# Modeling

**Building a data model for your Enterprise Knowledge Graph**

Taught by:

Paraskevi Zerva
Solutions Architect

# Learning Objectives

- How to build a data model, using Stardog Studio, to support your Enterprise Knowledge Graph
- Fundamentals of data models and data modeling as a process
- Basics of existing types of models including ontology models
- The process of successful ontology model development and associated best practices
- Foundational concepts of ontology models, illustrated through examples, including an overview of existing modeling standards and language representations

# Modeling

# Role of (Data) Modeling / Data Models

- Data modeling is the process used to define & analyze data requirements within the scope of information systems
  - Modeling is also used as a technique for detailing business requirements for specific databases (database modeling)
- A data model organizes elements of data and standardizes how they relate to one another and to the properties of real-world entities
  - It explicitly determines the structure of data, provides specific definition
  - It provides a way to formally represent data structures or a set of concepts such as entities, attributes and relations

# Types of Models & Ontology Models

- Three perspectives of data model instances exist at different phases of the modeling process:
  - Conceptual: describes the “what” of a domain by scoping down key concepts & rules
  - Logical: describes “how” key concepts and entities should be modeled & provides a technical map/model of rules and data structures
  - Physical: describes the physical means & structure by which data are stored
- Different types of models exist:
  - In relational world database model (schema) forms a blueprint on how a db is constructed
  - In a semantic/ knowledge graph world an ontology (model) defines the set of concepts and relationships that represent the content and structure of a domain, while also provides a strong focus on formal semantics

# Ontology Core Concepts

- An ontology defines a set of representational primitives with which to model a knowledge including:
  - classes (or sets/entities),
  - attributes (relations between a class member and text data e.g., date, string, boolean)
  - relationships (relations among class members)
  - information about their meaning (axioms and rules) and,
  - constraints (modeling tool mainly used for validation)
- In the context of knowledge graphs (KGs) an ontology model forms a flexible schema designed to highlight relationships between concepts (graph data structure)
- Ontologies are specified in languages that allow abstraction away from data structures closer in expressive power to first-order logic therefore operating at "semantic level"
- Stardog is based on the RDF open standard created to represent large-scale information systems.

# W3C Ontology Language Standards

RDF - Instance data

RDFS - Class/property hierarchy

OWL - Expressive definitions

SWRL - Rules with functions

SHACL - SHACL Shapes Constraint Language

# Model Development Process Steps

# How to Start the Development of a Model

- No single right way to model a domain
- Capture data requirements (dq) that emerge from business questions (bq) that need to be answered
- Build your model iteratively by identifying as part of the dq(s) the entities, attributes and relationships critical to your bq(s)
- Follow simple ontology development guidelines
- Adopt usual (software) engineering practices

# Example

- Business Question: Identify percentage of sales of most popular musical albums and music genres & categorize by date.
- Data Requirements:

  Entities: Album, Music Genre

  Attributes: date, sales_percentage

  Relationships : has_music_genre

# Ontology Development Steps

I. Determine the scope of the ontology (data requirements)
II. Identify the core classes (critical entities)
III. Build the class hierarchy
IV. Specify properties (relationships and attributes or else object and data type properties per W3C standard terminology)

# Ontology Scope (I)

- As minimal ontological commitment as possible (MVP)
- Over-specification hinders extensibility
- Generate competency questions
  - Ontology must be able to answer these questions
  - Determine if ontology contains enough information by defining the required entities and property relationships

# Core Classes (II)

- Specify core classes (representing entities), properties & individuals
- Helps setting the scope and structure
- Reuse terms (if possible)
  - Make sure what is the value you are getting from vocabulary reuse
  - Keep it lightweight (make sure it does not affect performance)
- Determine types of terms

# Reuse Terms & Standard Vocabularies

- RDFS - rdfs:label, rdfs:comment
- FOAF - foaf:Person, foaf:Organization, … (describing people)
- Dublin Core - dc:title, dc:author, dc:date, ..(metadata about digital docs)
- Standard vocabularies/ontologies
  - SKOS (taxonomies), schema.org, GoodRelations, FIBO (financial services domain)
- Be aware of existing ontology standards and reuse these consciously based on what you are trying to achieve within your data model domain

# Types of Terms

## Class vs. Relationship (Object Property)

a. Class `Person` to denote any person
b. Relationship (Object Property) `memberOf` to denote that Person has a “memberOf” relationship to a music band

## Class vs. Individual (or Instance)

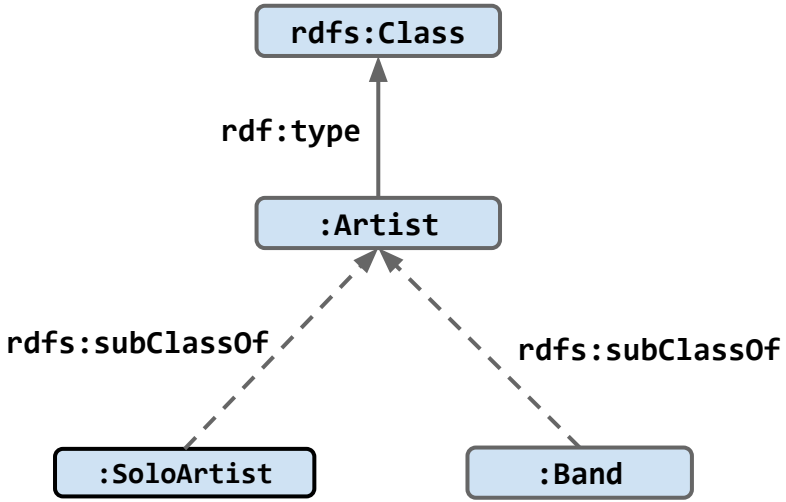
a. `Jazz_Music` as a subclass of `Music_Genre` (class hierarchy)
b. `Jazz_Music` as an instance of `Music_Genre` (type of thing)

# Class Hierarchy (III)

- Backbone of the ontology
- Important for understandability and reasoning (inheritance)
- Subsumption relation is the inheritance of properties from the parent (subsuming) class to the child (subsumed) class
- Anything true of a parent class is also true of all its child classes
- 2 options : a class is allowed to have only one parent (*single inheritance*), have a number of parents (*multiple inheritance*)
- Be principled but balance it with pragmatics (follow common-sense rules)
  - Avoid deep philosophical debates - be practical

# Class Hierarchy Example

- SoloArtist *is a :subclassOf of* Artist class
- Band *is a :subclassOf of* Artist class



# Class Hierarchy Guidelines

- Add classes as needed
  - Subclasses should have different characteristics than the super class
- Intermediate classes
  - Single subclass: incompleteness or redundancy
  - Too many subclasses: not having enough categories
- Consistent modeling
  - Homogenous siblings (subclasses) in the hierarchy
  - Part-whole relations vs. subclasses (composition vs inheritance)

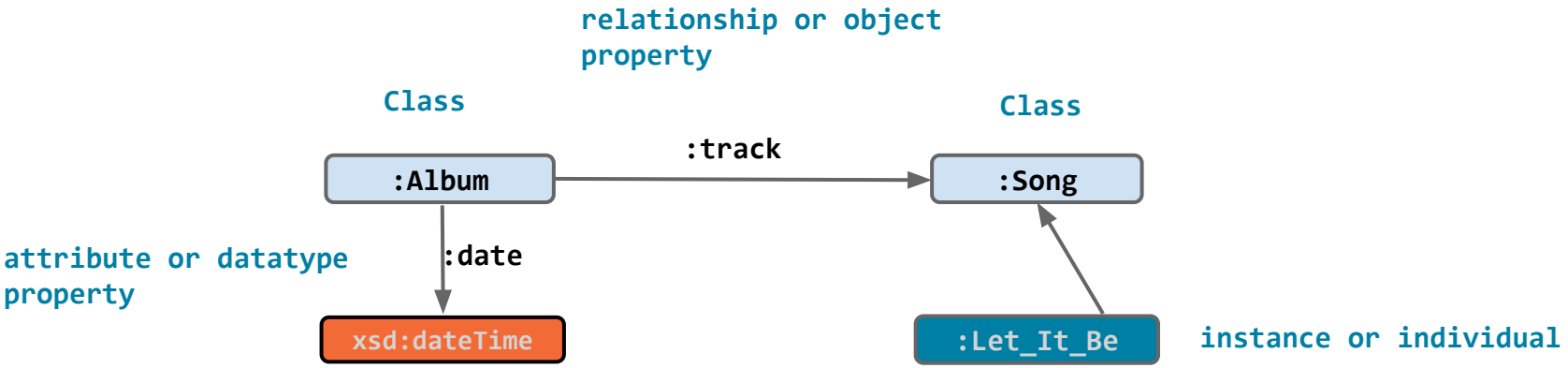
# Specify Properties (IV)

- Which class should/can/must be used by which property
- Object (relationships) & data type properties (attributes).
- Domain and range of properties
- Global definitions (`rdfs:domain`, `rdfs:range`) - apply to all instances of the property
- Local definitions (`owl:someValuesFrom`, `owl:allValuesFrom`) - constrain the range of a property in specific contexts by local restrictions to its class
- SHACL constraints for close-world database schema & data validation

# Glossary

- Classes refer to sets, collections, concepts, types of objects, kinds of things
- Properties refer to all types of object relationships or datatype properties (attributes)
- Relationships refer to object properties (based on OWL/RDF standard terminology)
- Attributes refer to datatype properties (based on OWL/RDF standard terminology)
- Individuals may also be called instances or objects (the basic or "ground level" objects)
- In terms of graph representation diagrams, Classes & Individuals correspond to nodes while properties correspond to edges
- Classes & Individuals could be used in the position of subject or object in the RDF tuple notation, while properties correspond to predicates

