

Predicting links in a Knowledge Graph

Overview of an approach from raw data to
insights



Laura Firey Monroe, Stardog - laura.firey.monroe@stardog.com



Christophe Gueret, Accenture Labs - christophe.gueret@accenture.com

Outline

The tutorial will run for 90 minutes and cover a “full stack” discussion from raw data to insights

Q&A available at any time + 15 minutes reserved at the end

- 1** Overview (15 min)
 - 1.2** Data integration
 - 1.3** Graph machine learning
- 2** Hands-on session (60 min)
 - 2.1** Putting a Knowledge Graph together
 - 2.2** Using graph machine learning
- 3** Closing and Q&A (15 min)



What this tutorial will and will not cover

The objective of the tutorial is to get an impression of the end-to-end pipeline for getting from raw data to predicting new links in the graph. No prior experience is needed.

What is covered

- General approach for predicting links in graph
- Using a no-code environment to design a graph easily

What is not covered

- In depth optimisation of the machine learning component
- Advanced features of Stardog

Dedicated tutorial material exists to explore things in depth: [Ampligraph](#), [Stardog](#)



Outline

Objective:

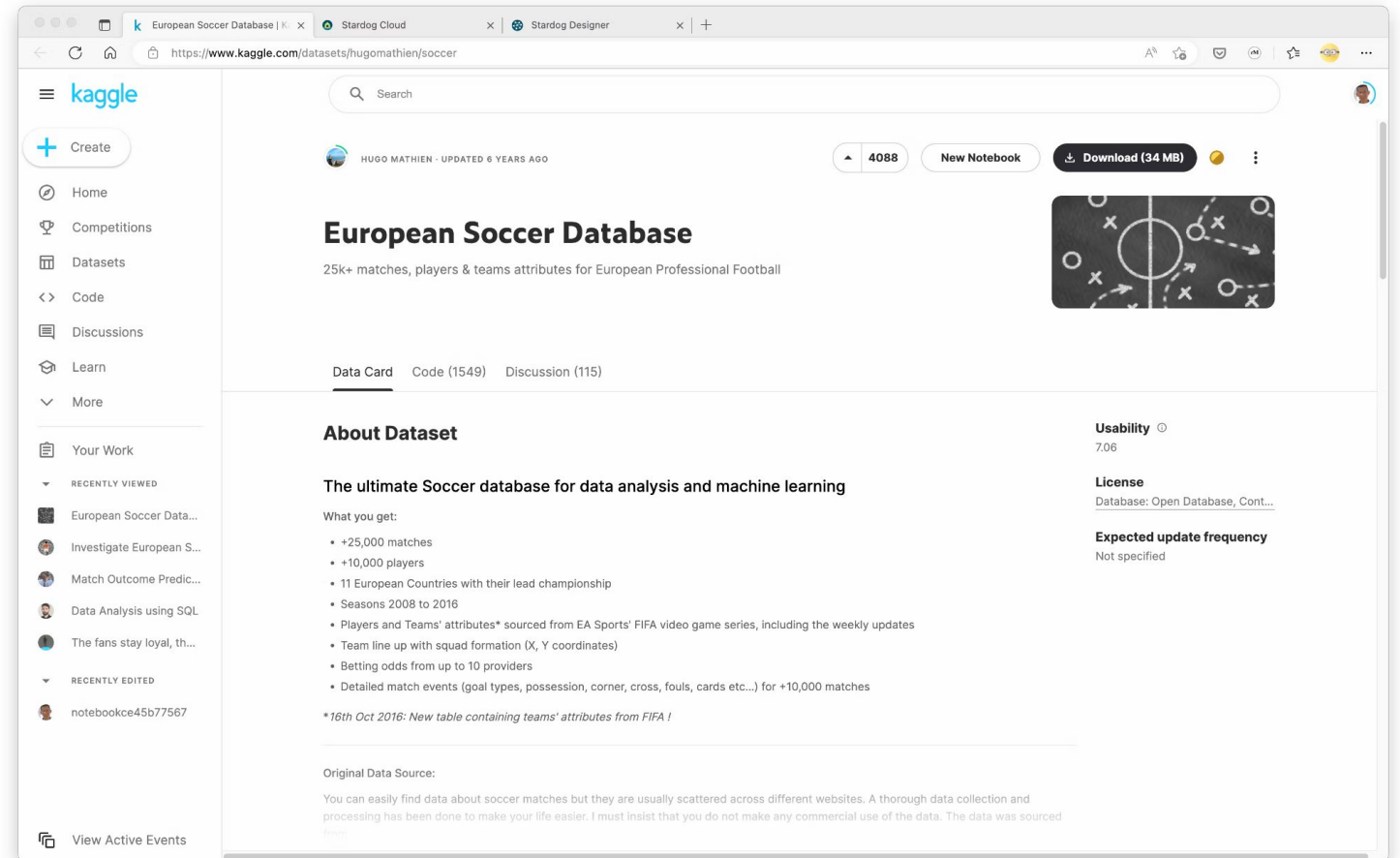
Get more familiar with the concepts behind the approach taken for the tutorial

- 1** Overview (15 min)
 - 1.2** Data integration
 - 1.3** Graph machine learning
- 2** Hands-on session (60 min)
 - 2.1** Putting a Knowledge Graph together
 - 2.2** Using graph machine learning
- 3** Closing and Q&A (15 min)

Use-case

Could we use this dataset to predict which player could play for a particular team?

We will approach this using a knowledge graph and a machine learning technique based on graphs



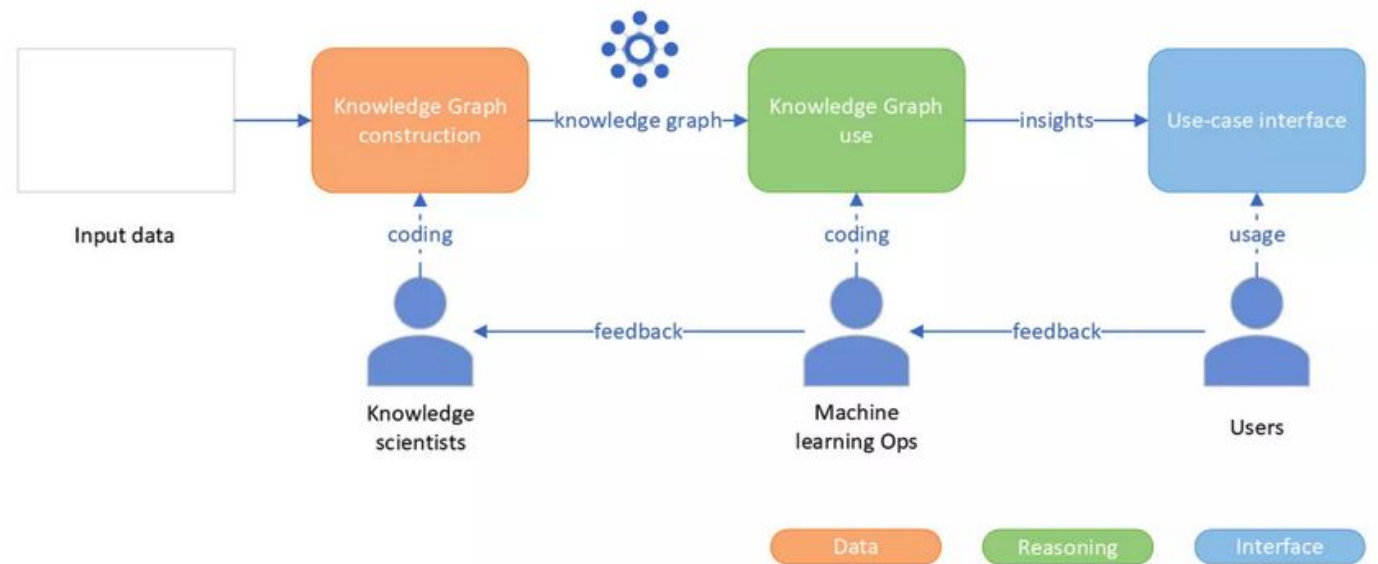
The screenshot shows the Kaggle dataset page for 'European Soccer Database' by Hugo Mathien. The page includes a search bar, a sidebar with navigation options like 'Home', 'Competitions', 'Datasets', and 'Code', and a main content area. The main content area features a title 'European Soccer Database', a description '25k+ matches, players & teams attributes for European Professional Football', and a 'Data Card' section. The 'Data Card' section includes an 'About Dataset' section with the subtitle 'The ultimate Soccer database for data analysis and machine learning' and a list of features: '+25,000 matches', '+10,000 players', '11 European Countries with their lead championship', 'Seasons 2008 to 2016', 'Players and Teams' attributes* sourced from EA Sports' FIFA video game series, including the weekly updates', 'Team line up with squad formation (X, Y coordinates)', 'Betting odds from up to 10 providers', and 'Detailed match events (goal types, possession, corner, cross, fouls, cards etc...) for +10,000 matches'. There is also a note about a new table containing teams' attributes from FIFA 16th Oct 2016. The 'Usability' is 7.06 and the 'License' is 'Database: Open Database, Cont...'. The 'Expected update frequency' is 'Not specified'. The 'Original Data Source' section explains that the data is sourced from various websites and is provided for personal use.

Approach

We will follow a common pattern that sees a team design a Knowledge Graph used by a machine learning team to derive insights.

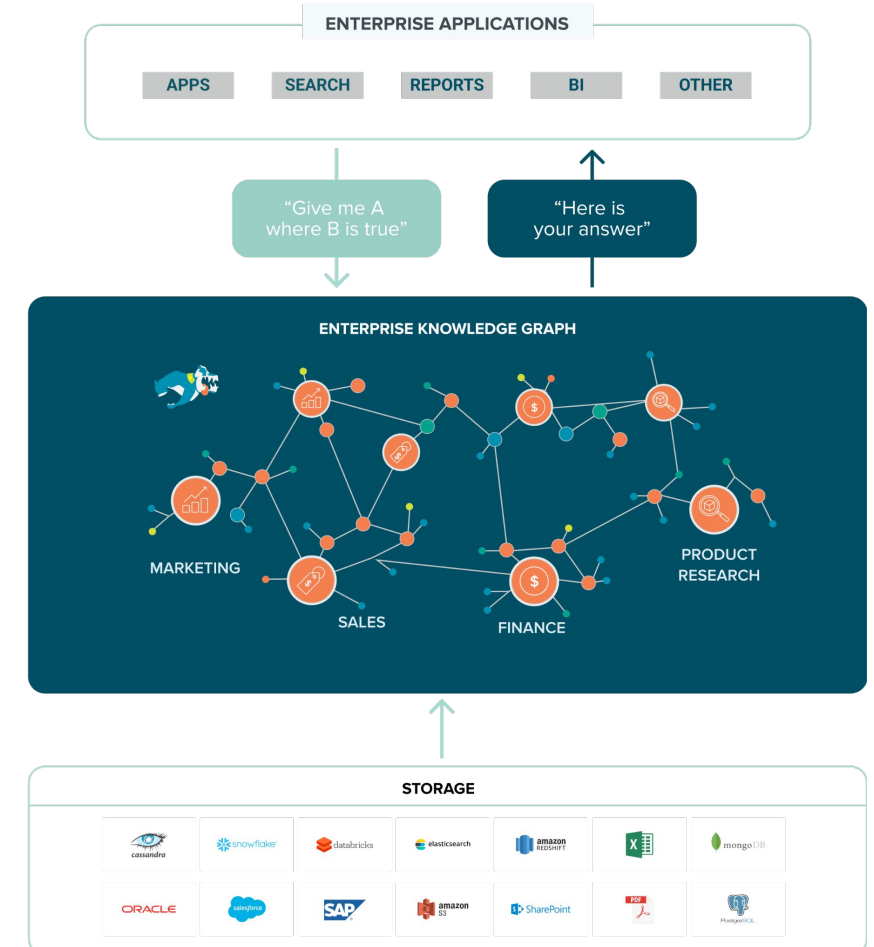
Our tech stack will be:

- **Stardog** for the graph
- **Ampligraph** for the machine learning
- **Jupyter** for the user interface



Connecting data

Knowledge Graphs are a flexible and scalable approach to integrate data from different silos around a defined target ontology



