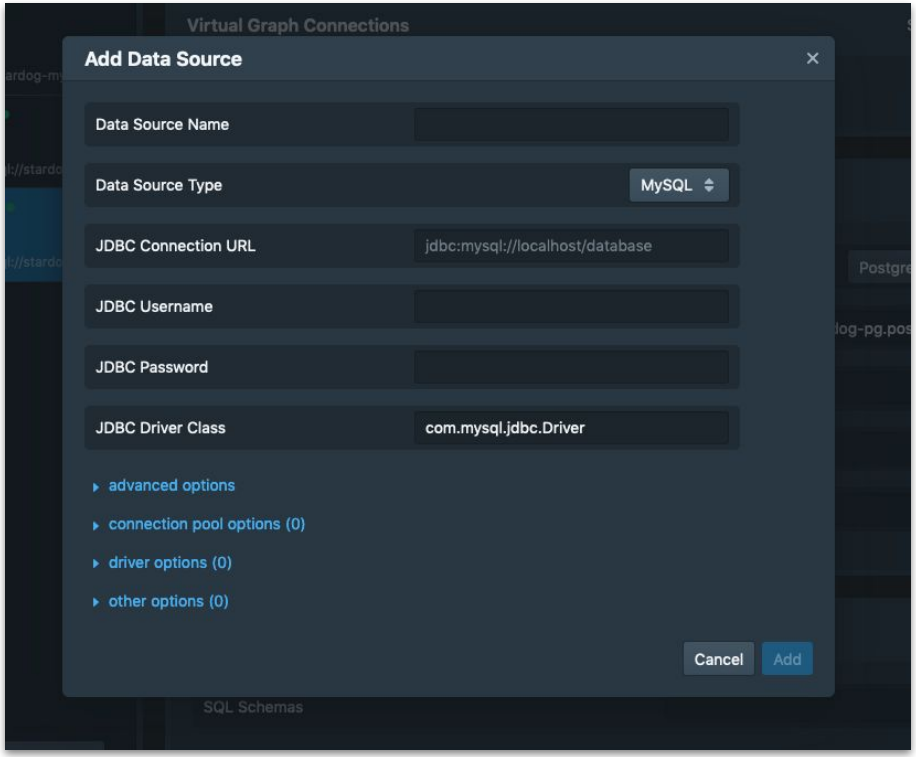
Data Sources

- Database connection parameters used by Virtual Graphs
- Can be implicitly created by adding a Virtual Graph or explicitly managed as a data source resource
- Any data source Stardog cannot connect to on startup is marked offline and can be administratively brought online
- “data-source” command line or Stardog Studio administration