# Diagram Representation



# Ontology Concepts through Example

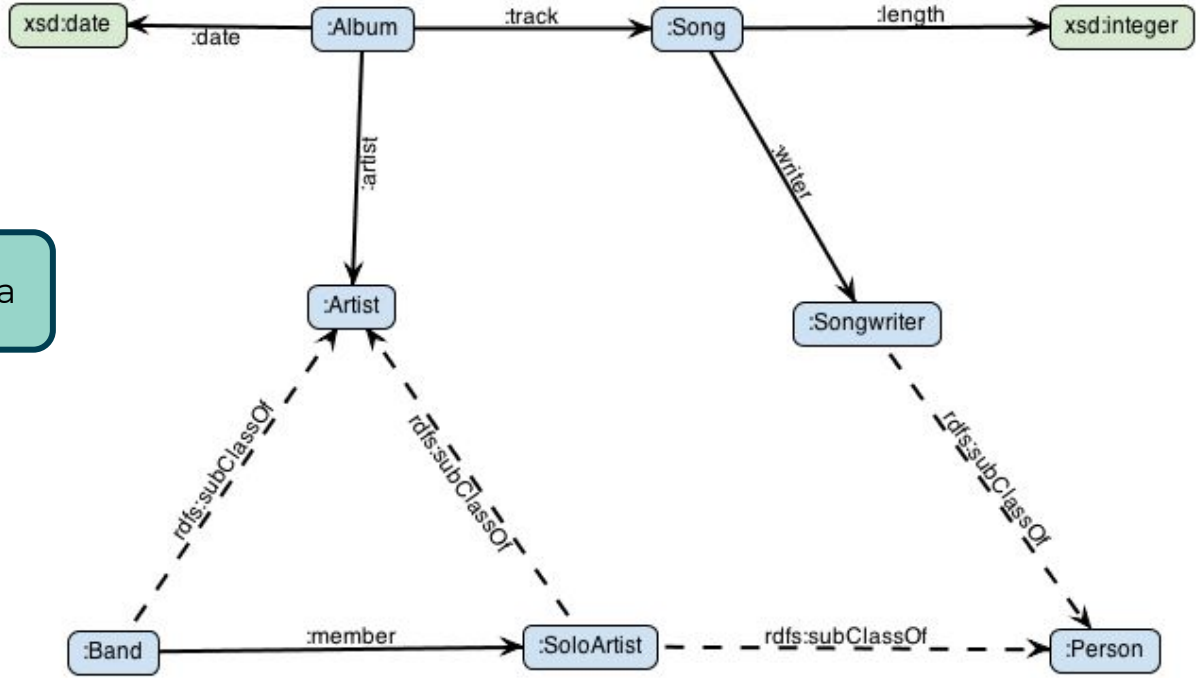
STARDOG ACADEMY

# Basic Modeling

- Start with 3-5 key concepts & their relationships
- A little modeling goes a long way
  - Over-specification hinders reuse
- Layer your models
  - Start simple
  - Extend and refine definitions as you go
- Naming should reflect the real world
  - Model will be your lingua franca for data
  - Should match what domain experts say/think

# Key Concepts for Music Data Example

RDF schema



# Declarations

## RDFS

```
:Album a rdfs:Class .

:track a rdf:Property .

:date a rdf:Property .
```

## OWL

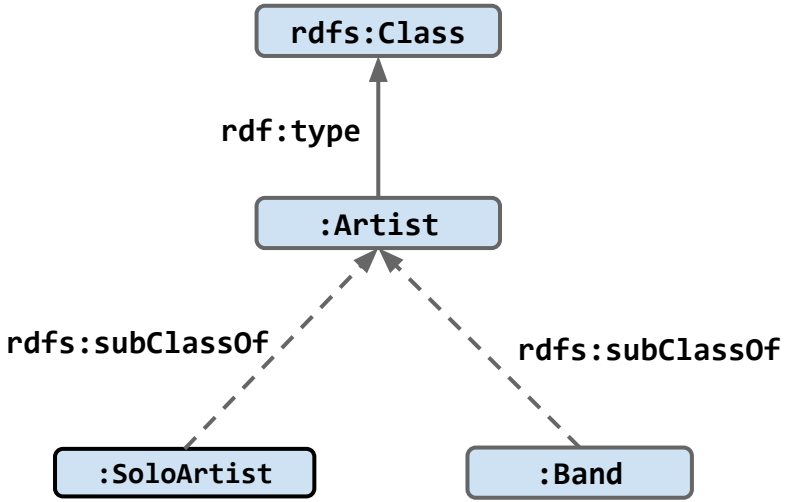
```
:Album a owl:Class .

:track a owl:ObjectProperty .

:date a owl:DatatypeProperty .
```

# Class (rdfs:Class)

- Represents a set of individuals (instances) or entities with common characteristics
- Can be organized in a hierarchy using rdfs:subClassOf
- Instances can be classified automatically under the hierarchy based on their properties



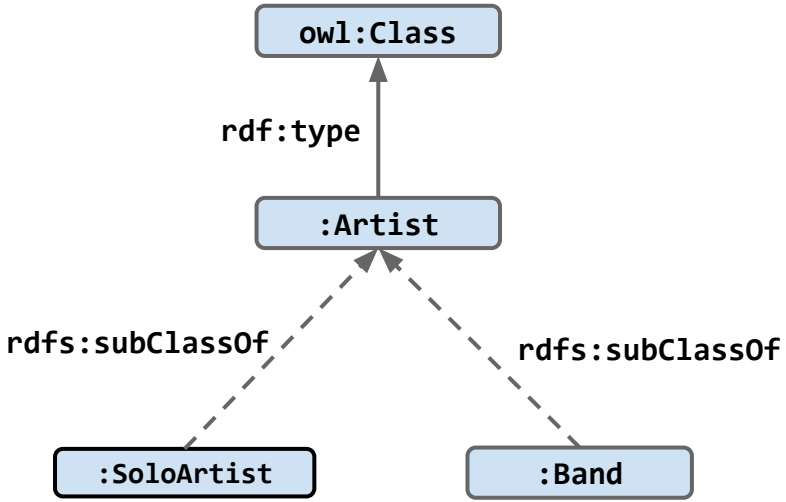
# Class (owl:Class or owl:Thing)

Alternatively the main class could be defined as of:

rdf:type owl:Class or

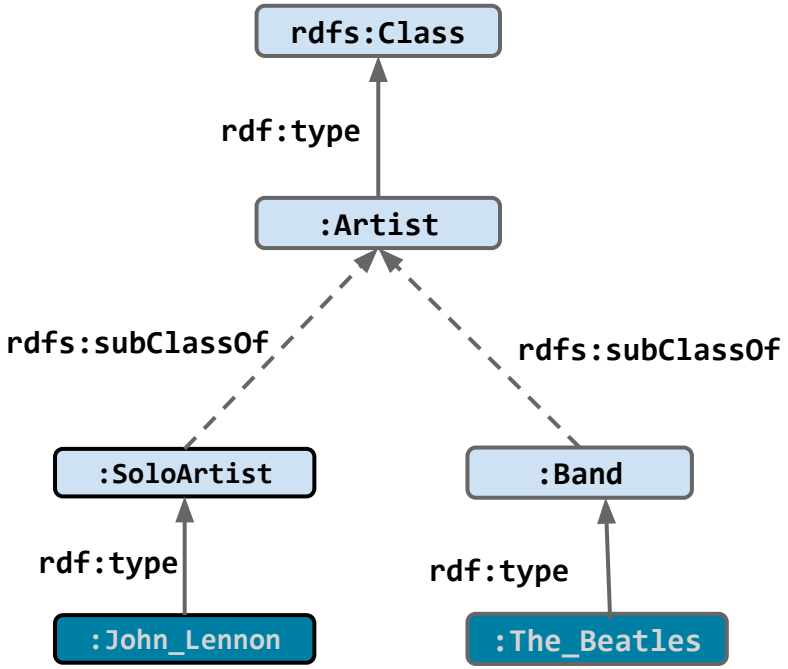
rdf:type owl:Thing

in the OWL language notation



# Individual vs. Class

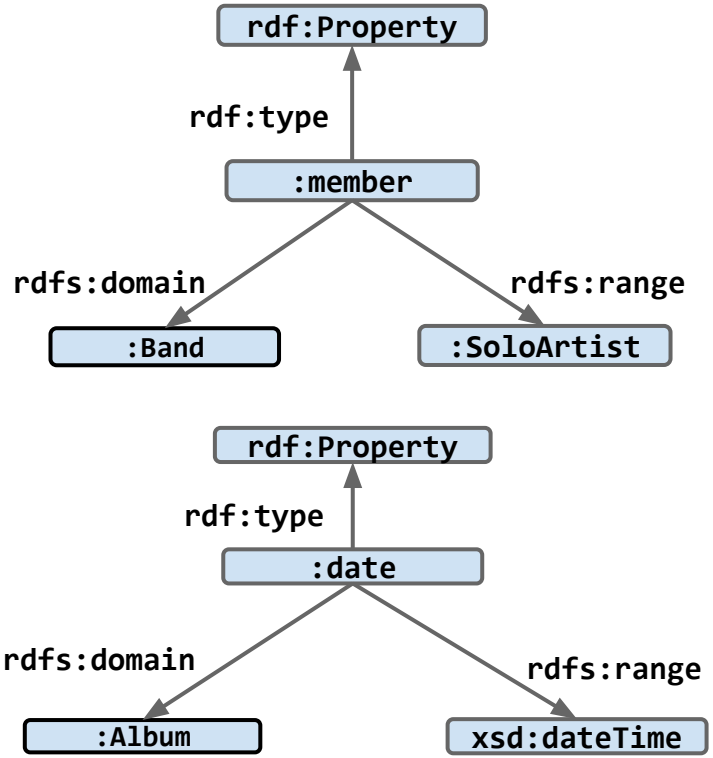
- Class represents a set individuals with common characteristics
- John_Lennon *is an instance of* the Solo Artist class
- The_Beatles *is an instance of* the Band class




It can take time to build the skill to know what to make a class and what to make an instance.