<https://www.stardog.com/blog/what-is-a-data-mesh-principles-and-architecture/>

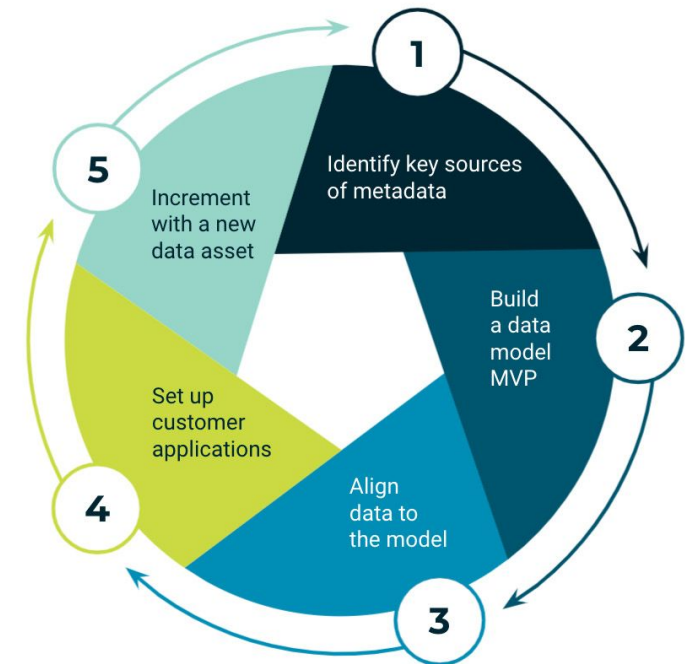


Pragmatic and scalable approach

It is not necessary, nor useful, to aim for the ultimate KG from the start of an implementation

Instead, the creation is best approach as an iterative construction centered around solving a particular use-case

For the tutorial today we start from scratch and implement one use-case but we could have used an existing data fabric, or plan to use the one we will make as a base for more use-cases

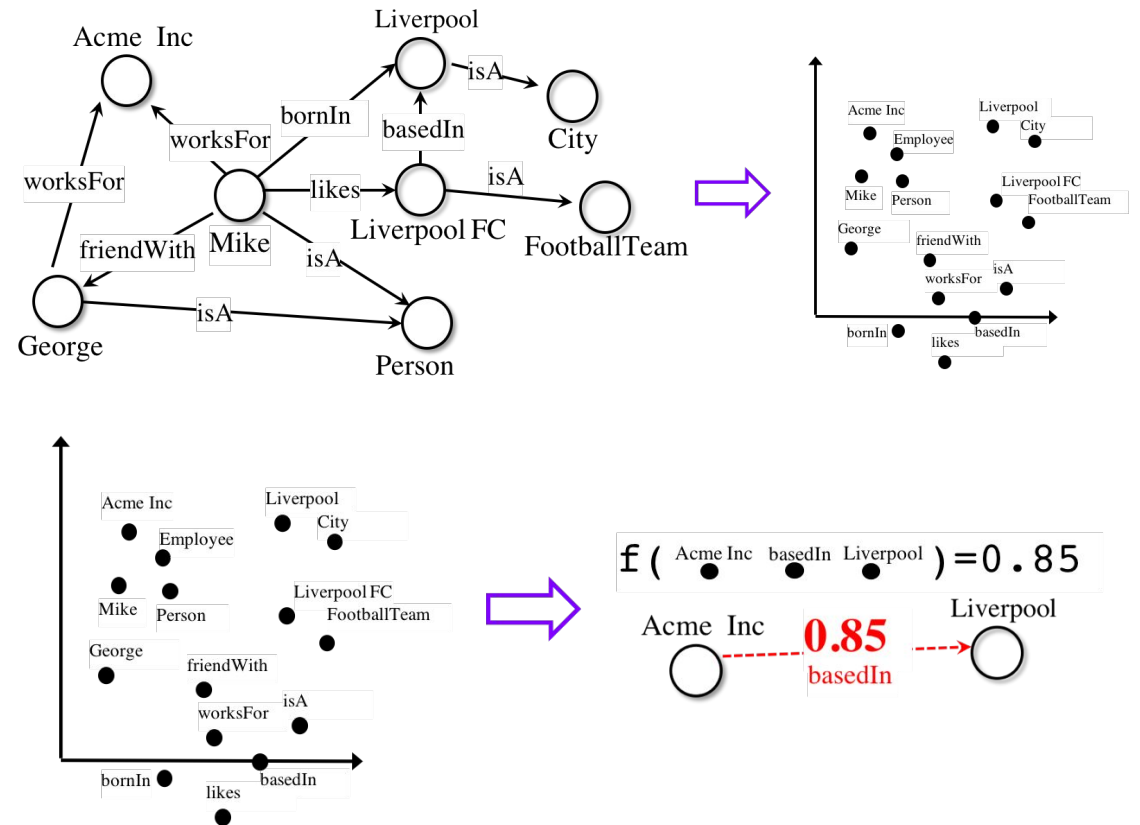


<https://www.stardog.com/blog/5-steps-to-building-a-data-fabric>

Graph machine learning

This is a set of machine learning techniques that project a graph content into an embedding space (set of vectors of high dimensions)

This embedding space is then used to make predictions about potential new edges in the graph



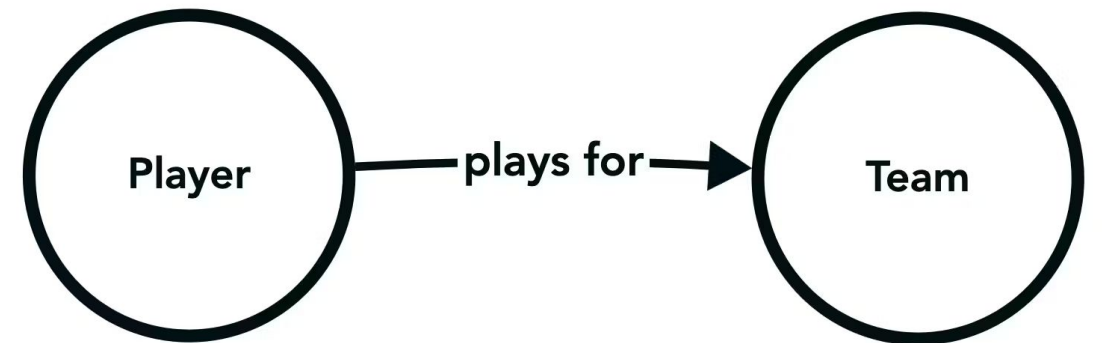
<https://github.com/Accenture/AmplifyGraph/>

Mapping our use-case as link prediction

Our use-case objective of predicting who plays for which team can be cast as a link “plays for” between two types of nodes “Player” and “Team” (at any point in time)

What we will need to make it work is

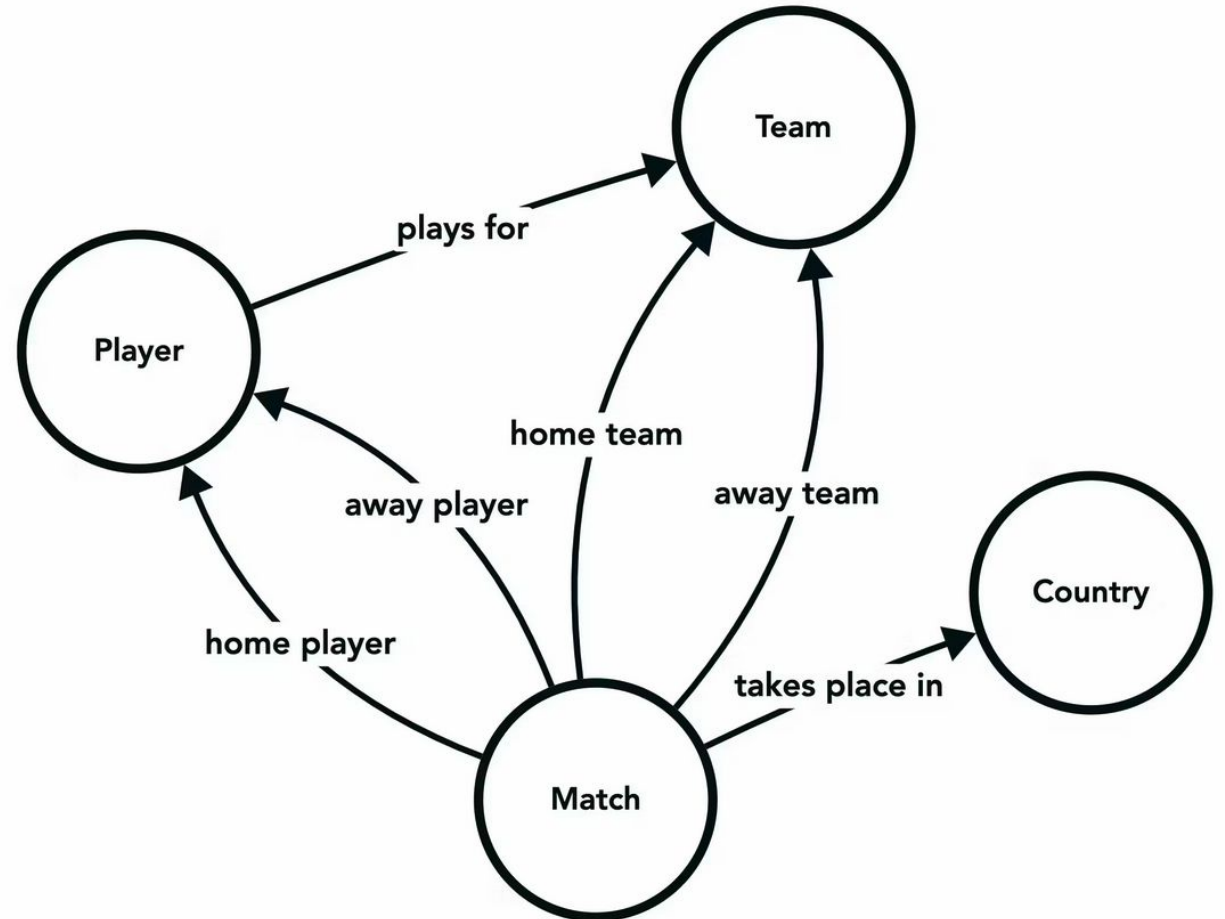
- Instances of that edge as training data
- More contextual information about Players and Teams



Target ontology

This is the target ontology. The design process (which we will cover) is to find a trade-off between what would be useful and what is available in the data

For example, the salary of the players and the budget of the teams likely play a big role in allocation. But this data was not available in the dataset considered



Most likely associations

Skipping right to the end, here is an example of the results aimed at

We will obtain proposal for new links between players and teams along with a likelihood score (uncalibrated so only meaningful as a rank)

	s	p	o	scores
65209	Lionel Messi	plays for	Real Madrid CF	0.511274
229018	Federico Peluso	plays for	Palermo	0.462421
88682	Emanuele Calaiò	plays for	Palermo	0.437658
49507	Cristiano Ronaldo	plays for	FC Barcelona	0.406167
199485	Jacques Alaixys Romao	plays for	Toulouse FC	0.387238
14556	Pablo Armero	plays for	Catania	0.384859
213089	Jordan Bowery	plays for	Stade Brestois 29	0.371333
140673	Eugen Polanski	plays for	Hamburger SV	0.371240
199509	Jacques Alaixys Romao	plays for	AS Saint-Étienne	0.362103
88678	Emanuele Calaiò	plays for	Torino	0.360968

Outline

Objective:

Implement our
use-case!

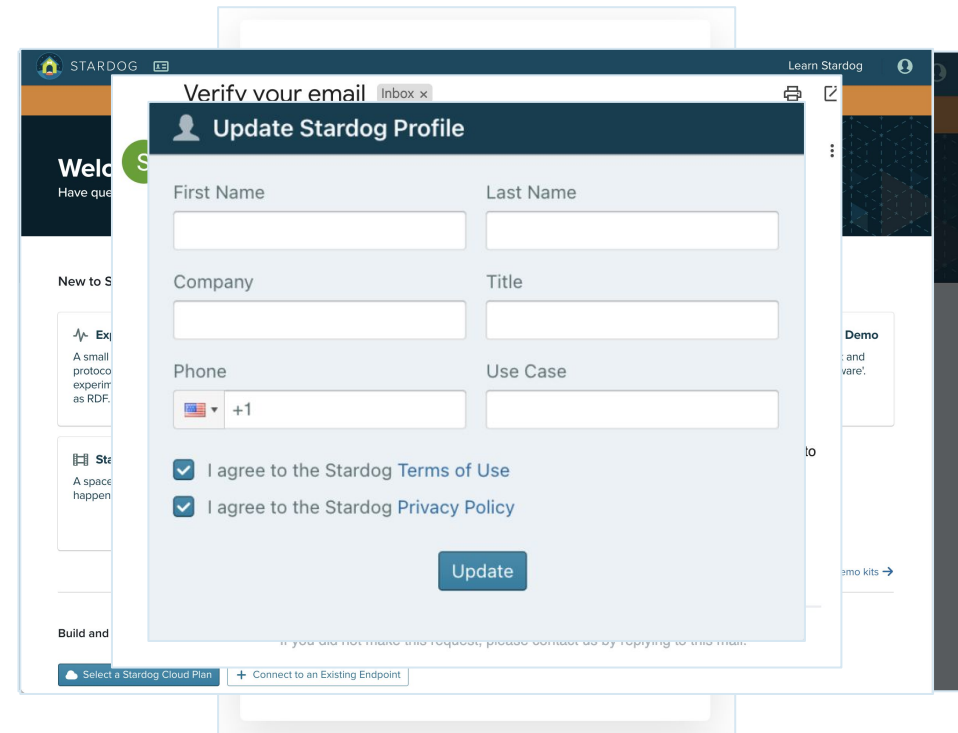
- 1** Overview (15 min)
 - 1.2** Data integration
 - 1.3** Graph machine learning
- 2** Hands-on session (60 min)
 - 2.1** Putting a Knowledge Graph together
 - 2.2** Using graph machine learning
- 3** Closing Q&A (15 min)