Data Sources in Stardog Studio



Add Data Source



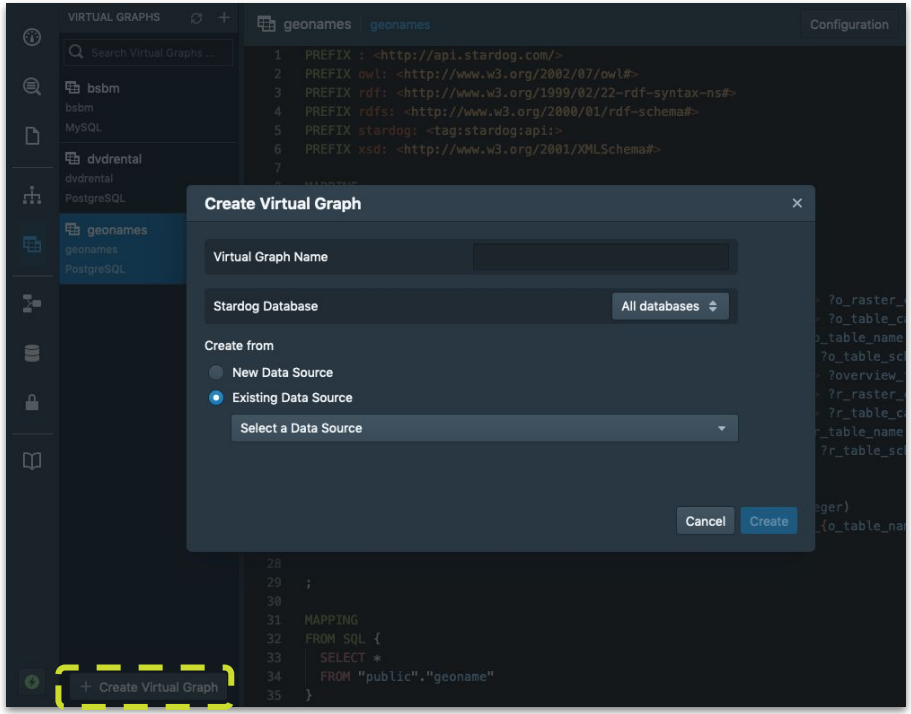
Mappings

Virtual Graphs

- Map structured data declaratively into RDF
- Support for RDBMS, CSV, MongoDB, Cassandra
- Mapping is conceptual, so you can:
  - Extract, transform, load (ETL) into RDF
  - Query on the fly
    - Rewrite SPARQL queries to the source language (SQL, CQL, etc.)

Create and Edit Mappings in Studio

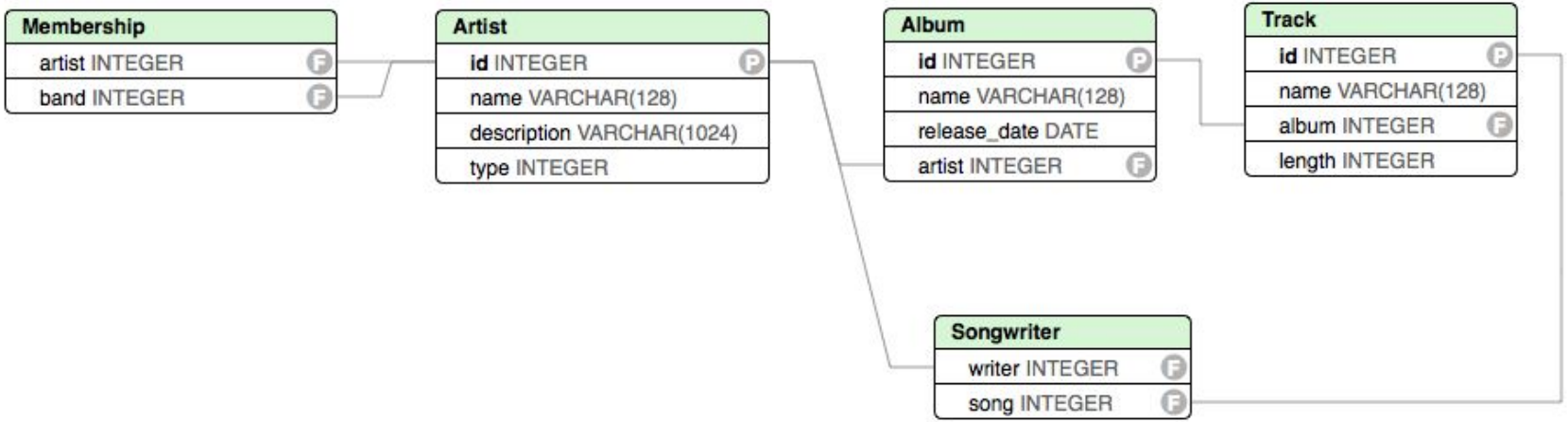
- Create a Virtual Graph with with data source
- Generate mappings
- Edit mapping (various syntaxes)



R2RML: Mapping from RDB to RDF

- R2RML is a standard for mapping RDB sources to RDF
- Provides two kinds of mappings:
  - Direct mappings
    - Auto-generated from relational schema
  - Customizable mappings
    - Structure and vocabulary chosen by the author