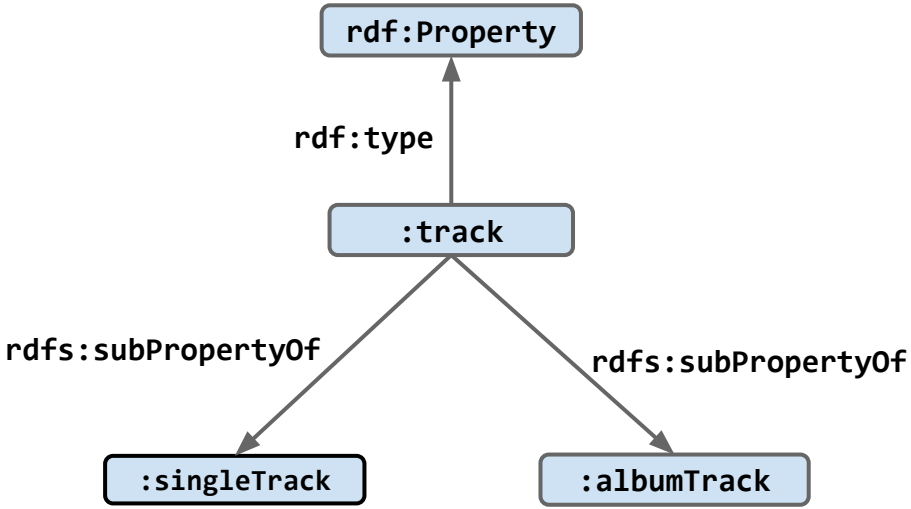
# Property (1/2)

- Represents relationships
  - Object property defines relationships between individuals.
    Individual ➔ Individual
  - Data property defines relationships between an individual and a literal.
    Individual ➔ literal (e.g., string, boolean)



# Property (2/2)

- Can be organized in a hierarchy as well through rdfs:subPropertyOf



# Correlate Relational and Ontology Models

- A lot of the data come from the relational world but how do the two worlds correlate?
- A **table** generally represents a **class** and a **row** in a table is an **instance** of a class.
- The **properties** correspond to **column names** and **cell values** are the **nodes**, literals and individuals, that the node is connected to.
- If you want to know more on how to turn relational data to a (knowledge) graph and represent with an ontology model check the training on **Virtual graphs**.

# Naming Conventions

- For classes, the local name starts with a capital letter

  ```
  :Album
  ```

- For properties (object or datatype), the local name starts with lower-case letter

  ```
  :track
  ```

# Ontology Concepts Notation

- `rdf:type` (to state that a data resource is an instance of a class)
- `rdfs:Class or owl:Class` (both are supported by Stardog ; consistency is recommended)
- `rdfs:Property, owl:ObjectProperty, owl:DatatypeProperty` (to state properties)
- `rdfs:subClassOf, rdfs:subPropertyOf` (to state class and property hierarchies)

# Practical Considerations

# Practical Considerations

- Naming scheme
- Ontology modularity
- Ontology Layering
- Ontology versioning
- Testing and debugging

# Naming Scheme

- Any scheme works as long as it is consistent
  - Capitalization and delimiters

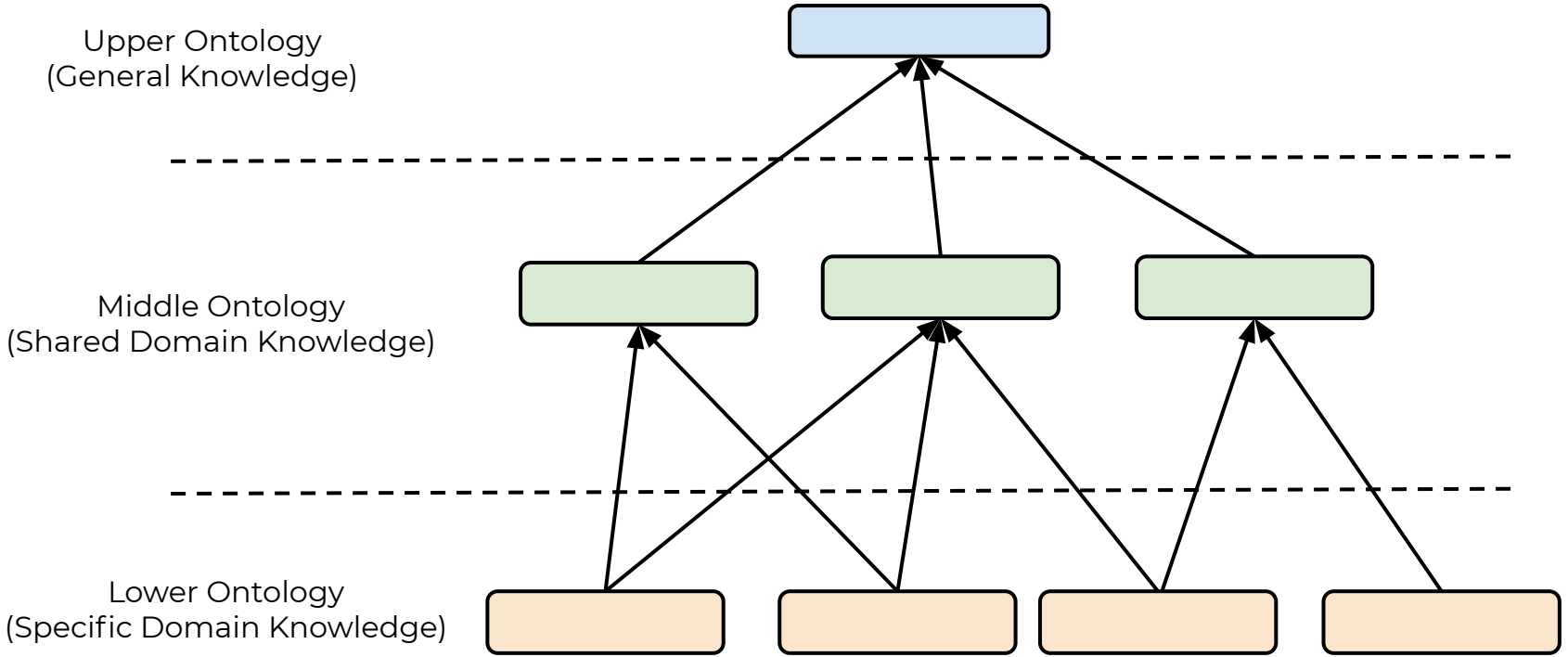
    member, hasMember, has_member
  - Suffix or prefix conventions

    isMemberOf, memberOf, hasMember
  - Singular vs plural
- Use annotations instead of overloading names
  - Annotations improve clarity, avoid misuse

# Modularity

- Single monolithic ontology is hard to maintain
- Divide domain into submodules
- Use separate ontologies [and namespaces]
- Reuse and share terms between ontologies

# Ontology Layering



# Ontology Versioning

- Ontologies should be considered as software artifacts
- Use a version control system
- Ontologies should be semantically versioned
- Version ID should be structured and meaningful
  - Can be changed according to well-understood rules
  - SemVer http://semver.org/

# SemVer Basics

- Version ID is in the form X.Y.Z
- X, Y, Z fields are integers
- X is major, Y is minor, Z is patch
- Additional labels can be appended to the version ID as needed

# SemVer Principle

Given version X.Y.Z, increment the:

- Major version X when you make incompatible changes,
- Minor version Y when you add functionality in a backwards-compatible manner, and
- Patch version Z when you make backwards-compatible bug fixes

# Versioning in OWL

- Each ontology has an ontology IRI
- Optionally an ontology might have a version IRI

```
<http://www.example.com/ont> owl:versionIRI <http://www.example.com/ont/2.0>
```

- Ontology IRI should be unique if there is no version IRI
- Ontology IRI and version IRI pair should be unique

# Testing Ontologies

- Check for logical errors
  - Unsatisfiable classes
  - Inconsistent instances
- Inspect inferences
  - Detect unintentional inferences
- Develop unit tests (treat ontology models as programs)
  - SPARQL queries / expected results (expected schema behavior)
  - SHACL (shapes definition to make tests declarative)

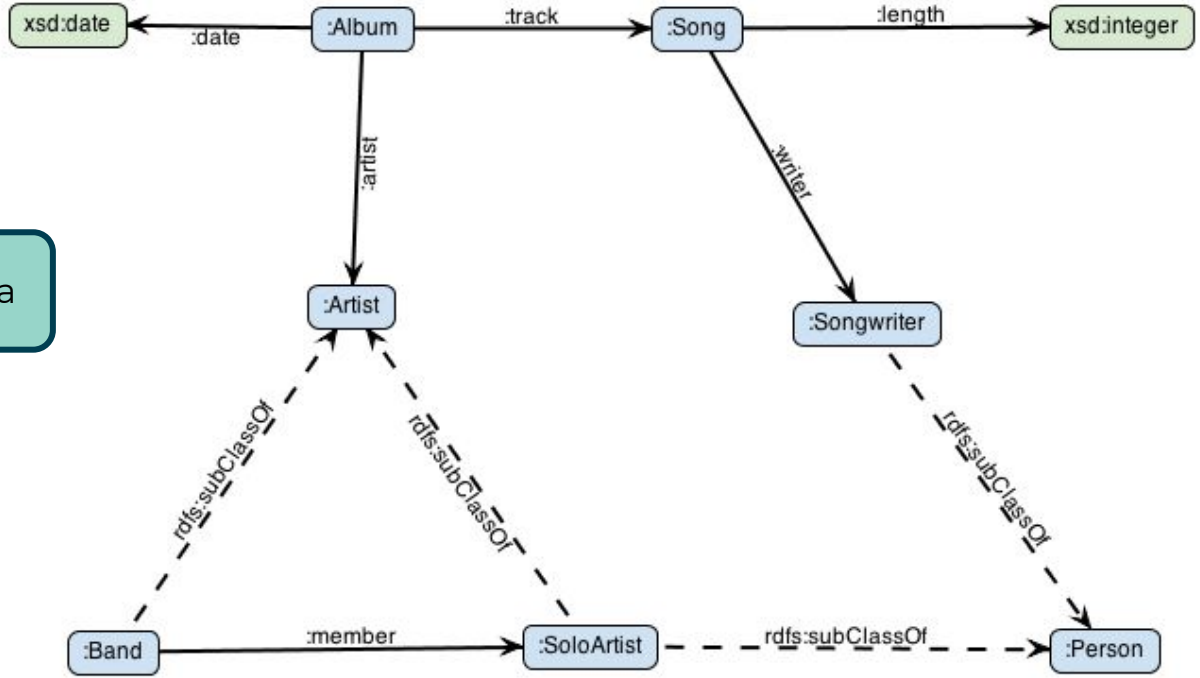
# Ontology Model Development Tools

# Ontology Development Tools/Editors

- Stardog Studio Models Editor (primary modeling tool)
- Protégé: A free open-source ontology editor and framework (Stardog reads protege models)
- Next we are going to use Stardog Studio Models Editor to walk you through the fundamentals of building a music ontology model using the Music Data Example