Get a Stardog cloud account

We will use Stardog Cloud to create the Knowledge Graph

Head to <https://cloud.stardog.com> to create an account there



The image shows a screenshot of the Stardog Cloud web interface. The main focus is a modal window titled "Update Stardog Profile" with a "Verify your email" notification at the top. The form contains the following fields and options:

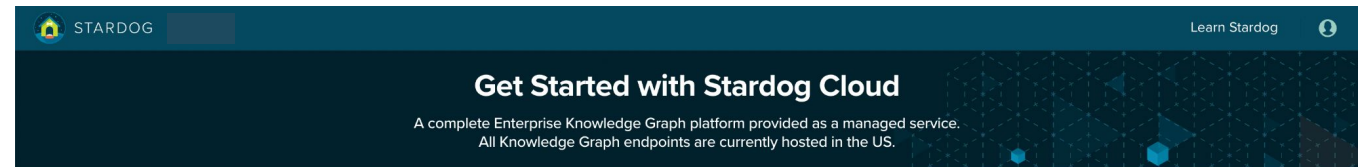
- First Name**: Text input field.
- Last Name**: Text input field.
- Company**: Text input field.
- Title**: Text input field.
- Phone**: Text input field with a country code dropdown (currently showing +1).
- Use Case**: Text input field.
- I agree to the Stardog Terms of Use
- I agree to the Stardog Privacy Policy
- Update**: A blue button to submit the form.

At the bottom of the modal, there are two buttons: "Select a Stardog Cloud Plan" and "+ Connect to an Existing Endpoint".

Select a Stardog Cloud environment

Next, pick either a free instance to get started or register for the essentials pack for a more capable instance.

For this tutorial the free instance is sufficient to play around



Select the plan that works for you

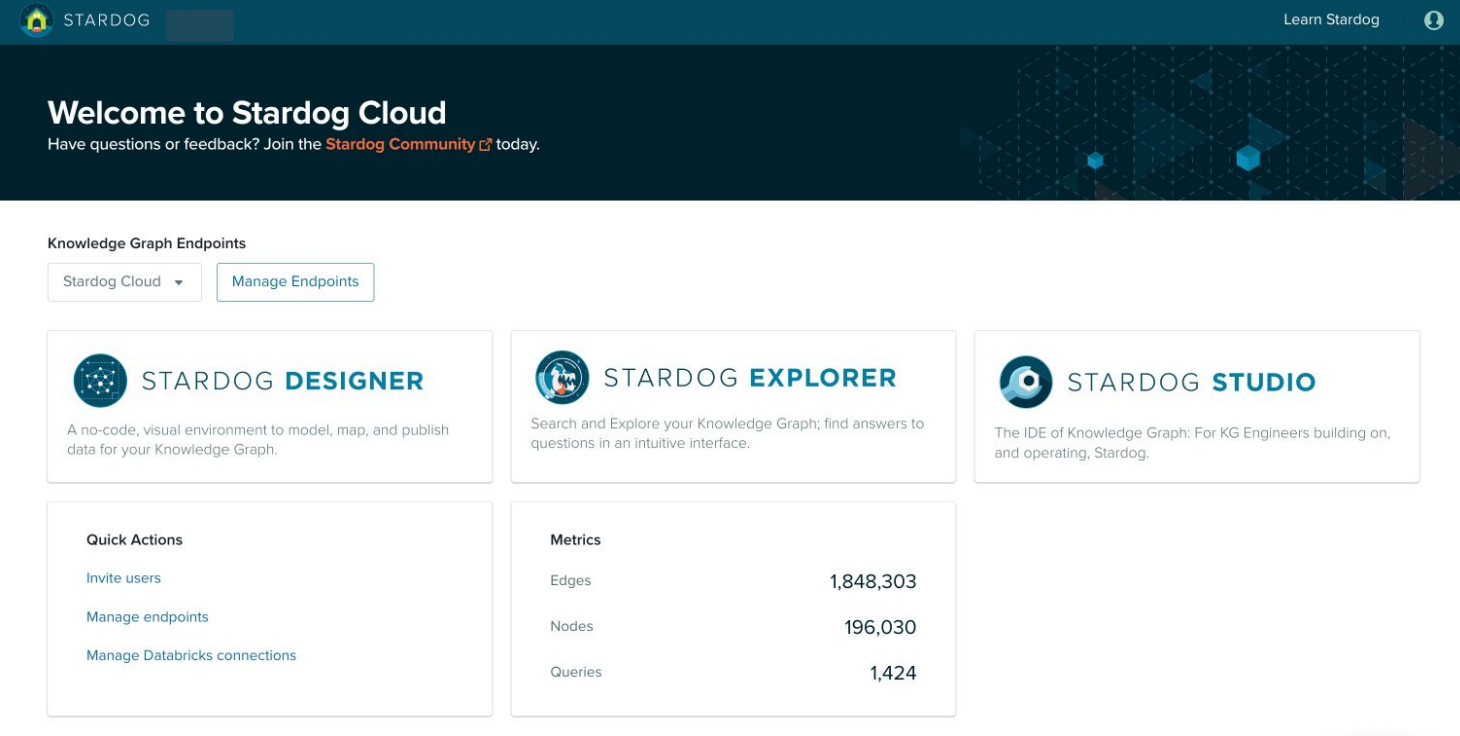
Free	Essentials	Enterprise
Learning environment to create your first Knowledge Graph.	Everything you need for a PoC or development environment.	Full production environment tailored to the needs of your company.
Free	\$99/month	Custom
Feature Highlights	Everything in Free, plus	Everything in Essentials, plus
<ul style="list-style-type: none">Full Stardog EKG PlatformStore up to 10M edges in 3 databasesCommunity Support, no SLA	<ul style="list-style-type: none">Store up to 50M edges5 DatabasesCommunity Support, 95% uptime SLA	<ul style="list-style-type: none">Unlimited edgesUnlimited databasesPremium Support, 99.9% uptime SLA
Get Stardog Free	Get Stardog Essentials	Contact Us

Connecting to Knowledge Graph endpoints later on

The environment you just created will be automatically selected for you as your Knowledge Graph Endpoint. Your eventual additional endpoints connected to through Stardog Cloud will also be available

This interface can connect to any machine your web browser can reach

(it is a web client connecting directly to the server, no data transits via Stardog's infrastructure)



The screenshot shows the Stardog Cloud dashboard. At the top, there is a header with the Stardog logo and a "Learn Stardog" link. Below the header, a welcome message reads "Welcome to Stardog Cloud" and "Have questions or feedback? Join the Stardog Community today." The main content area is titled "Knowledge Graph Endpoints" and features a "Stardog Cloud" dropdown menu and a "Manage Endpoints" button. Below this, there are three cards for "STARDOG DESIGNER", "STARDOG EXPLORER", and "STARDOG STUDIO". The "STARDOG EXPLORER" card includes a "Metrics" table with the following data:

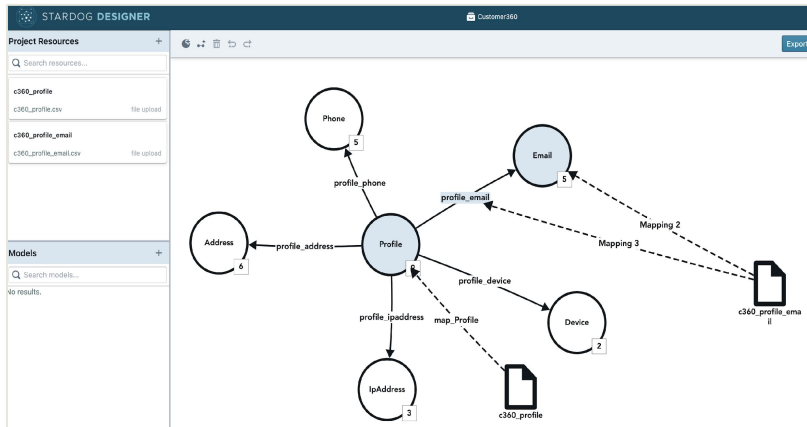
Metrics	
Edges	1,848,303
Nodes	196,030
Queries	1,424

Stardog tooling



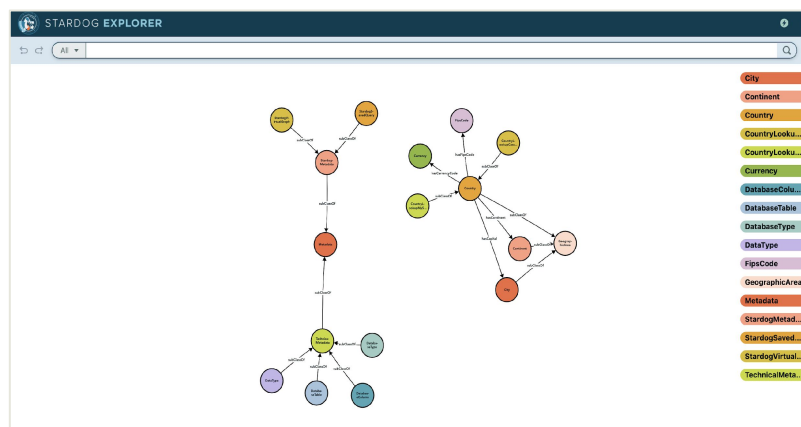
STARDOG DESIGNER

A no-code, visual environment to model, map, and publish data for your Knowledge Graph



STARDOG EXPLORER

An intuitive interface to visually browse and query your Knowledge Graph



STARDOG STUDIO

A complete IDE to program your Knowledge Graph

The screenshot shows the Stardog Studio interface. The top part displays a 'QUERIES' pane with a search bar and a list of queries. The main workspace shows a SPARQL query editor with the following code:

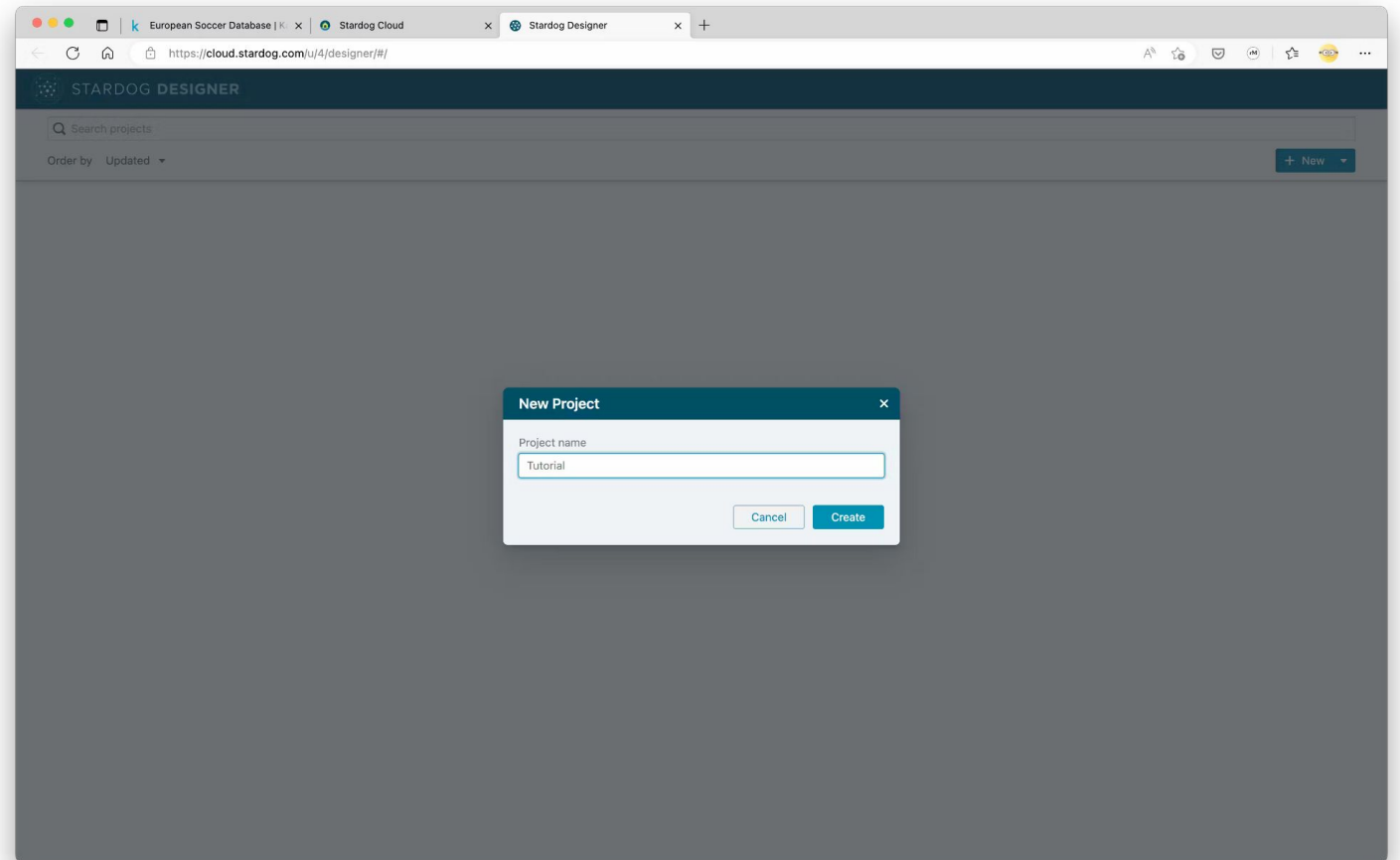
```
1 Query that returns the performance history for a specific benchmark.
2 # This query can be run against the Stardog server at
3 # http://ec2-54-174-180-239.compute-1.amazonaws.com:5828 that stores all historical starbench
4 # results.
5 #
6 # Set the ?test in the filter to see the history
7
8
9 prefix : <http://stardog.com/sbdashboard/>
10 prefix br: <http://stardog.com/sbdashboard/branches/>
11 prefix rgr: <http://stardog.com/sbdashboard/rungroups/>
12 prefix test: <http://stardog.com/sbdashboard/tests/>
13
14 SELECT ?test ?date ?build ?commit ?actual ?expected {
15   ?run a :RunGroup ;
16     ?forTest/rdfs:label ?test ;
17     ?date ?date ;
18     ?buildNum ?build ;
19     ?forBranch br:develop ;
20     ?forStarbenchBranch br:SB_master ;
21     ?buildHash ?commit ;
22     ?buildNum ?build ;
23     ?data ?data ;
24
25     bind(xsd:decimal(strbefore(?date, "/")) as ?actual)
26     bind(xsd:decimal(replace(strafter(?date, "/"), ".", "")) as ?expected)
27     filter (?test = "sbapi06_load")
28 }
29 order by desc(?date) desc(?build)
```