Relational Schema



Mappings

name	length	id	album
Love Me Do	125	1	1

Track

song	writer
1	1
1	2

Songwriter

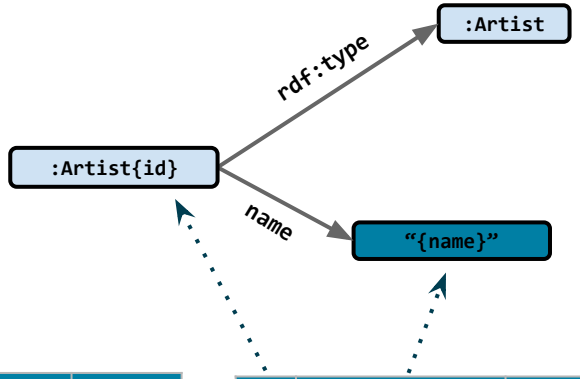
id	name	release_date	artist
1	Please Please Me	1963-03-22	5
2	McCartney	1970-04-17	2
3	Imagine	1971-10-11	1

Album

id	name	type
1	John Lennon	1
2	Paul McCartney	1
5	The Beatles	2

Artist

Mappings: Artist



name	length	id	album
Love Me Do	125	1	1

Track

song	writer
1	1
1	2

Songwriter

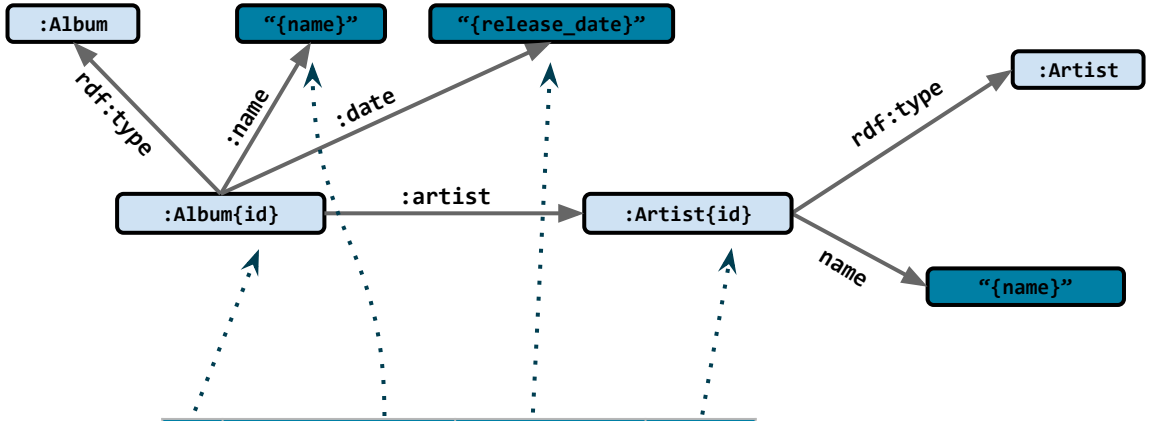
id	name	release_date	artist
1	Please Please Me	1963-03-22	5
2	McCartney	1970-04-17	2
3	Imagine	1971-10-11	1