# Key Concepts for Music Data Example

RDF schema



# Stardog Studio Models Demo

STARDOG ACADEMY

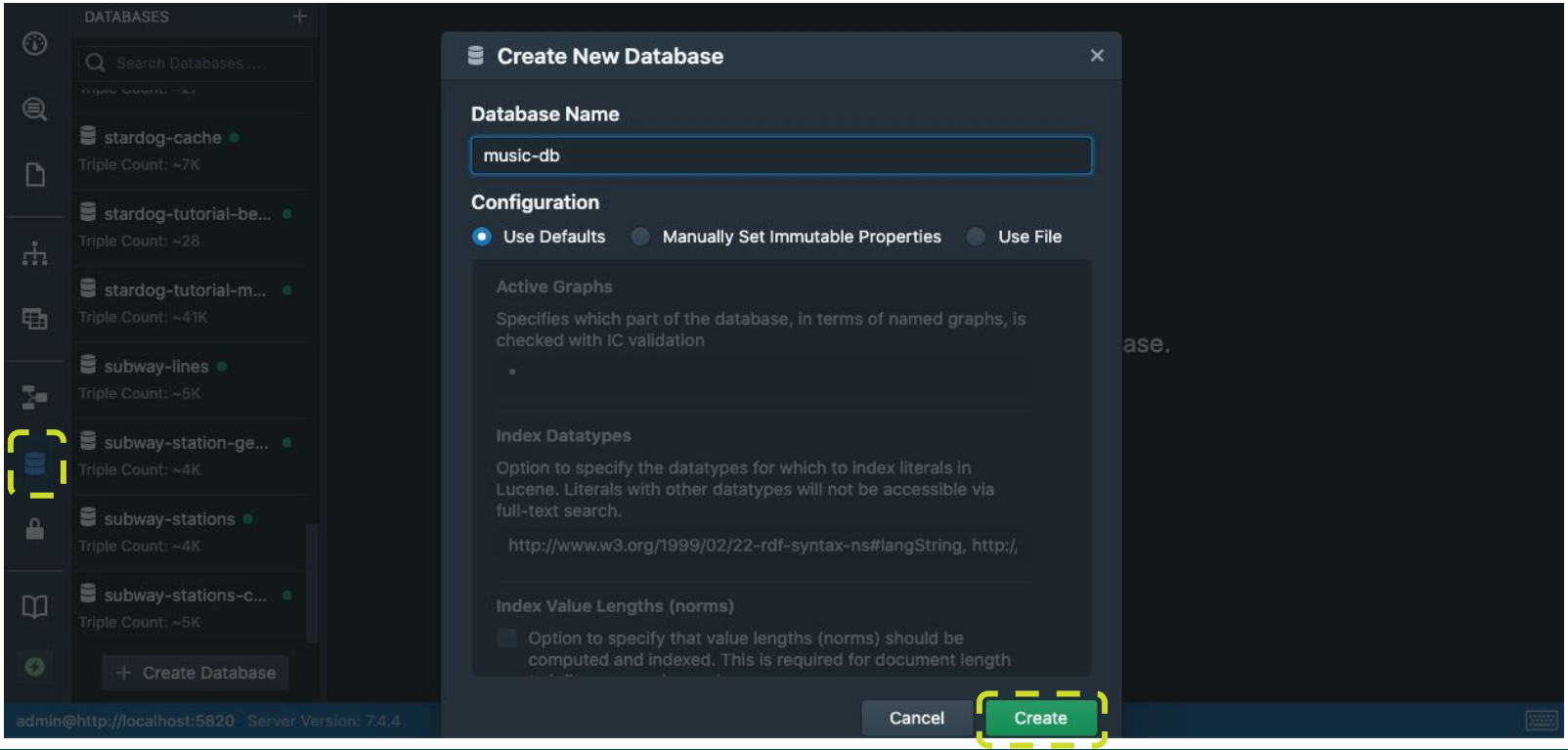
# Step 0: Identify Data Requirements & Key Concepts to Model

Looking on the data we realize that the critical entities to model are:

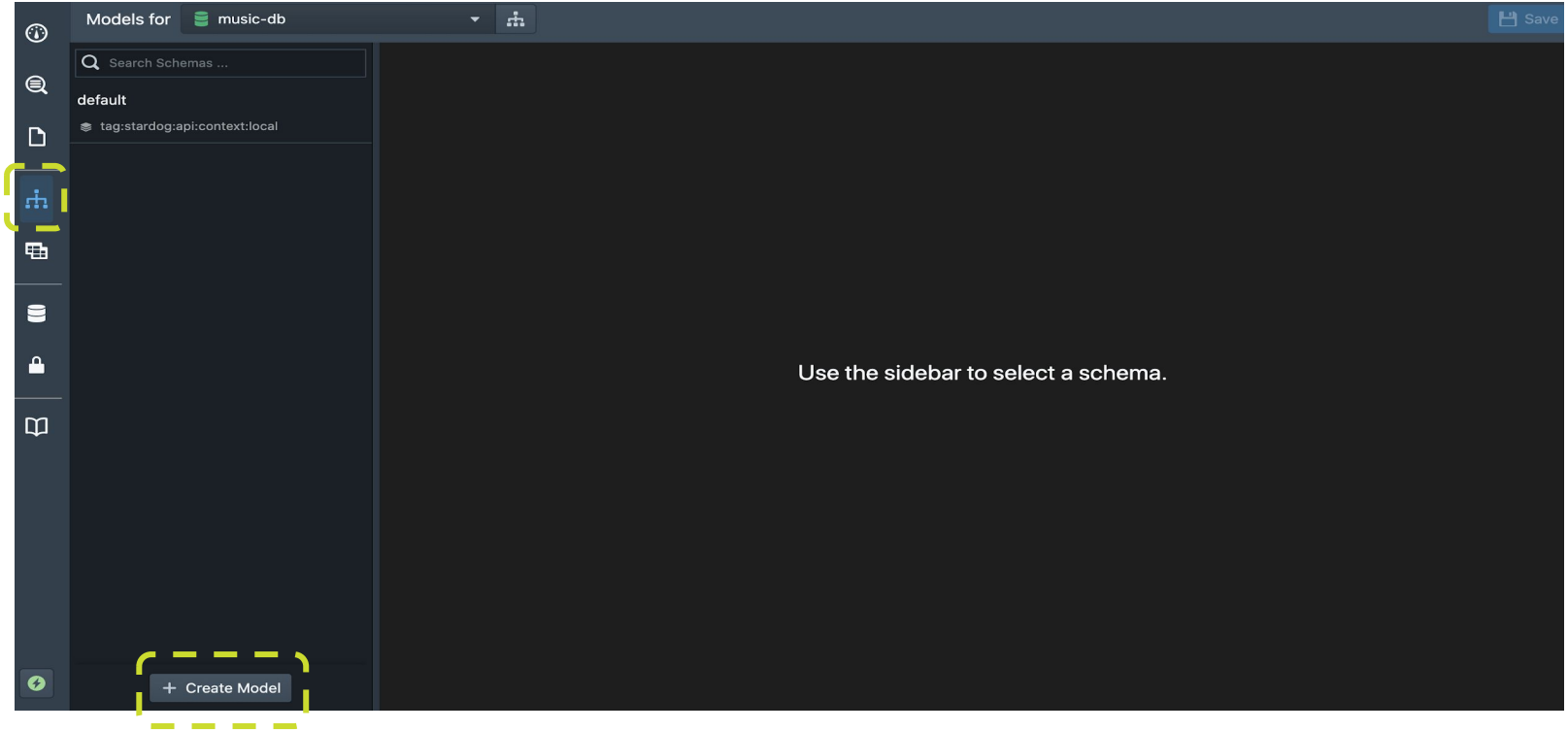
**Classes:** Person, Artist, SoloArtist, Band, Album, Song, Songwriter