The bottom part of the interface shows a 'Run to File' button and a 'Run to Dashboard' button.



Get started in Designer

The first thing to do is create a project



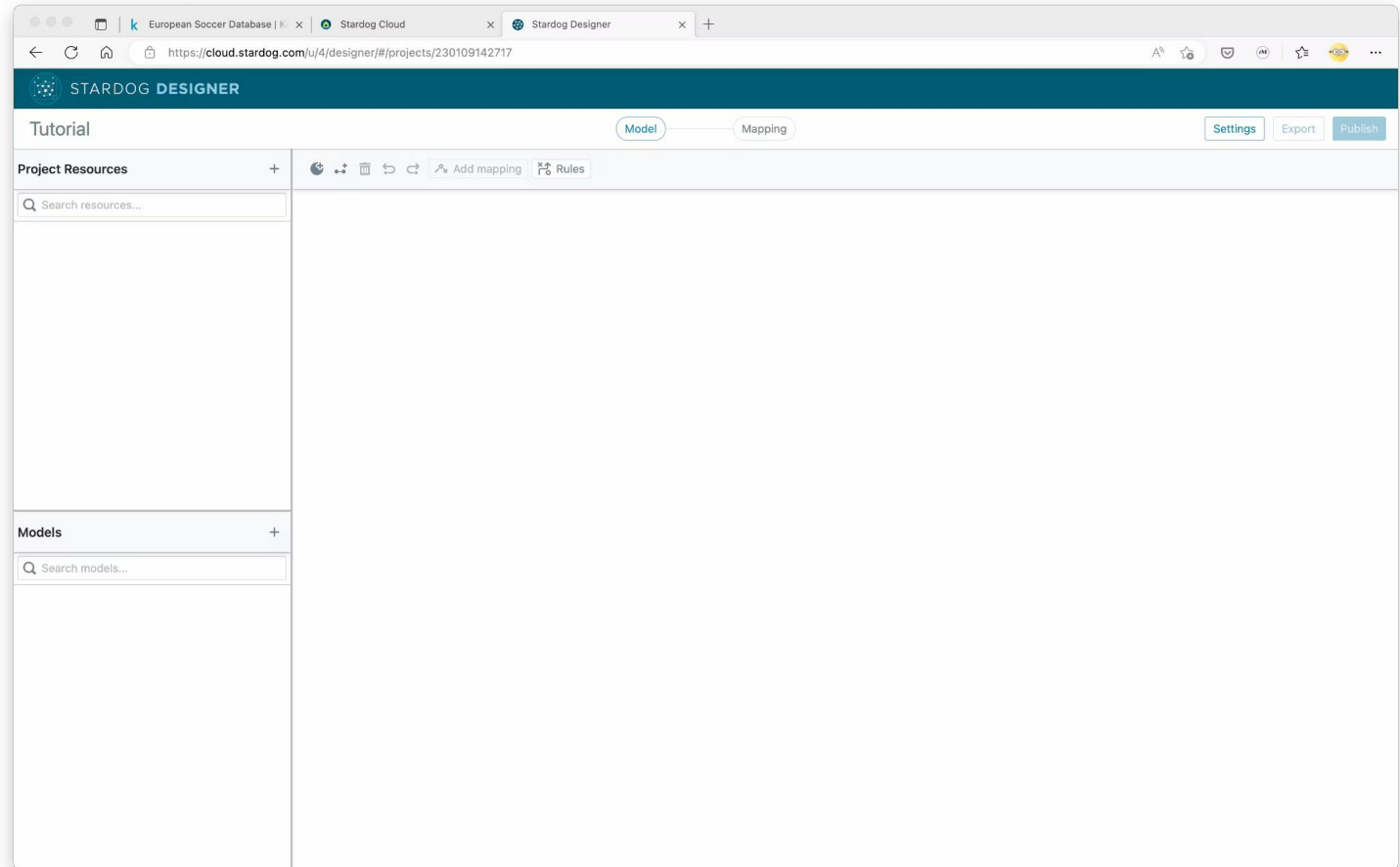
Designer interface

This is the interface of designer. It is organized around two key parts:

- The model, or ontology, which expressed how the data is integrated
- The dataset mapping which map the raw data to triples according to the model

We'll now do several things:

- Add concepts
- Add relations
- Add attributes
- Add rules
- Map the raw data



Adding concepts

Let's return to the data

We can use Kaggle to conveniently explore the concepts in the data

The exercise here consists in going from *columns to concepts* and figure out *what* the data is about

The screenshot shows the Kaggle interface for the 'European Soccer Database' dataset. The main content area displays a table for the 'Country' column, which has 11 rows and 11 unique values. The table lists the following countries: Belgium, England, France, Germany, Italy, Netherlands, Poland, Portugal, Scotland, Spain, and Switzerland. The interface includes a search bar, navigation tabs (Business, Sports, Games, Video Games, Football, Europe), and a 'Data Explorer' sidebar on the right showing the dataset structure (database.sqlite) with columns like Country, League, Match, Player, Player_Attributes, Team, and Team_Attributes. The 'Summary' section indicates 1 file and 199 columns.

# id	A name
1	Belgium
1729	England
4769	France
7889	Germany
18257	Italy
13274	Netherlands
15722	Poland
17642	Portugal
19694	Scotland
21518	Spain
24558	Switzerland

Figuring out the concepts in the data

Below is an example of a result. We assume that league information will not be relevant for the use-case and therefore skip mapping it into the fabric.

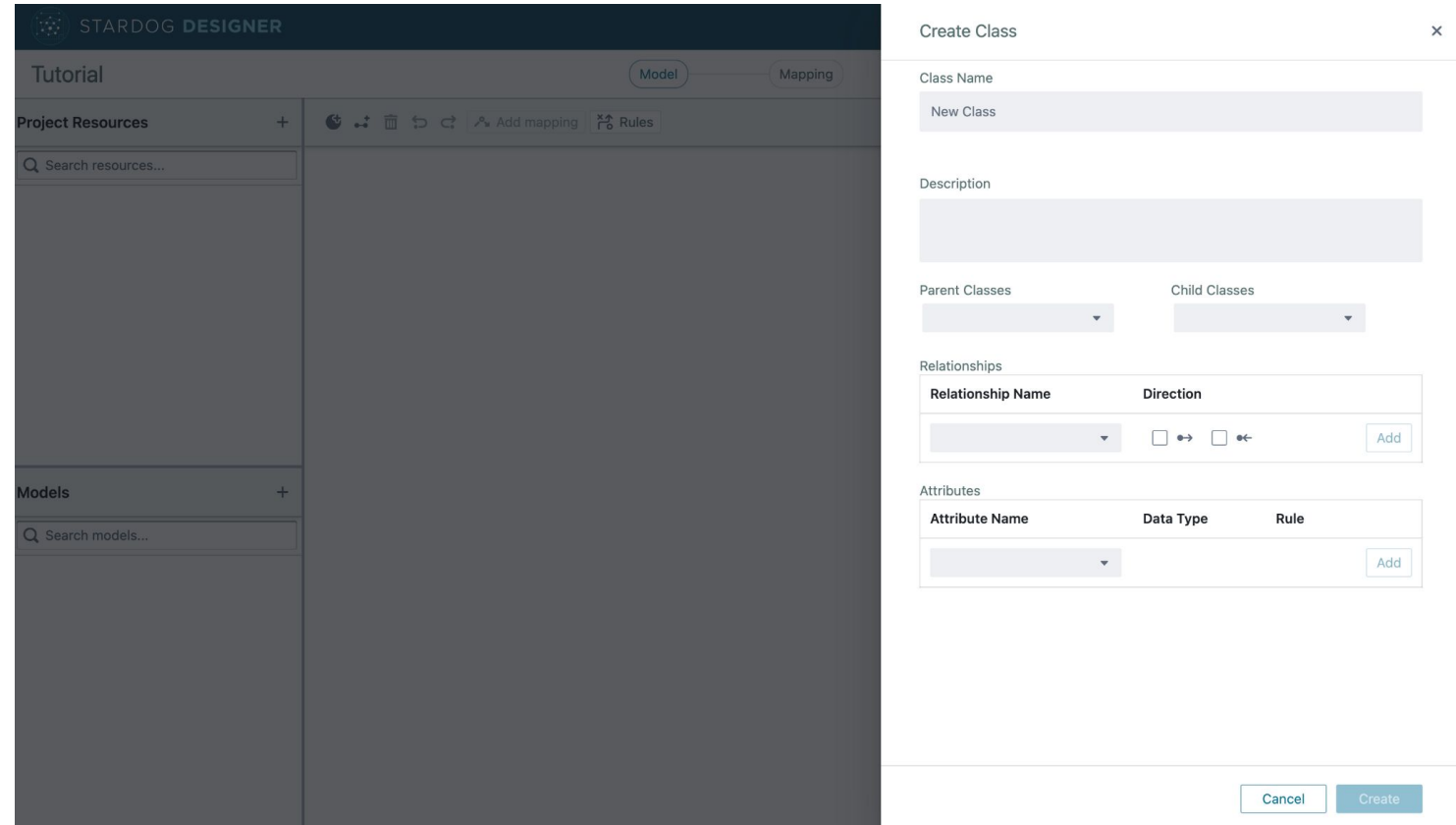
Table	Content	Concepts
Country	Name of the countries	Country
League	List of leagues a particular country can be in	
Match	Competition details with list of players involved and outcomes	Match, Team, Player
Player	Demographics data about the players	Player
Player_Attributes	Key statistics about the players	Player
Team	Name of the teams	Team
Team_Attributes	Key statistics about the teams	Team



Adding classes

Classes can be added with the button “circle+”

There are options for a concept hierarchy which we will not use



The screenshot displays the Stardog Designer interface. On the left, there are panels for 'Project Resources' and 'Models', both with search bars and a '+' button. The main workspace is currently empty. On the right, a 'Create Class' dialog box is open, featuring the following fields and options:

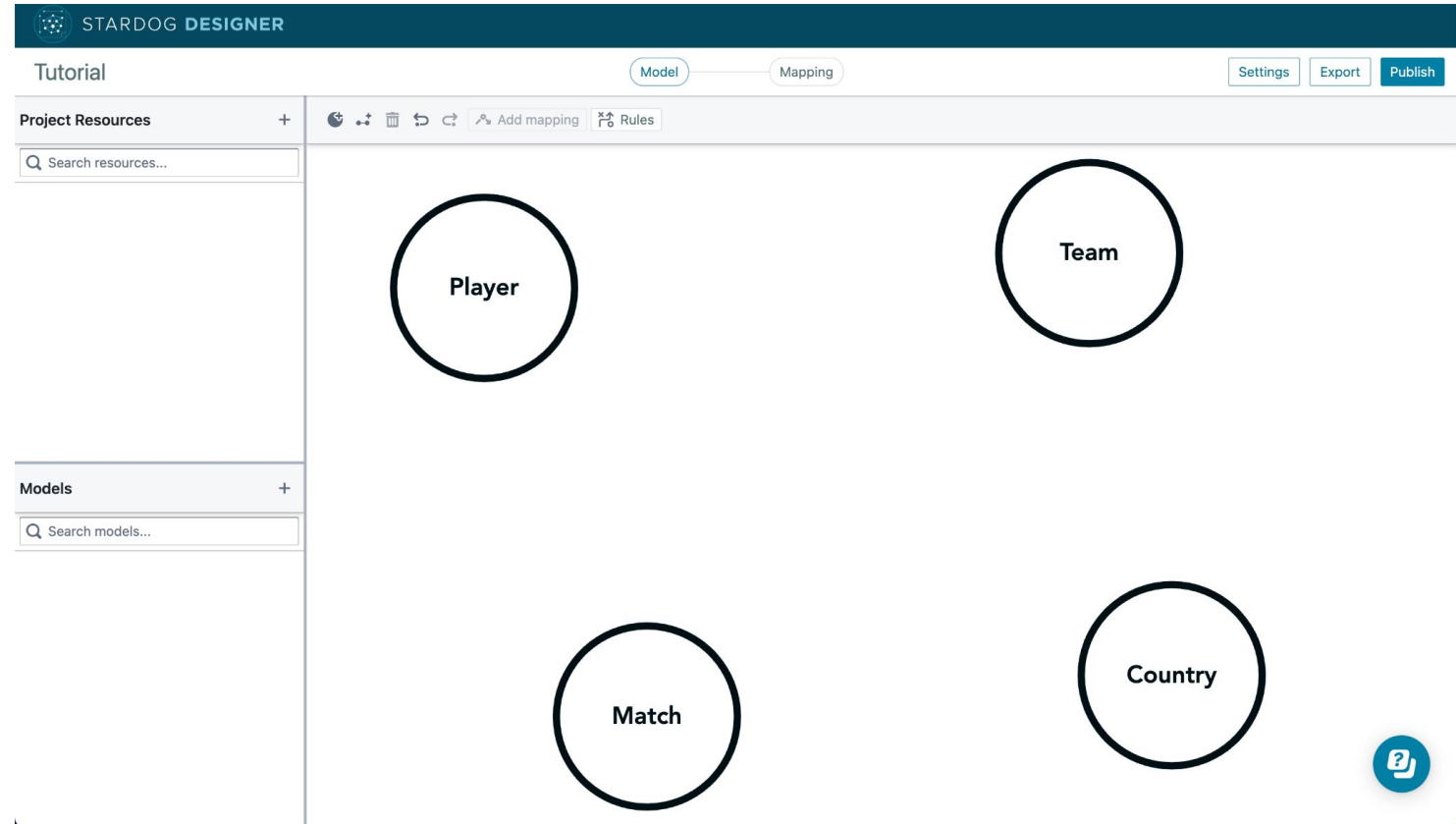
- Class Name:** A text input field containing 'New Class'.
- Description:** A large text area for entering a description.
- Parent Classes:** A dropdown menu.
- Child Classes:** A dropdown menu.
- Relationships:** A table with columns for 'Relationship Name' and 'Direction'. It includes radio buttons for bidirectional (↔) and inverse (←) relationships, and an 'Add' button.
- Attributes:** A table with columns for 'Attribute Name', 'Data Type', and 'Rule'. It includes an 'Add' button.

At the bottom of the dialog, there are 'Cancel' and 'Create' buttons.

End result

According to the table defined earlier we need to create a total of 4 concepts

- Match
- Country
- Team
- Player



Adding relationships

Returning to the definition of the concepts

Relationships are defined in the interface used to create the concepts

The objective is to describe the relation between nodes in semantically meaningful way

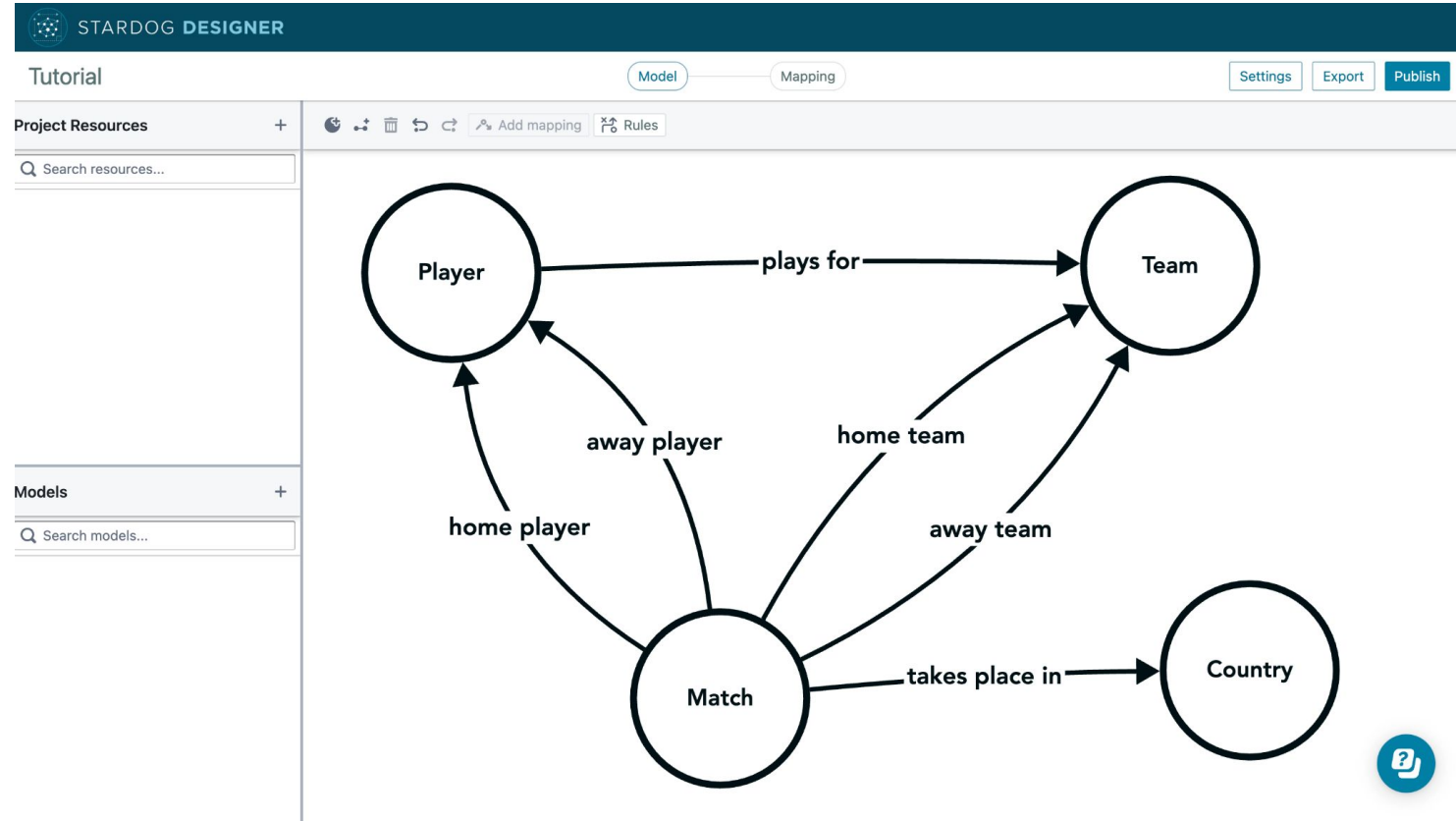
Most often the domain (source node) and range (target node) are not specified as part of the edge label

The screenshot displays the Stardog Designer web interface. The main workspace shows a class diagram with three nodes: 'Match', 'Country', and 'Team'. An arrow labeled 'takes place in' points from the 'Match' node to the 'Country' node. The 'Country' node is highlighted with a thick black border. On the right side, the 'Class Detail' panel is open, showing the configuration for the 'Match' class. The 'Class Name' field is set to 'Match'. Below it, there are sections for 'Parent Classes', 'Child Classes', 'Relationships', and 'Attributes'. The 'Relationships' section contains a table with columns for 'Relationship Name' and 'Direction'. One relationship is defined: 'takes place in' with a bidirectional direction (indicated by a checked box and arrows pointing both ways). The 'Attributes' section is currently empty.

End result

We are now very close to our ontology!

The last thing missing is some attributes about the players



Adding concept attributes

Attributes are configured next to relations

An attribute is something that applies to an instance of a node but does not reflect a relation to another node. It's still a triple in the graph nonetheless!

Typically, those attributes (also called “properties” sometimes) are the likes of

- Date of birth, etc
- Age, duration, etc
- Rank, rating, score, etc

Attributes have a data type. They can be a string, an integer, a timestamp, etc

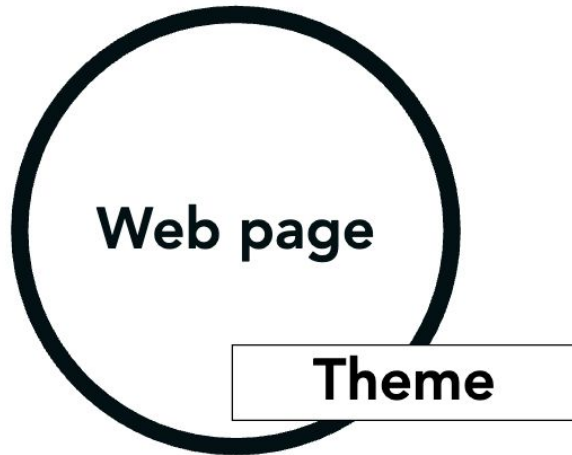
The screenshot shows the Stardog Designer interface. The main workspace displays a graph model with two nodes: 'Player' and 'Match'. The 'Player' node has an attribute 'rating' attached to it. The 'Match' node has relationships 'away player' and 'home player' connected to the 'Player' node. The 'Class Detail' panel on the right shows the configuration for the 'Player' class. It includes a 'Description' field, 'Parent Classes' and 'Child Classes' dropdowns, and a 'Relationships' table. The 'Attributes' table shows the 'rating' attribute with a data type of 'float'.

Relationship Name	Direction
away player	<input type="checkbox"/> ↔ <input checked="" type="checkbox"/> ←
home player	<input type="checkbox"/> ↔ <input checked="" type="checkbox"/> ←
plays for	<input checked="" type="checkbox"/> ↔ <input type="checkbox"/> ←

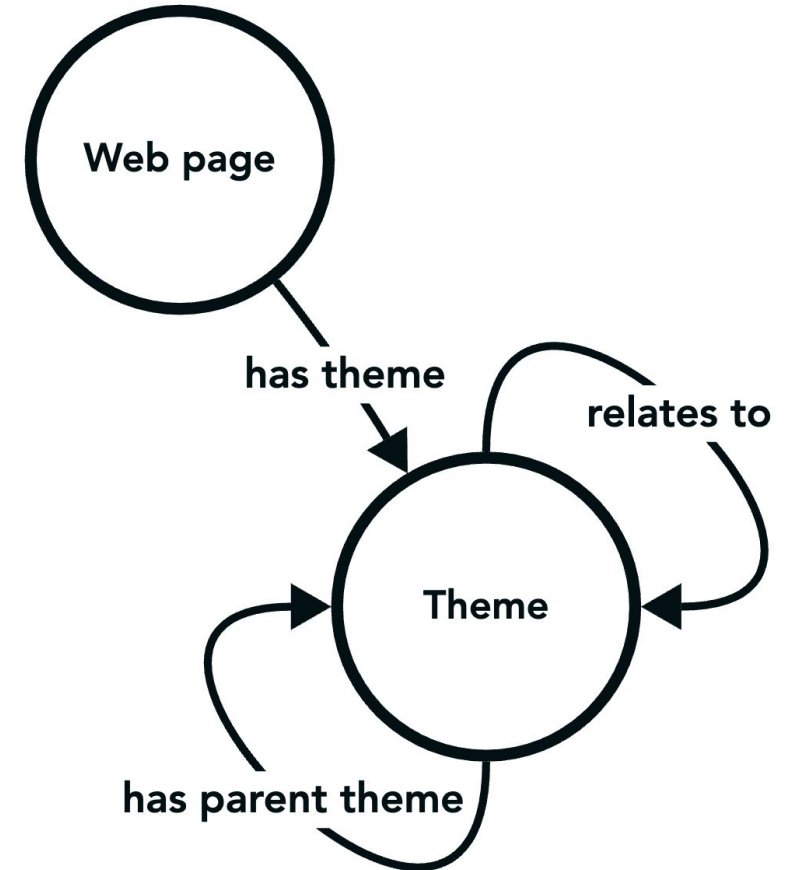
Attribute Name	Data Type	Rule
rating	float	Create

Is this a relation or an attribute?