Album

id	name	type
1	John Lennon	1
2	Paul McCartney	1
5	The Beatles	2

Artist

Mappings: Album



name	length	id	album
Love Me Do	125	1	1

Track

song	writer
1	1
1	2

Songwriter

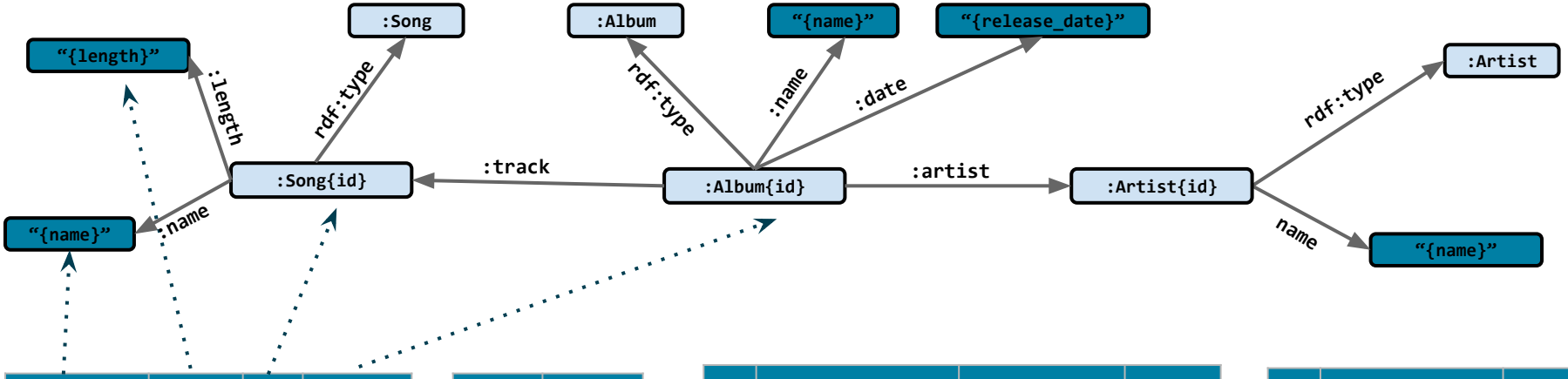
id	name	release_date	artist
1	Please Please Me	1963-03-22	5
2	McCartney	1970-04-17	2
3	Imagine	1971-10-11	1

Album

id	name	type
1	John Lennon	1
2	Paul McCartney	1
5	The Beatles	2

Artist

Mappings: Song



name	length	id	album
Love Me Do	125	1	1

Track

song	writer
1	1
1	2

Songwriter

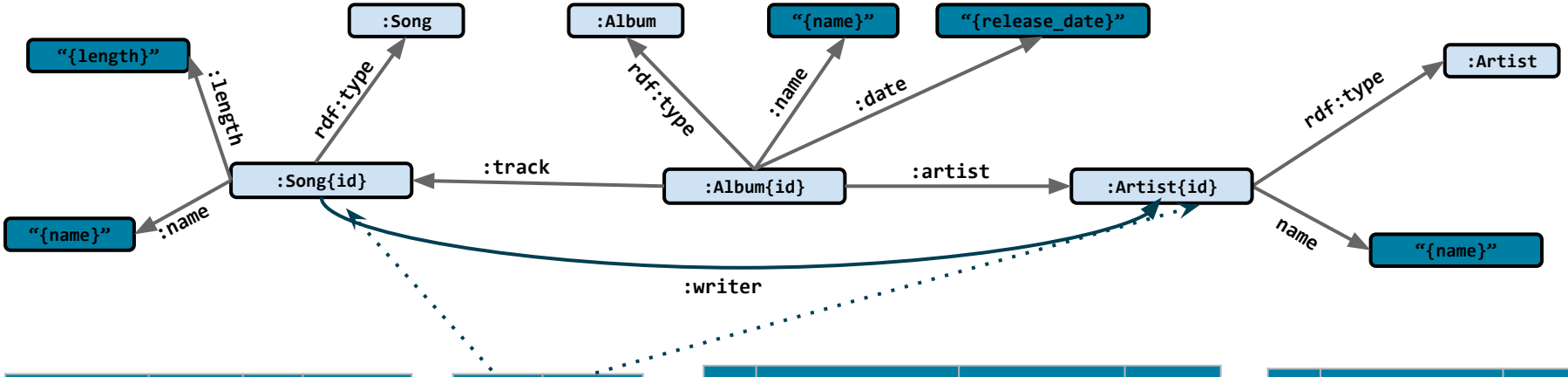
id	name	release_date	artist
1	Please Please Me	1963-03-22	5
2	McCartney	1970-04-17	2
3	Imagine	1971-10-11	1