**Properties:** Member of Band , track & date of Album, length and writer of Song

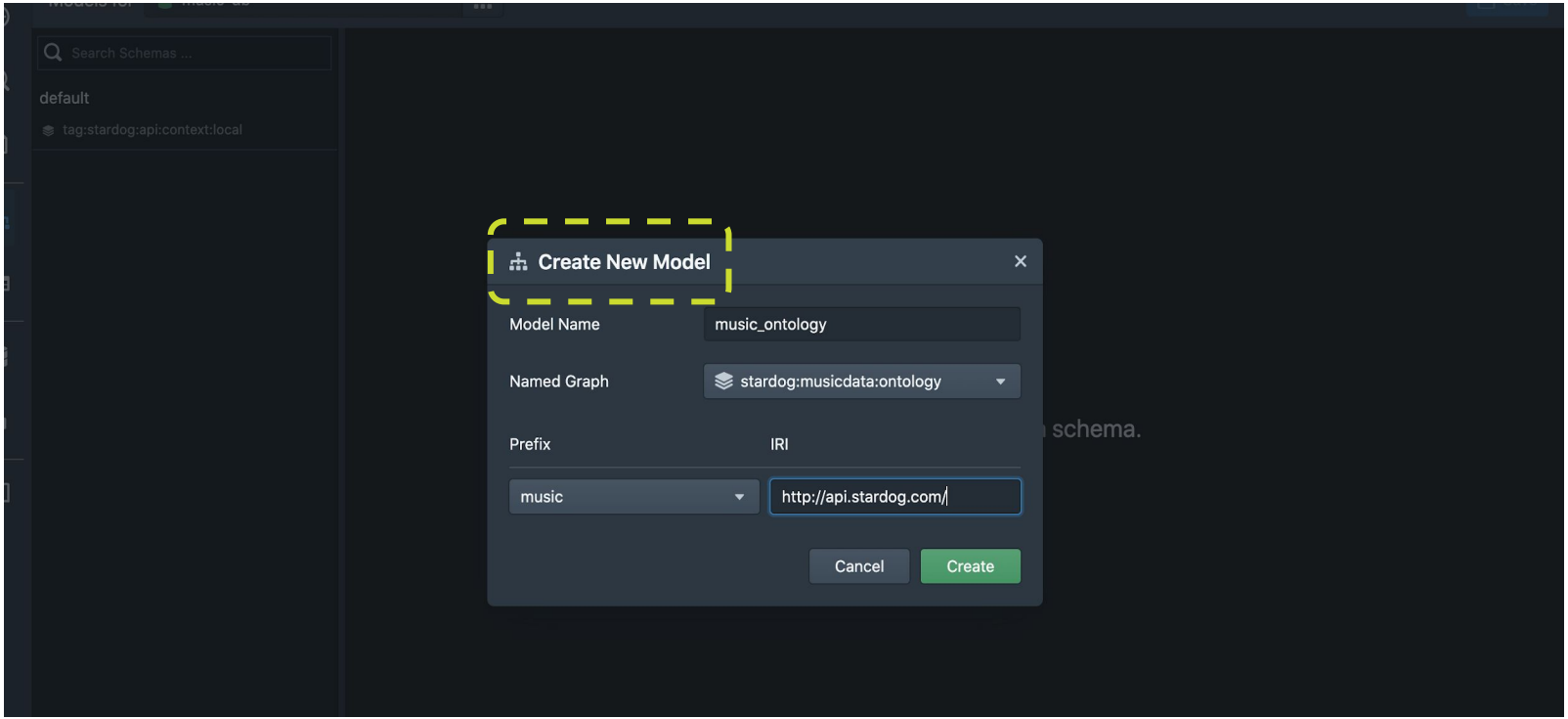
# Step 1: Create a music database in Studio



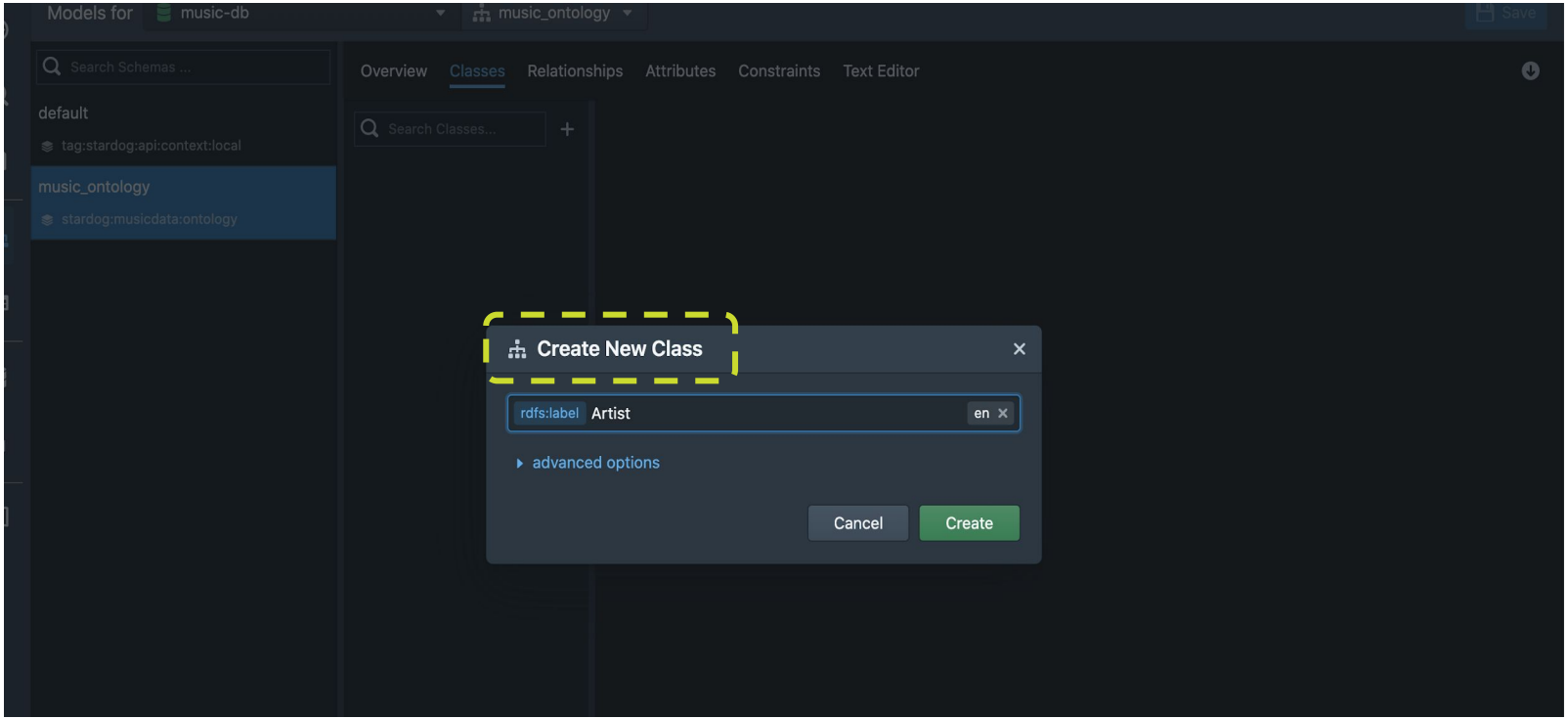
# Step 2: Create a model through Models Editor in Studio



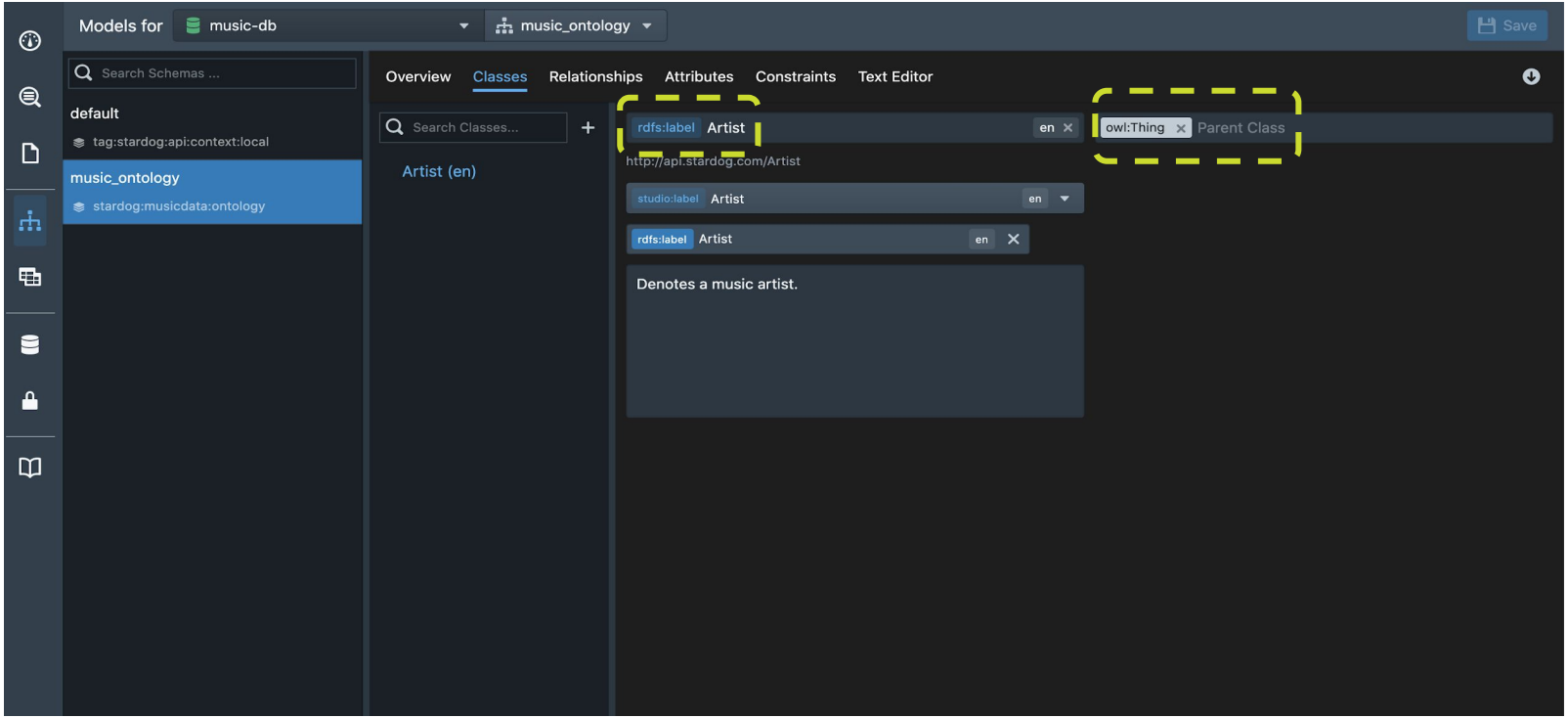
# Step 2: Create a Model through Models Editor in Studio