It depends...



VS

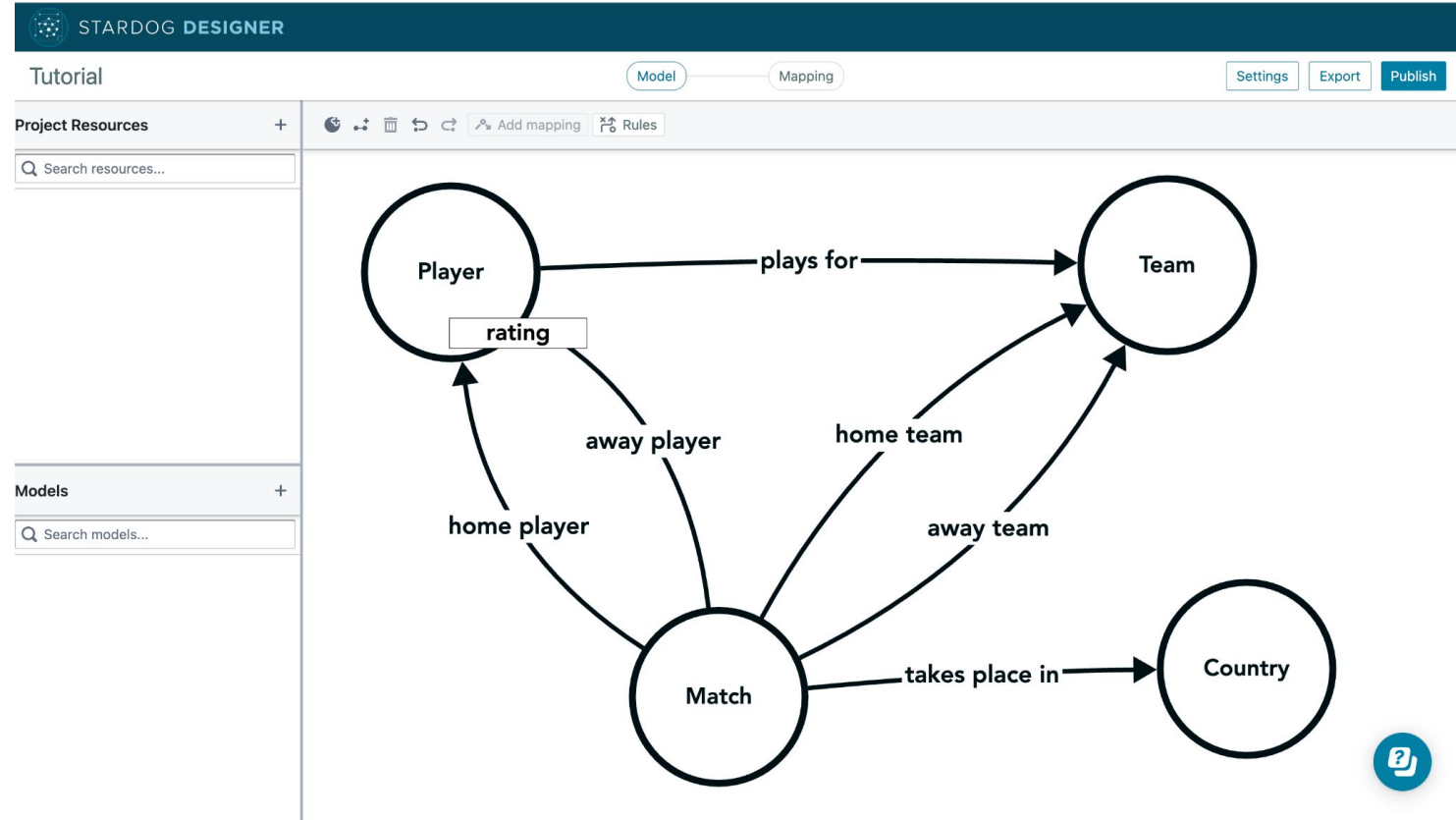


Adding rules

Reasoning rules

We add a reasoning rule to deduce the “plays for” instances from other edges based on a IF / THEN pattern.

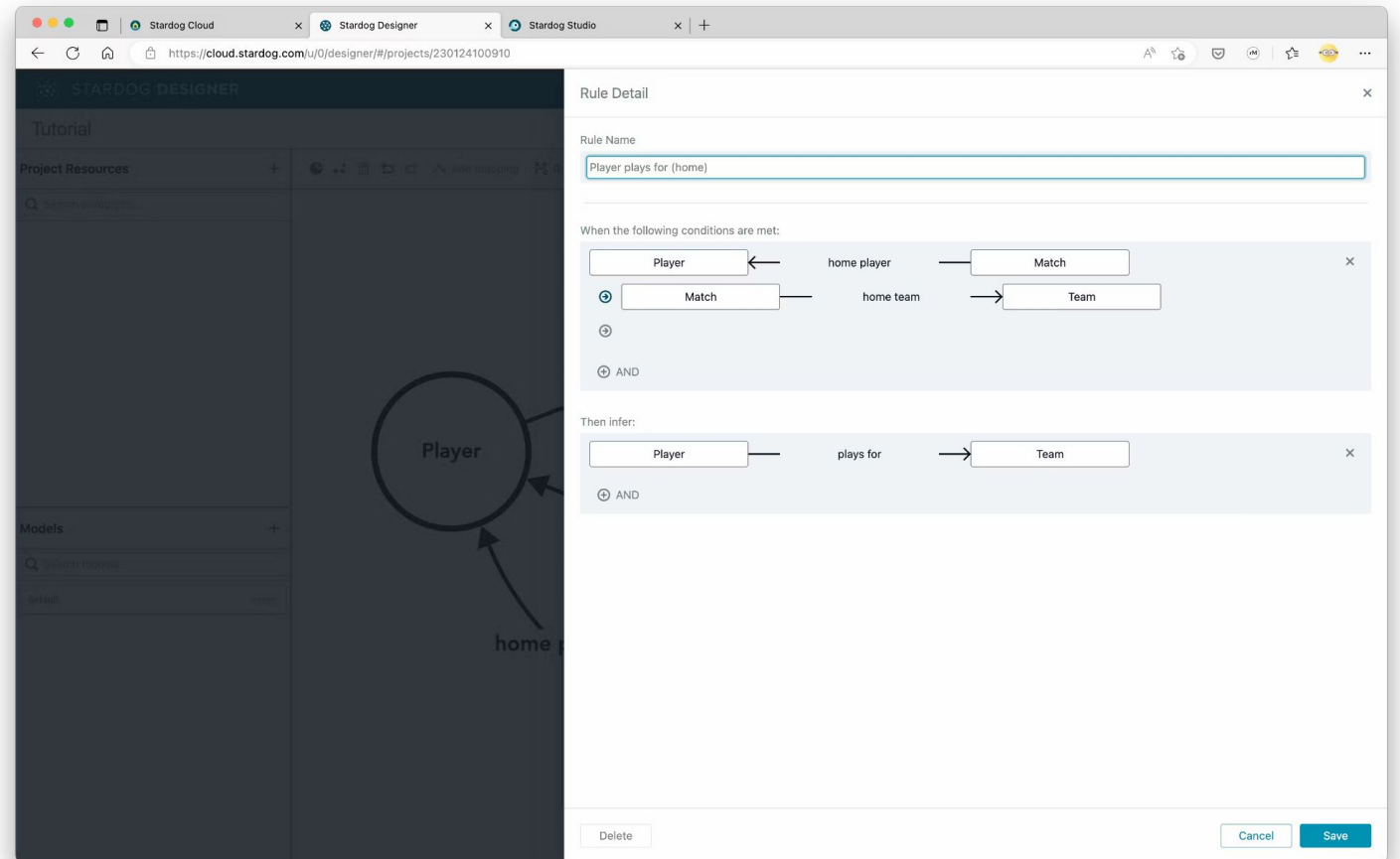
This allows for focusing on expressing core factual data in the graph, and let Stardog figure out the rest.



Definition of the rules

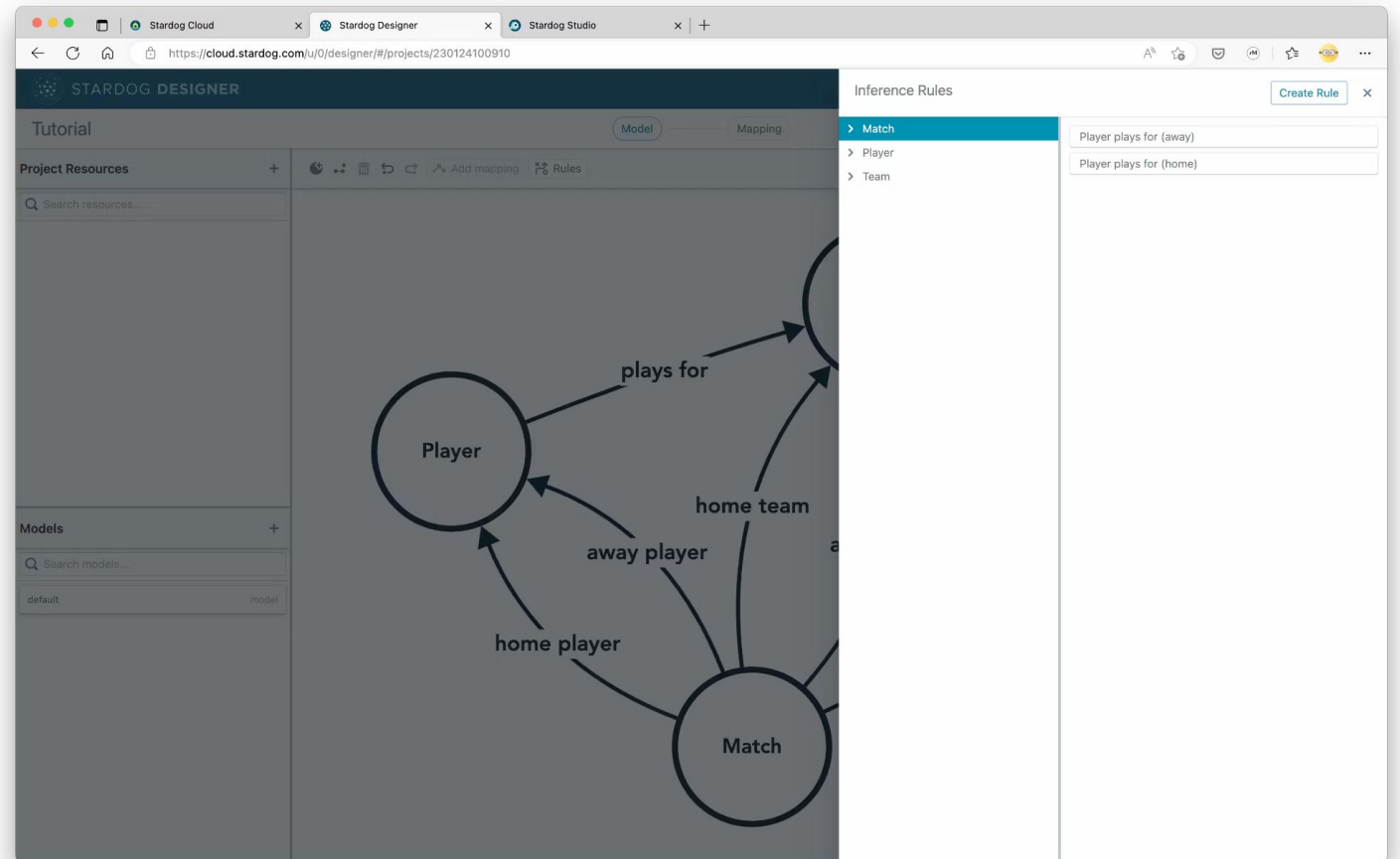
Each rule consists in a set of triples to match as a condition, and another set of triples to create as an outcome

The rules are not executed when they are created. As we will observe later, they are instead used to re-write incoming queries (and are then consumed backwards to their definition)