Album

id	name	type
1	John Lennon	1
2	Paul McCartney	1
5	The Beatles	2

Artist

Mappings: Songwriter



name	length	id	album
Love Me Do	125	1	1

Track

song	writer
1	1
1	2

Songwriter

id	name	release_date	artist
1	Please Please Me	1963-03-22	5
2	McCartney	1970-04-17	2
3	Imagine	1971-10-11	1

Album

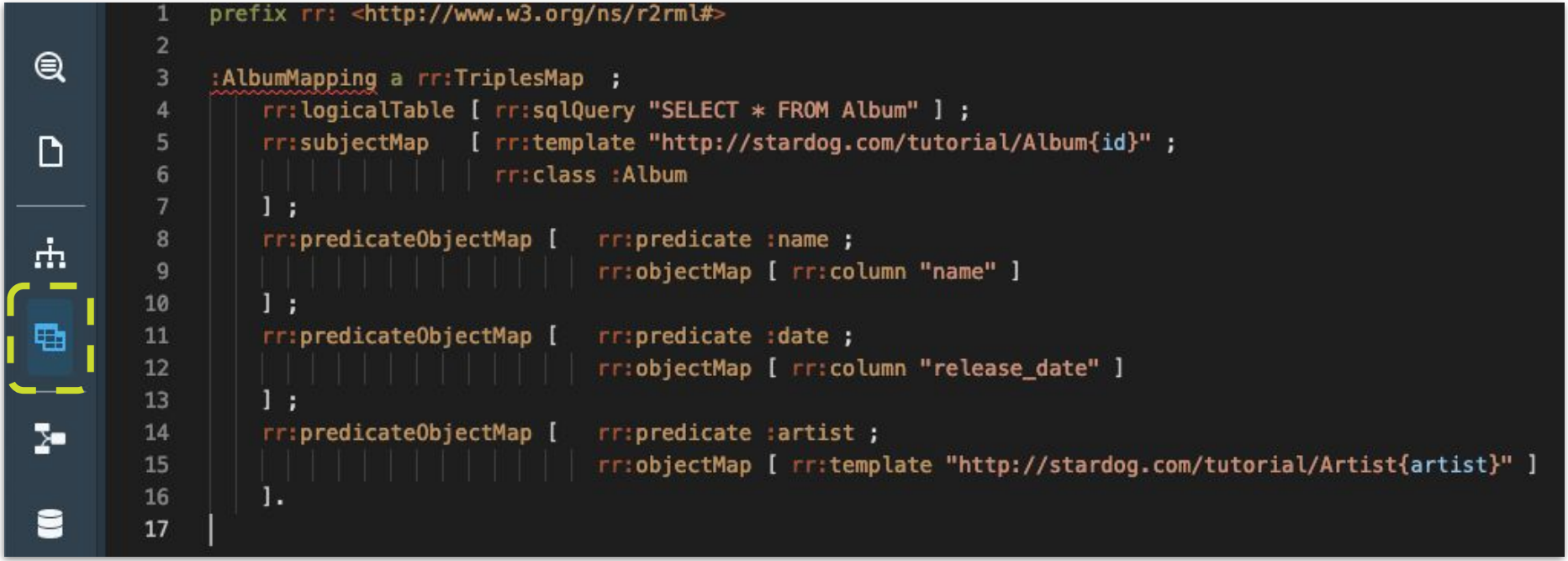
id	name	type
1	John Lennon	1
2	Paul McCartney	1
5	The Beatles	2

Artist

R2RML

- A **triples map** is a mapping from one row of a relational table (or a SQL query) to RDF triples
- A **subject map** defines the subject that is shared by all the triples generated from that triples map
- Triples are generated based on **predicate object maps**