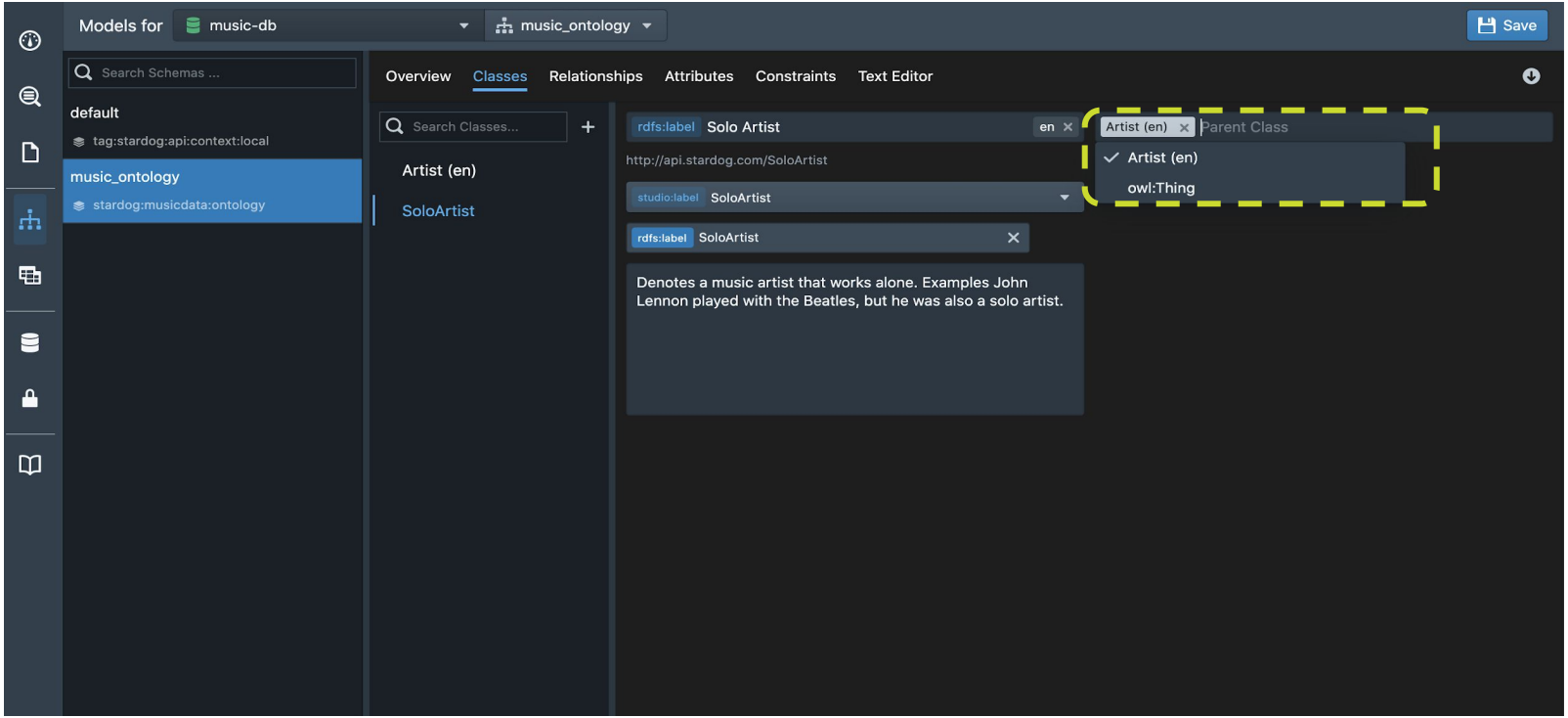
# Step 3: Add Key Classes



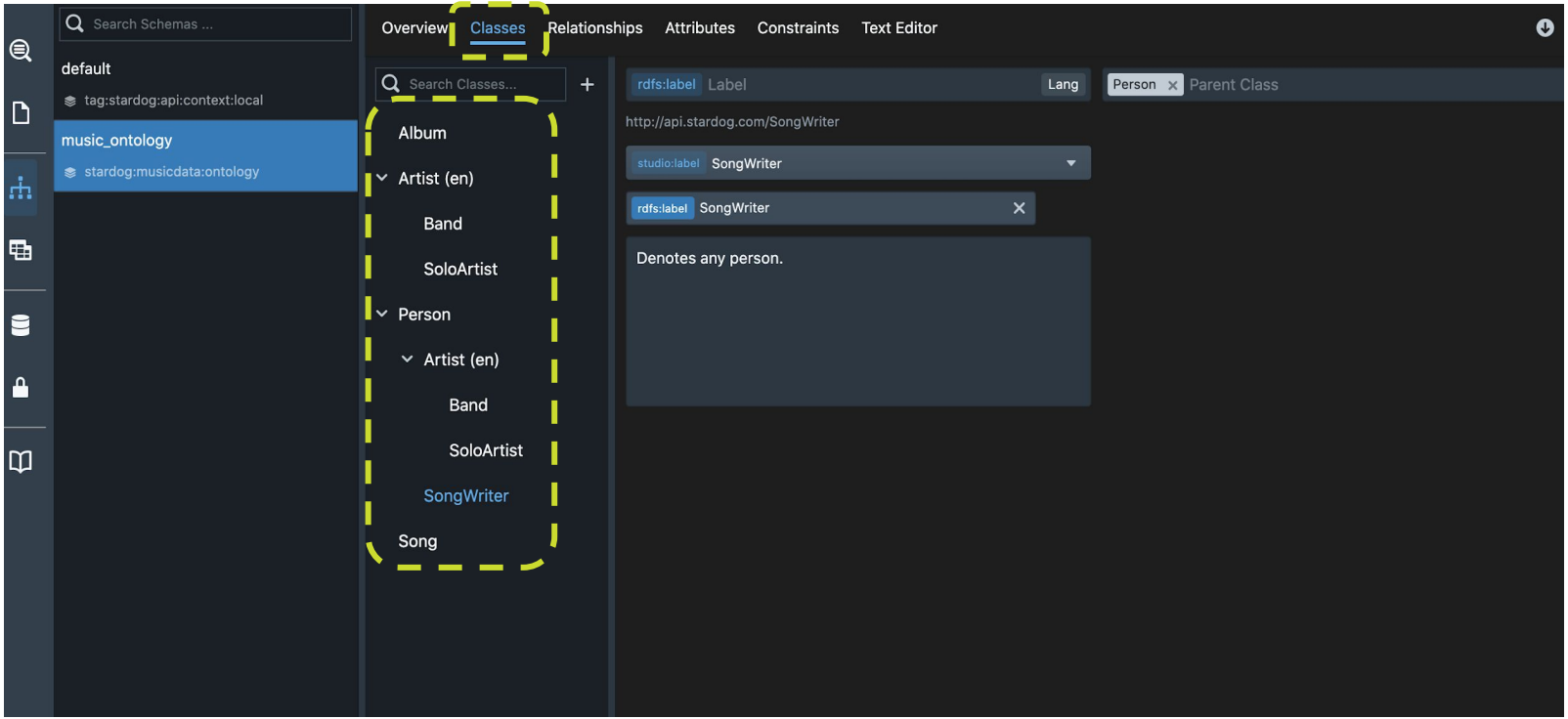
# Step 3: Add Key Classes



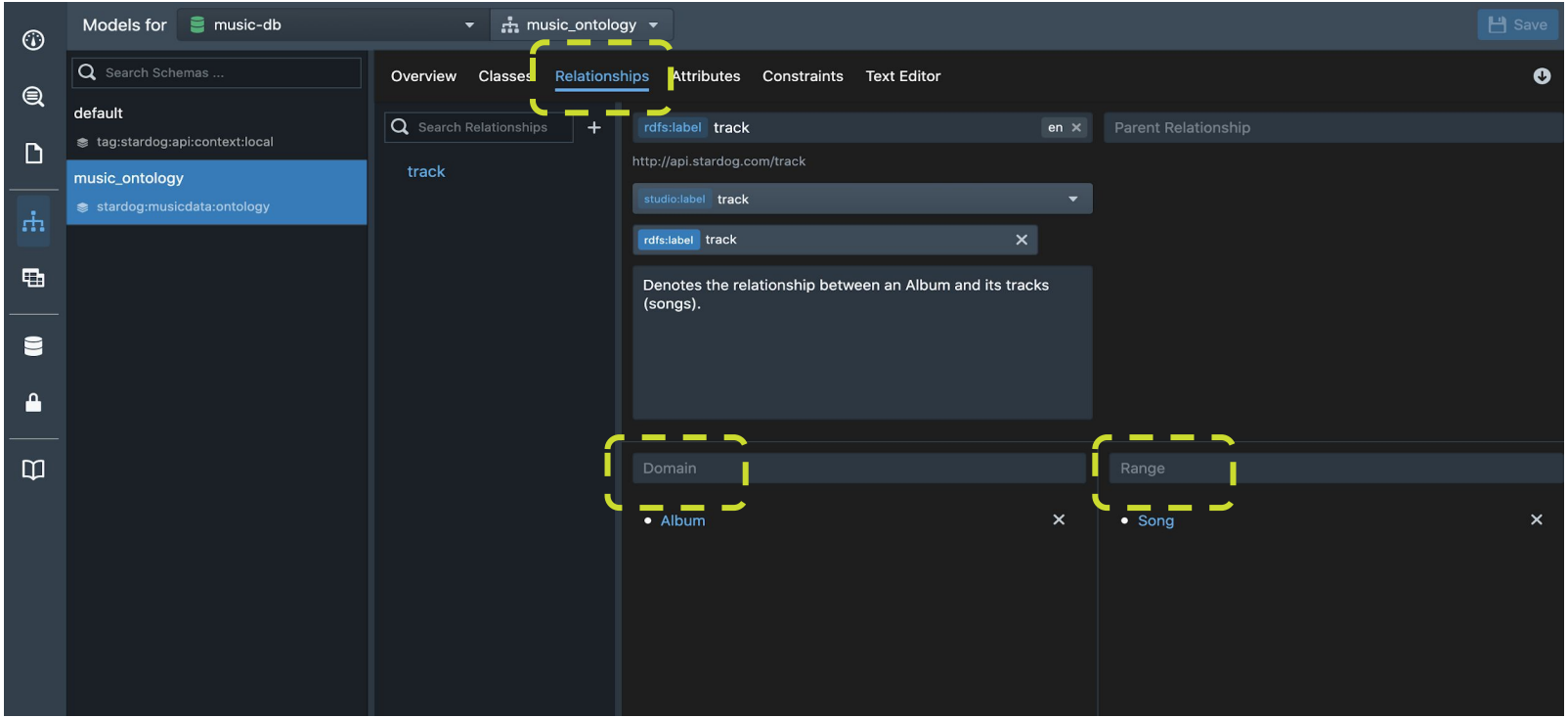
# Step 3: Add Key Classes



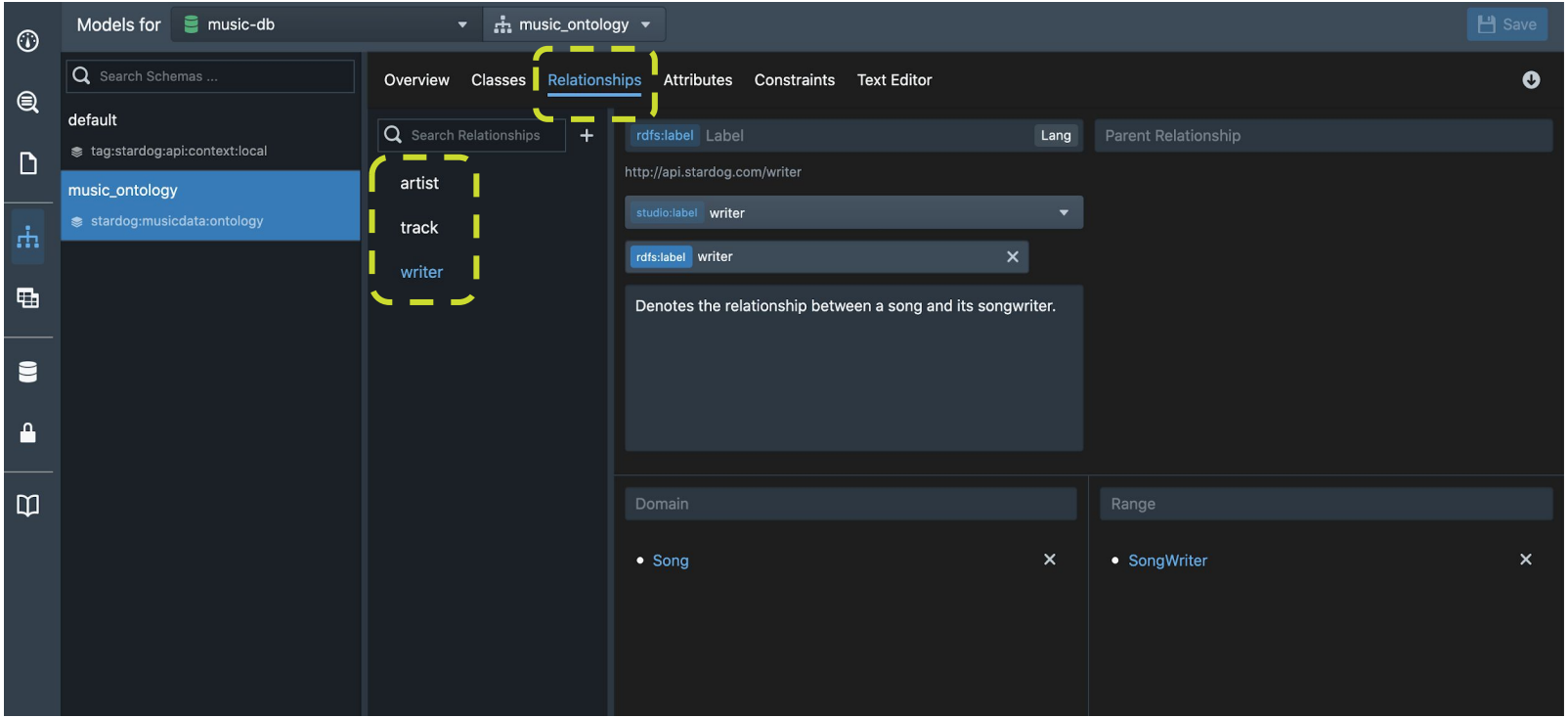
# Step 3: Add Key Classes



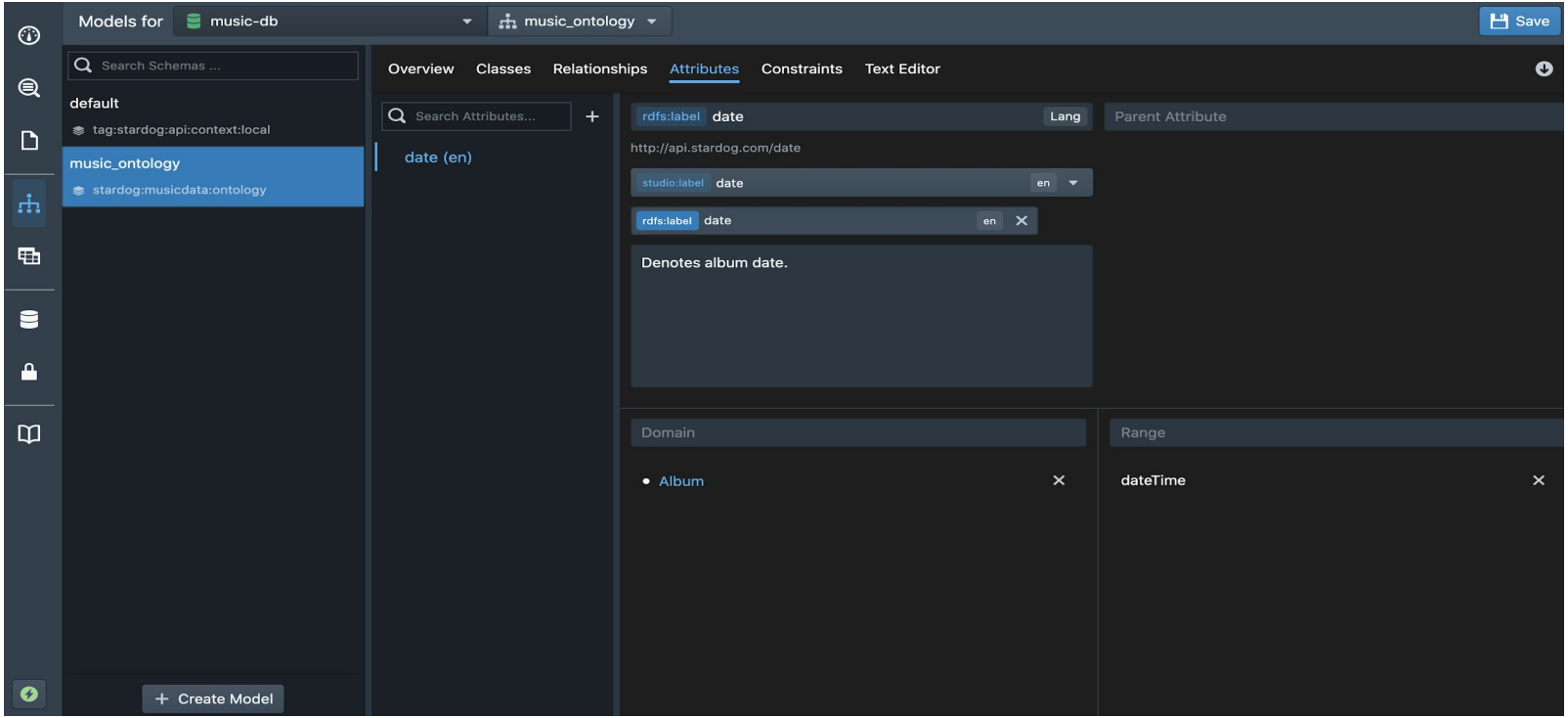
# Step 4: Add Relationships (Object Properties)