R2RML Syntax



```
prefix rr: <http://www.w3.org/ns/r2rml#>

:AlbumMapping a rr:TriplesMap  ;
    rr:logicalTable [ rr:sqlQuery "SELECT * FROM Album" ] ;
    rr:subjectMap   [ rr:template "http://stardog.com/tutorial/Album{id}" ;
                      rr:class :Album
    ] ;
    rr:predicateObjectMap [   rr:predicate :name ;
                              rr:objectMap [ rr:column "name" ]
    ] ;
    rr:predicateObjectMap [   rr:predicate :date ;
                              rr:objectMap [ rr:column "release_date" ]
    ] ;
    rr:predicateObjectMap [   rr:predicate :artist ;
                              rr:objectMap [ rr:template "http://stardog.com/tutorial/Artist{artist}" ]
    ].

```

Example Mapping: SMS

```
MAPPING :AlbumMapping
FROM SQL {
  SELECT * FROM Album
}
TO {
  ?album a :Album ;
         :name ?name ;
         :artist ?artist ;
         :date ?release_date
}
WHERE {
  BIND(template("http://stardog.com/tutorial/Album{id}") AS ?album)
  BIND(template("http://stardog.com/tutorial/Artist{artist}") AS ?artist)
}
```

Example Mapping: Name

Optional Name

```
MAPPING :AlbumMapping
FROM SQL {
  SELECT * FROM Album
}
TO {
  ?album a :Album ;
         :name ?name ;
         :artist ?artist ;
         :date ?release_date
}
WHERE {
  BIND(template("http://stardog.com/tutorial/Album{id}") AS ?album)
  BIND(template("http://stardog.com/tutorial/Artist{artist}") AS ?artist)
}
```

Example Mapping: Input

Input Table

```
MAPPING :AlbumMapping
FROM SQL {
  SELECT * FROM Album
}
TO {
  ?album a :Album ;
         :name ?name ;
         :artist ?artist ;
         :date ?release_date
}
WHERE {
  BIND(template("http://stardog.com/tutorial/Album{id}") AS ?album)
  BIND(template("http://stardog.com/tutorial/Artist{artist}") AS ?artist)
}
```

Example Mapping: Output

Output Graph

```
MAPPING :AlbumMapping
FROM SQL {
  SELECT * FROM Album
}
TO {
  ?album a :Album ;
         :name ?name ;
         :artist ?artist ;
         :date ?release_date
}
WHERE {
  BIND(template("http://stardog.com/tutorial/Album{id}") AS ?album)
  BIND(template("http://stardog.com/tutorial/Artist{artist}") AS ?artist)
}
```

Example Mapping: Values

```
MAPPING :AlbumMapping
FROM SQL {
  SELECT * FROM Album
}
TO {
  ?album a :Album ;
         :name ?name ;
         :artist ?artist ;
         :date ?release_date
}
WHERE {
  BIND(template("http://stardog.com/tutorial/Album{id}") AS ?album)
  BIND(template("http://stardog.com/tutorial/Artist{artist}") AS ?artist)
}
```