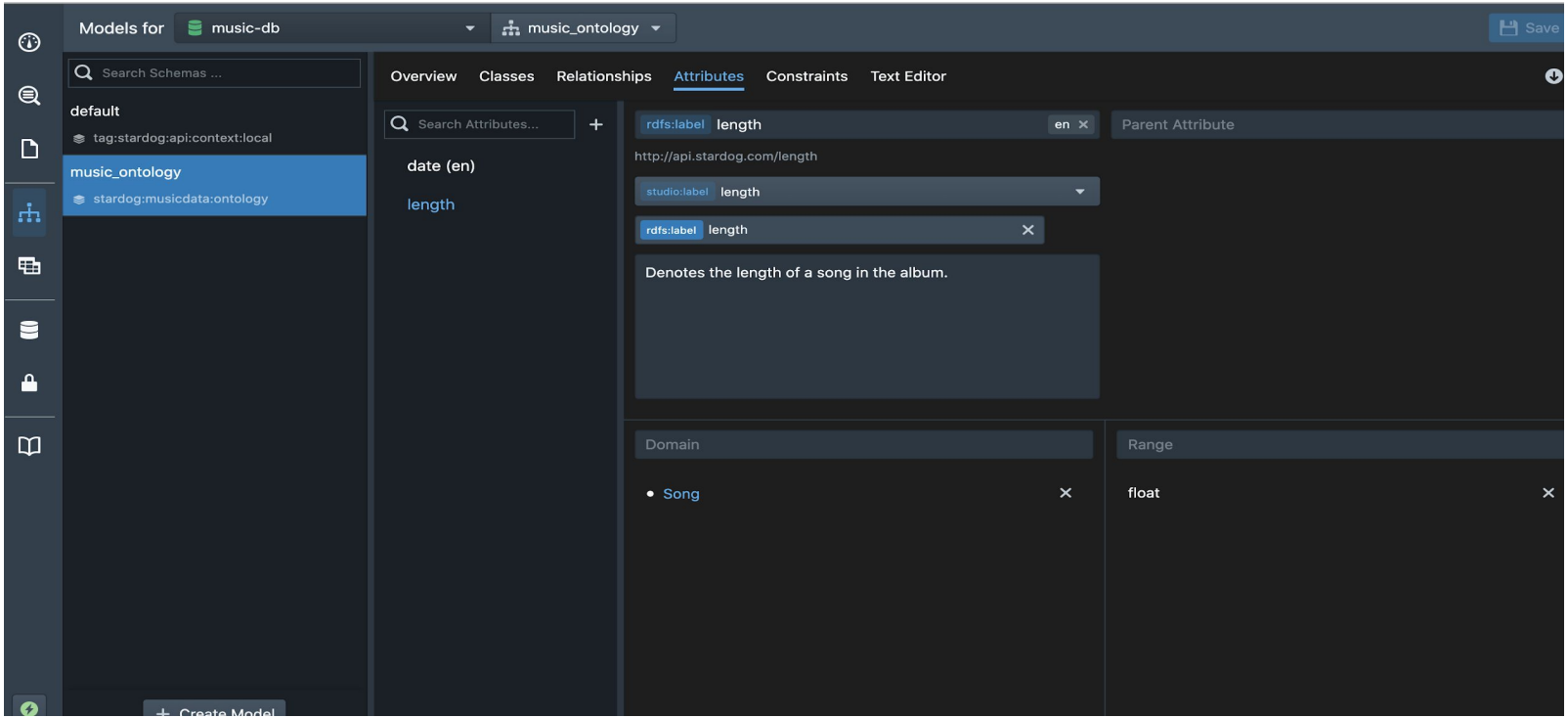
# Step 4: Add Relationships (Object Properties)



# Step 5: Add Attributes (Datatype Properties)



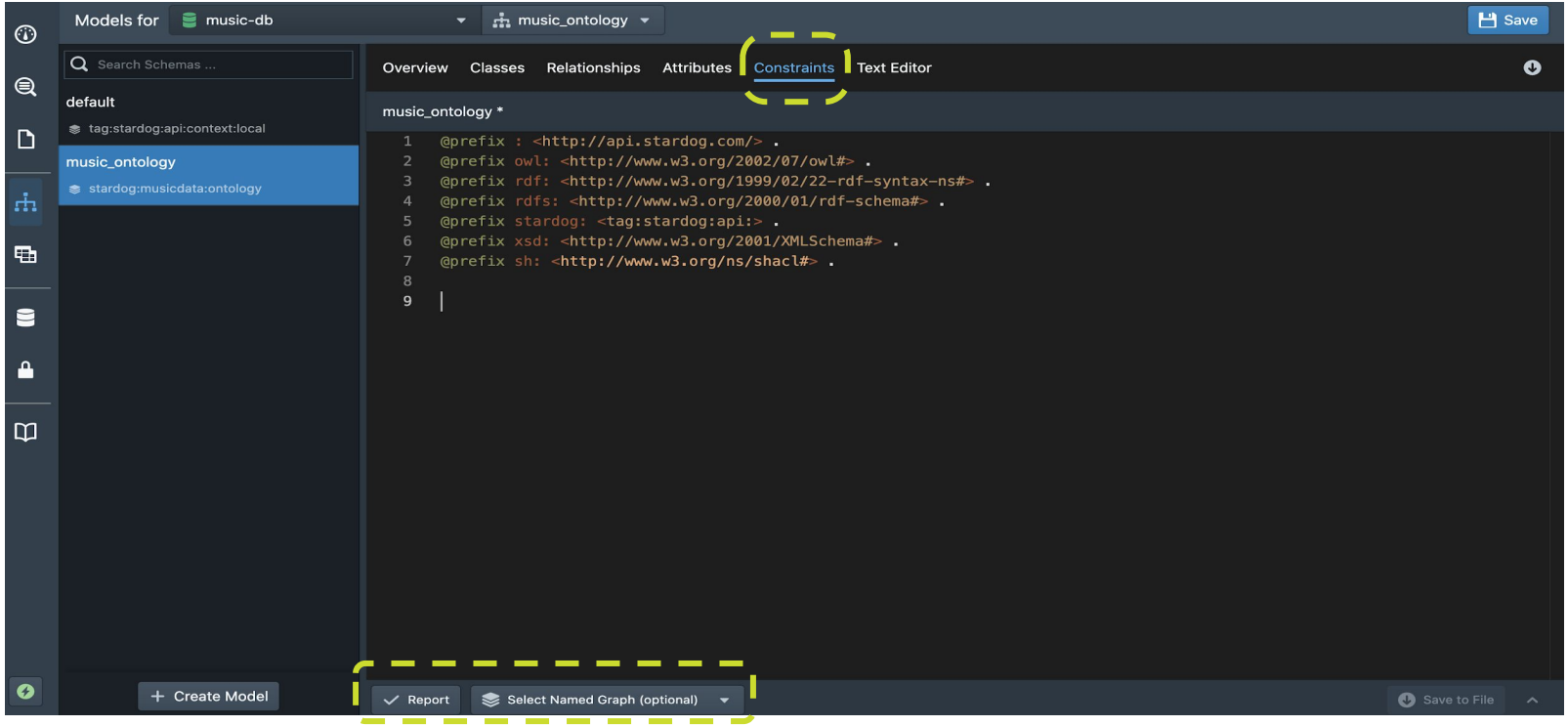
# Step 5: Add Attributes (Datatype Properties)



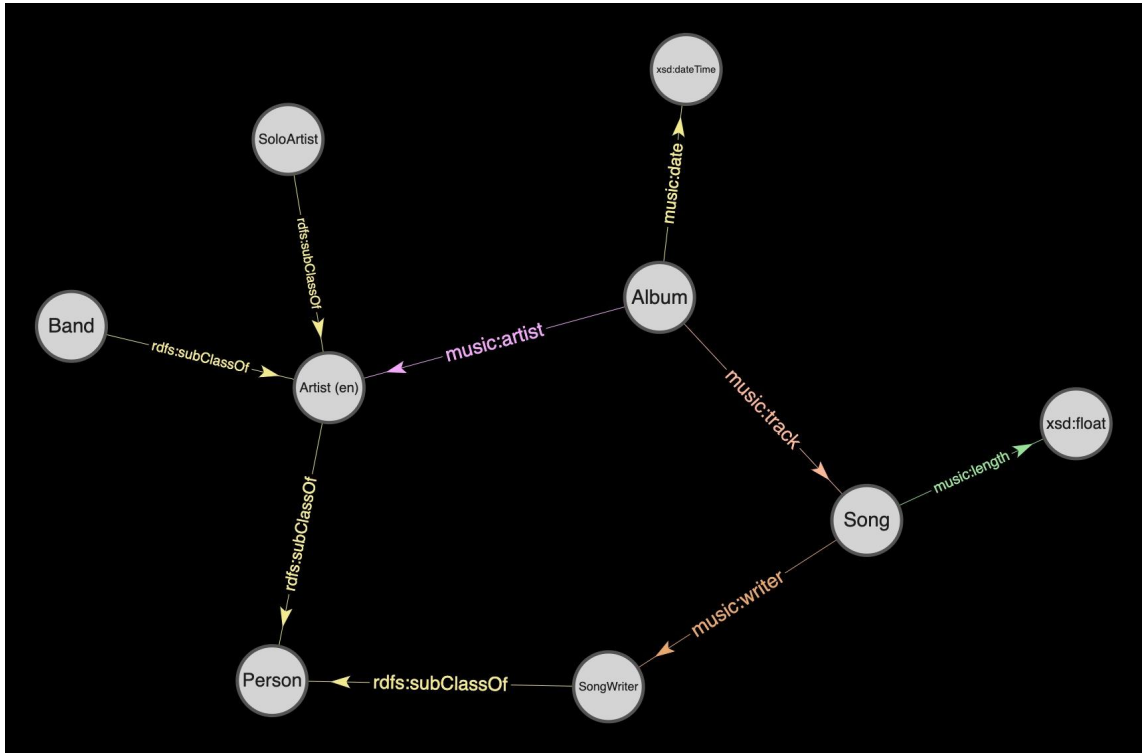
# Step 6: Add Constraints

- Models editor also allows you add SHACL constraints through the constraints tab
- Though will cover in detail the SHACL constraints tutorial as part of separate training session

# Step 6: Add Constraints Tab



# Graph Visualization Overview



# Learning Objectives

# Learning Objectives

- How to build a data model, using Stardog Studio, to support your Enterprise Knowledge Graph
- Fundamentals of data models and data modeling as a process
- Basics of existing types of models including ontology models
- The process of successful ontology model development and associated best practices
- Foundational concepts of ontology models, illustrated through examples, including an overview of existing modeling standards and language representations

# Thank you