Value Mappings

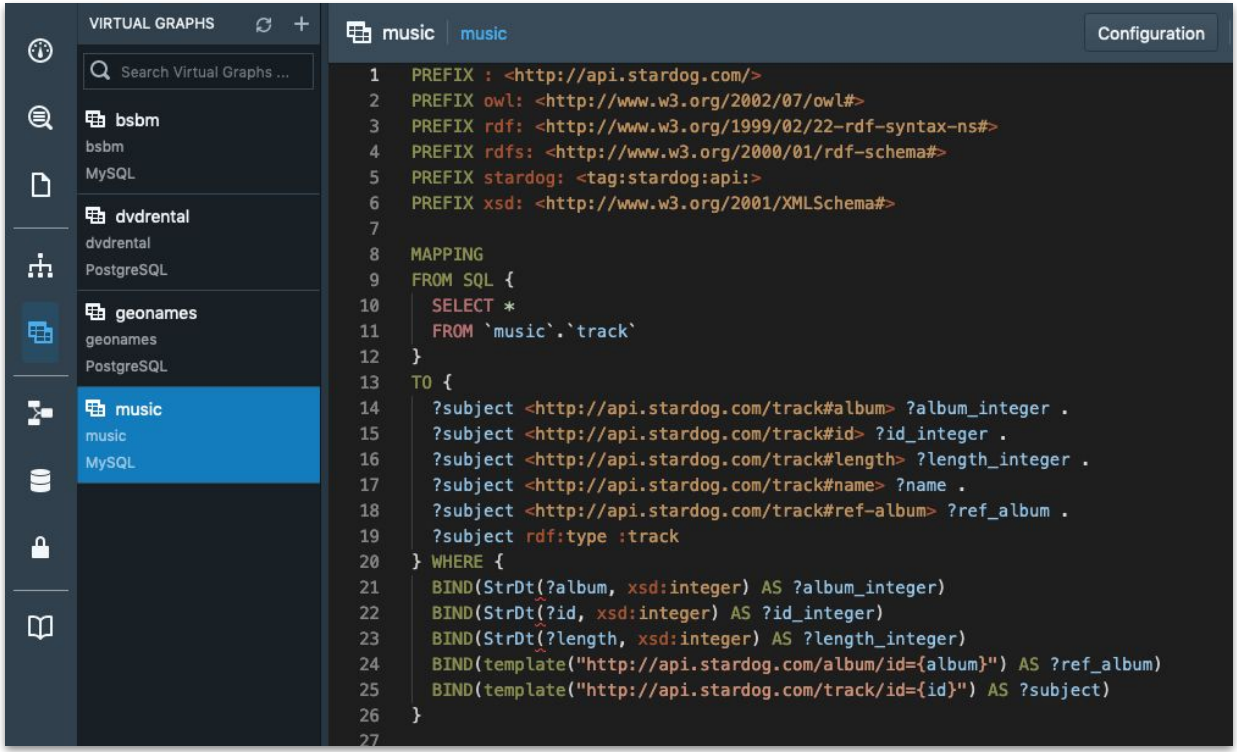
Example Mapping: NoSQL Source

Different FROM

```
prefix : <http://example.com/>

MAPPING <urn:accounts>
FROM JSON {
  "accounts" : {
    "_id" : "?id",
    "acct" : "?acctNum",
    "customerName" : [ "?name" ],
    "card" : [ {
      "number" : "?ccNumber",
      "expiration" : "?ccExpr" }
    ]
  }
}
TO {
  ?holder :hasAcct ?acct .
  ?holder :hasName ?name .
}
WHERE {
  BIND (template("http://example.com/acct/{acctNum}") AS ?acct)
  BIND (template("http://example.com/holder/{ccNumber}_{name}") AS ?holder)
```

Generated Virtual Graphs in Studio



Using Virtual Graphs

Virtual Graphs in Stardog

- A Virtual Graph has three components:
  - A unique name
    - Conforms to regex [A-Za-z]{1}[A-Za-z0-9_-]
  - Configuration options for the data source
    - Primarily JDBC connection parameters
  - Mappings for the data
    - Mappings in R2RML or Stardog Mapping Syntax

Adding Virtual Graphs

Register a Virtual Graph (implicit data source)

```
$ stardog-admin virtual add music.properties music_mappings_r2rml.ttl
```

Example properties file

```
jdbc.url=jdbc\:mysql\://localhost/music
jdbc.username=admin
jdbc.password=admin
jdbc.driver=com.mysql.jdbc.Driver
```

Managing Virtual Graphs

Remove a Virtual Graph

```
$ stardog-admin virtual remove music
```

List registered graphs

```
$ stardog-admin virtual list
```

Export mappings or properties

```
$ stardog-admin virtual mappings music
```

```
$ stardog-admin virtual options music
```

Querying Virtual Graphs

- IRI of a Virtual Graph is `virtual://vgName`
- Query it as any other named graph

```
SELECT ?member ?name {
  GRAPH <virtual://music> {
     ?band a :Band ;
           :name "The Beatles" ;
           :member ?member .
     ?member :name ?name .
  }
}
```

Querying Virtual Graphs

- Combine local queries with Virtual Graphs

```
SELECT ?member ?name ?birthDate {
  GRAPH <virtual://music> {
     ?band a :Band ;
           :name "The Beatles" ;
           :member ?member .
     ?member :name ?name .
  }
  ?member :birthDate ?birthDate
}
```

Virtual Transparency

- Changes query context to include Virtual Graphs, avoiding the need to specify the URI, e.g. `GRAPH <virtual://`
- Set `virtual.transparency` to true
- Special URIS
  - tag:stardog:api:context:virtual - refers to all Virtual Graphs
  - tag:stardog:api:context:all - union of all Virtual Graphs with local graphs
- Disable on a per query basis
  - Query hint: `#pragma virtual.transparency off`

Named Graph Alias

- Feature enables creation of graph URI to represent a set of named graphs and Virtual Graphs
- Enabled with the database property `graph.aliases`
- Can be used in SPARQL Protocol, FROM, FROM NAMED, USING and USING NAMED
- Adding and updating graph aliases is done with data
  - Special named graph: `<tag:stardog:api:graph:aliases>`
  - Special predicate: `<tag:stardog:api:graph:aliases>`
- Some limitations, e.g. `GRAPH keywords`, `CONSTRUCT`, `INSERT` or `UPDATE`

Materialize Virtual Graphs

Import the contents of Virtual Graph

```
$ stardog-admin virtual import myDb dept.properties dept.ttl
```

Import the contents of Virtual Graph into a named graph

```
$ stardog-admin virtual import -g http://example.com/targetGraph myDb dept.properties dept.ttl
```

Materialize via SPARQL update queries

```
ADD <virtual://dept> TO <http://example.com/targetGraph>  # adds all triples to existing triples
```

```
COPY <virtual://dept> TO <http://example.com/targetGraph>  # removes existing triples first
```

Import CSV Files

- CSV files represent tabular data
- Use mappings to convert CSV content into RDF
- Only imports supported

```
$ stardog-admin virtual import myDb cars.ttl cars.csv
```

Virtual Graph Cache

Virtual Graph Cache

- Cached datasets can be run in conjunction with a Stardog Server to serve a variety of purposes
- Dataset can be an entire graph, Virtual Graph, or a query result (currently experimental)
- Cache datasets can be created that contain a portion of a Virtual Graph that does not update frequently, while allowing federated queries to remain fully virtualized
- Cache targets can be fully managed out of band and populated by 3rd party tools and integrations

Cache Commands

- Create a cached graph containing the contents of a Virtual Graph (assuming node1 is a registered cache target)

  `$ stardog-admin cache create cache://virtual-name --graph virtual://name --target node1`
- Create a cached graph but do not load the contents

  `$ stardog-admin cache create cache://virtual-name --graph virtual://name --target node1 --register-only`
- Create a cached query containing the results of a query on database 'db' *(experimental)*

  `$ stardog-admin cache create cache://query_file --query query_file.rq --target node1 --database db`
- Create a cached query with a custom refresh script *(experimental)*

  `$ stardog-admin cache create cache://query_file --query query_file.rq --target node1 --database db --refresh-script refresh_file.ru`

Learning Objectives

Learning Objectives

- Defining and configuring a Virtual Graph
- Options available for virtual mapping to data sources
- How Virtual Graphs interact with other features

Thank you

STARDOG ACADEMY