Checking that everything is fine

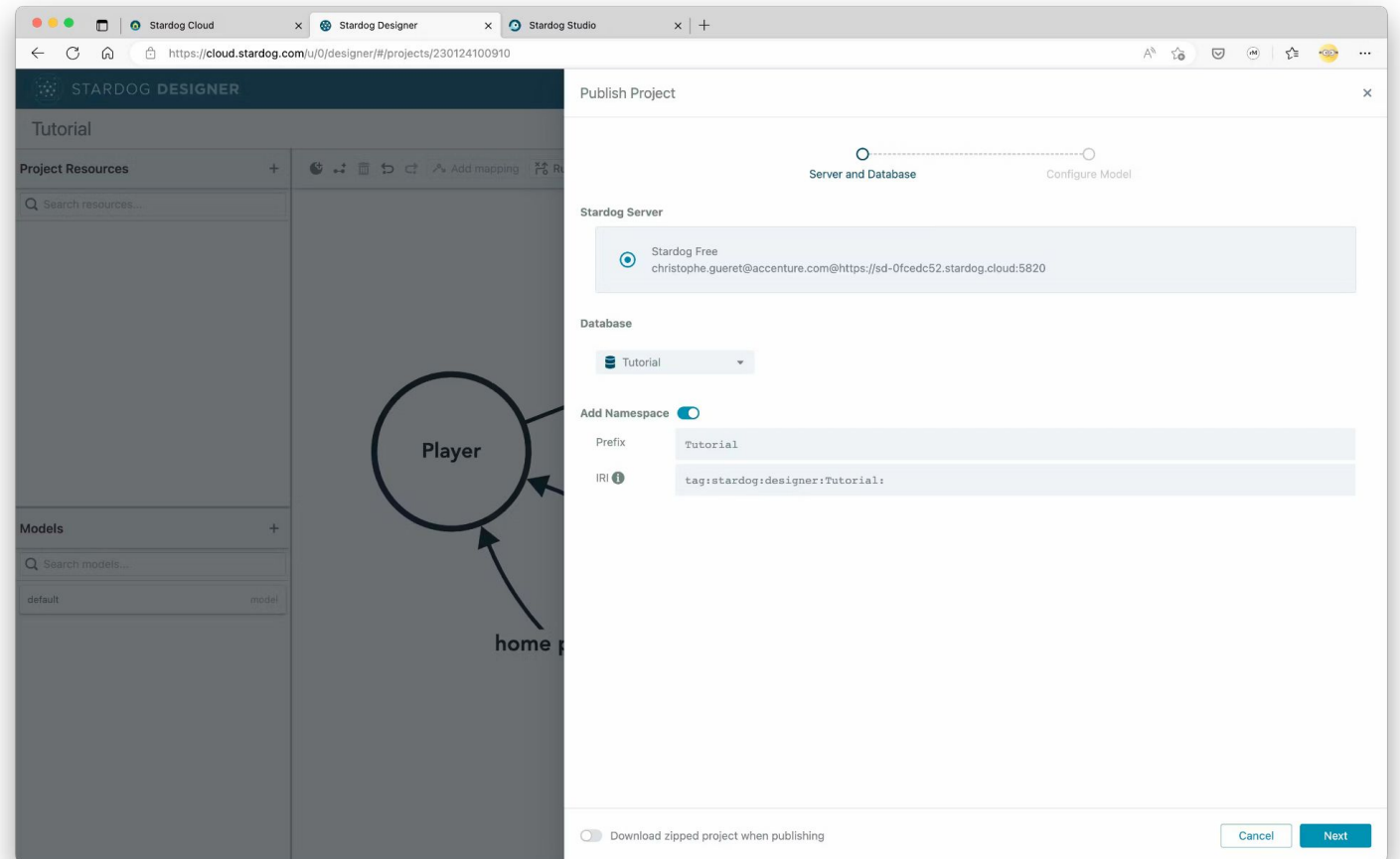
The end result is two rules expressing “plays for” for both the away and home team context



Publishing the ontology

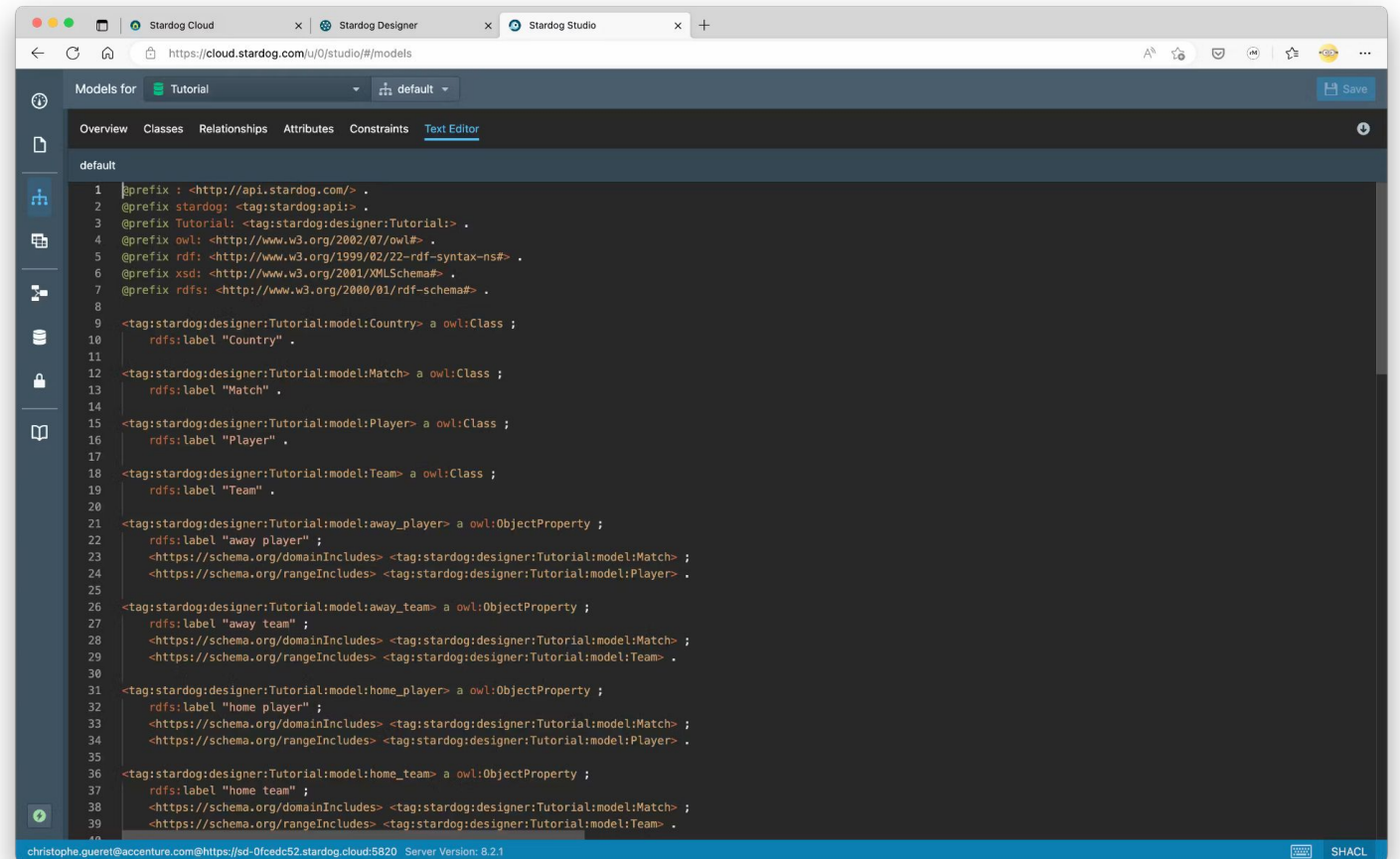
Export to Stardog

All the work done in Designer can now be exported to the server



Looking at the work done in Studio

If we switch to Studio we can see how many triples were created so far and see them in the text editor of the ontology



The screenshot shows the Stardog Studio web interface. The browser tabs include 'Stardog Cloud', 'Stardog Designer', and 'Stardog Studio'. The URL is <https://cloud.stardog.com/lu/0/studio/#/models>. The interface displays the 'Models for Tutorial' page with a 'Text Editor' tab selected. The editor shows the following RDF ontology code:

```
1 @prefix : <http://api.stardog.com/> .
2 @prefix stardog: <tag:stardog:api> .
3 @prefix Tutorial: <tag:stardog:designer:Tutorial:> .
4 @prefix owl: <http://www.w3.org/2002/07/owl#> .
5 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
6 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
7 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
8
9 <tag:stardog:designer:Tutorial:model:Country> a owl:Class ;
10   rdfs:label "Country" .
11
12 <tag:stardog:designer:Tutorial:model:Match> a owl:Class ;
13   rdfs:label "Match" .
14
15 <tag:stardog:designer:Tutorial:model:Player> a owl:Class ;
16   rdfs:label "Player" .
17
18 <tag:stardog:designer:Tutorial:model:Team> a owl:Class ;
19   rdfs:label "Team" .
20
21 <tag:stardog:designer:Tutorial:model:away_player> a owl:ObjectProperty ;
22   rdfs:label "away player" ;
23   <https://schema.org/domainIncludes> <tag:stardog:designer:Tutorial:model:Match> ;
24   <https://schema.org/rangeIncludes> <tag:stardog:designer:Tutorial:model:Player> .
25
26 <tag:stardog:designer:Tutorial:model:away_team> a owl:ObjectProperty ;
27   rdfs:label "away team" ;
28   <https://schema.org/domainIncludes> <tag:stardog:designer:Tutorial:model:Match> ;
29   <https://schema.org/rangeIncludes> <tag:stardog:designer:Tutorial:model:Team> .
30
31 <tag:stardog:designer:Tutorial:model:home_player> a owl:ObjectProperty ;
32   rdfs:label "home player" ;
33   <https://schema.org/domainIncludes> <tag:stardog:designer:Tutorial:model:Match> ;
34   <https://schema.org/rangeIncludes> <tag:stardog:designer:Tutorial:model:Player> .
35
36 <tag:stardog:designer:Tutorial:model:home_team> a owl:ObjectProperty ;
37   rdfs:label "home team" ;
38   <https://schema.org/domainIncludes> <tag:stardog:designer:Tutorial:model:Match> ;
39   <https://schema.org/rangeIncludes> <tag:stardog:designer:Tutorial:model:Team> .
40
```

The bottom status bar shows the user email 'christophe.gueret@accenture.com' and the server version '8.2.1'.



Adding some data

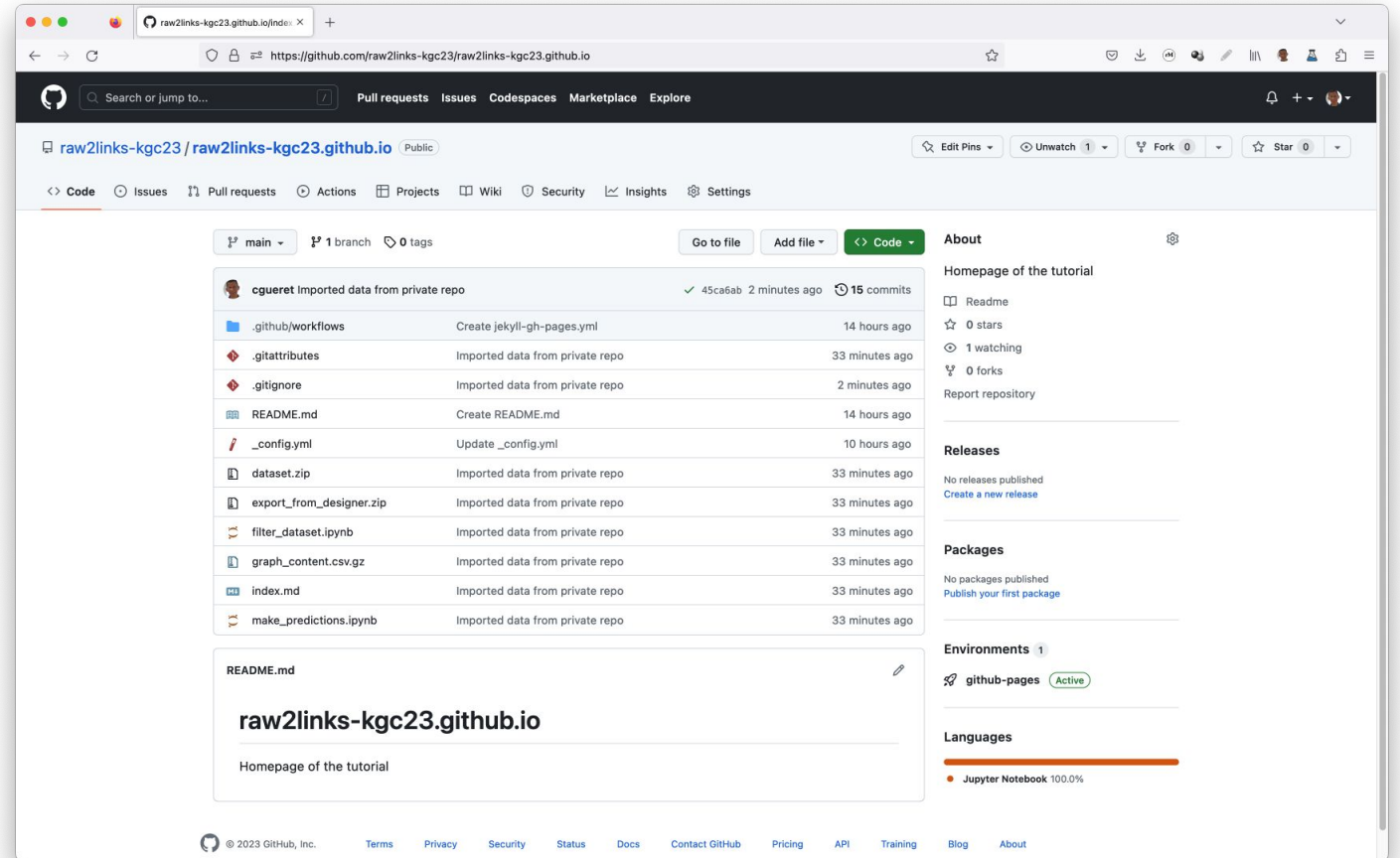
Download data

Go to:

<https://raw2links-kgc23.github.io/>

And download:

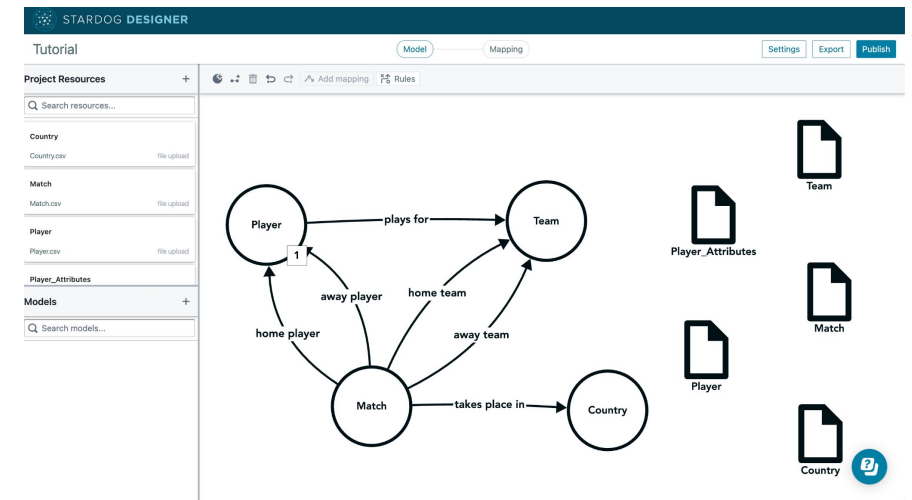
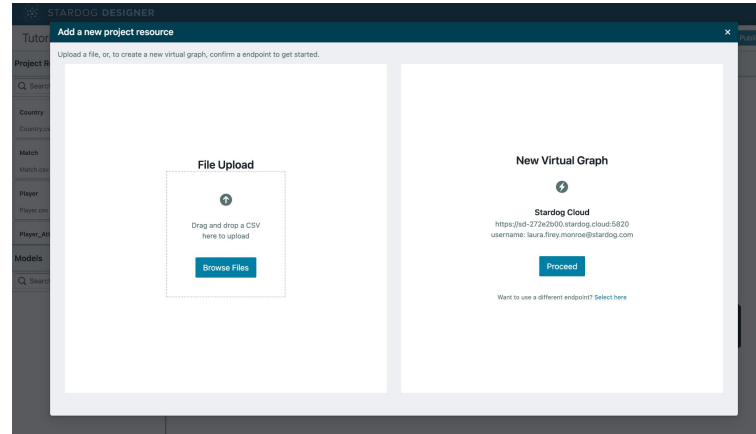
dataset.zip



Upload data

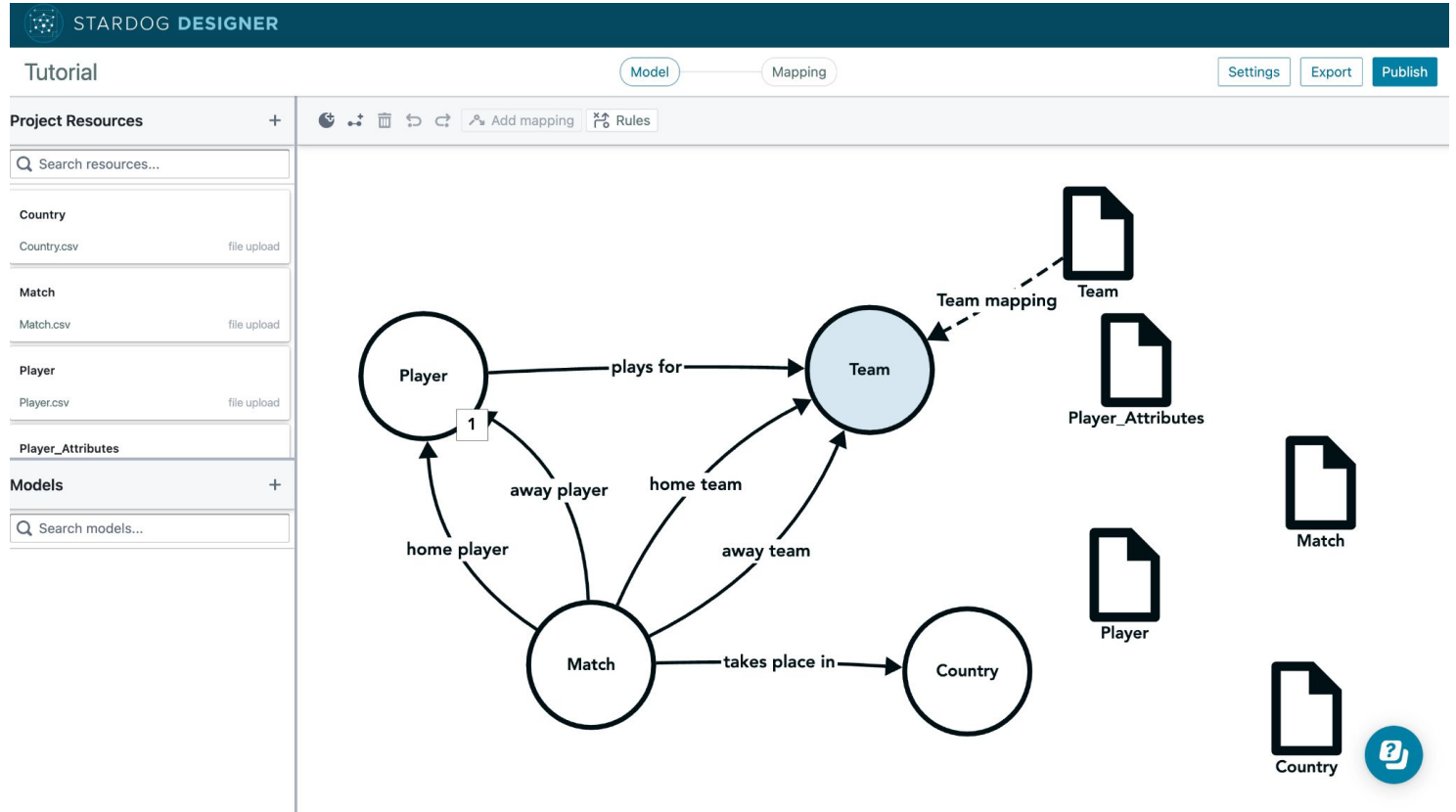
Upload 5 CSVs

- Country.csv
- Match.csv
- Player_Attributes.csv
- Player.csv
- Team.csv



Map data

- Mapping data can be done in variety of ways through Designer
 - In the canvas



Map data

- Mapping data can be done in variety of ways through Designer
 - In the canvas
 - Through suggestions

The screenshot shows the Stardog Designer interface in the 'Mapping' tab. The main area displays a mapping table for the 'Country - Country' model. The table has columns for 'Country' (Type: CSV), 'Score', and 'Action'. The 'Country' column is expanded to show two rows: 'Primary Identifier' with 'id' and a 'HIGH' score, and 'Label' with 'name' and a 'LOW' score. Below the table, there are sections for 'Relationships' (showing 'takes place in' with a 'Source Class' dropdown) and 'Attribute'.

Country	Score	Action
Primary Identifier	HIGH	✓ ✗
Label	LOW	✓ ✗

Map data

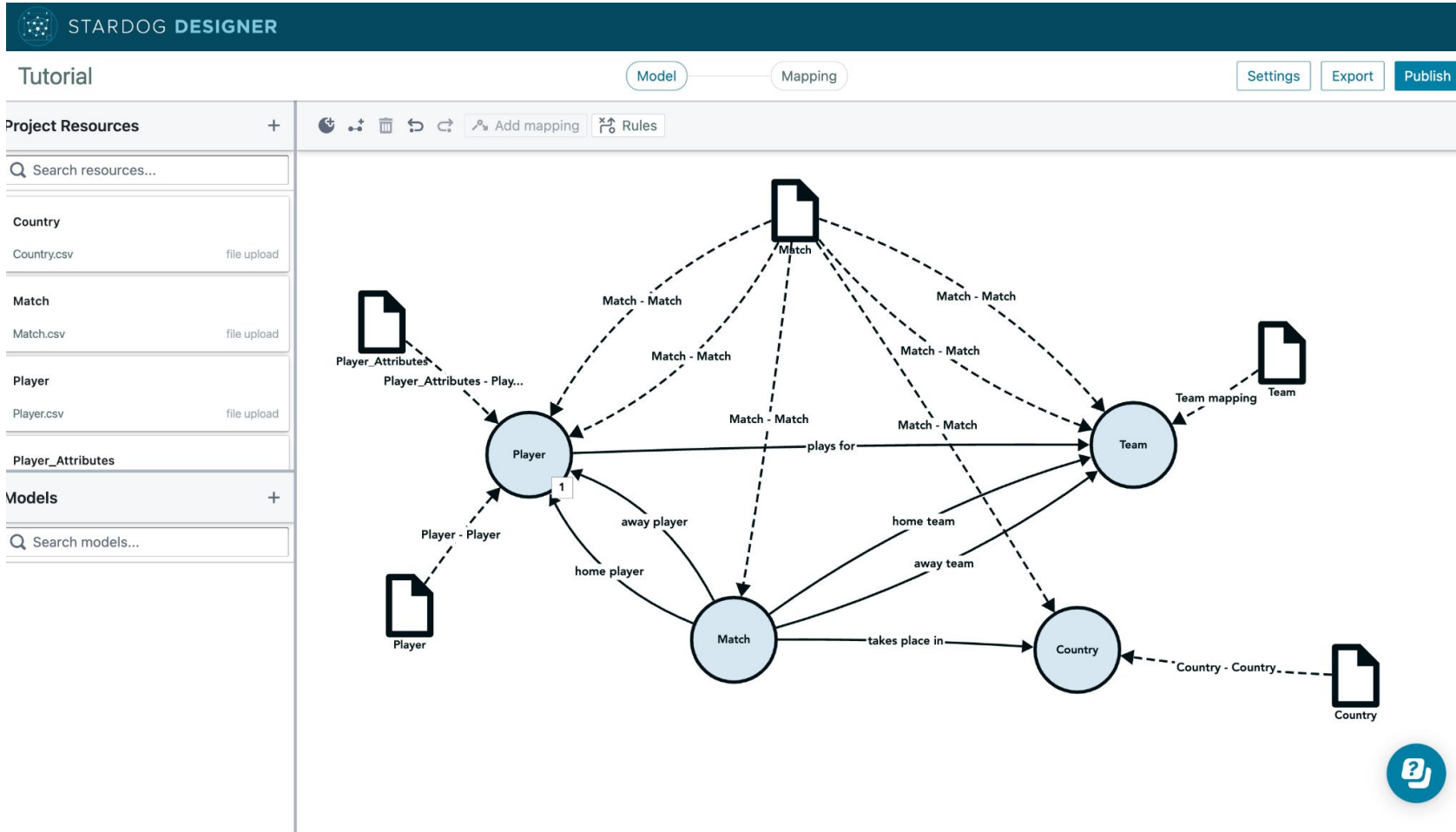
- Mapping data can be done in variety of ways through Designer
 - In the canvas
 - Through suggestions

Don't forget to also map your relationships!

Match - Match Match	Match Type: CSV
▼ Relationships	
↔ away player	Target Class: Player Define Target Identifier in: <input checked="" type="radio"/> New mapping <input type="radio"/> Existing mapping away_player_id x Preview: 33863
↔ away team	Target Class: Team Define Target Identifier in: <input checked="" type="radio"/> New mapping <input type="radio"/> Existing mapping away_team_id x Preview: 8472
↔ home player	Target Class: Player Define Target Identifier in: <input checked="" type="radio"/> New mapping <input type="radio"/> Existing mapping home_player_id x Preview: 31013
↔ home team	Target Class: Team Define Target Identifier in: <input checked="" type="radio"/> New mapping <input type="radio"/> Existing mapping home_player_id x Preview: 31013
↔ takes place in	Target Class: Country Define Target Identifier in: <input checked="" type="radio"/> New mapping <input type="radio"/> Existing mapping country_id x Preview: 1729



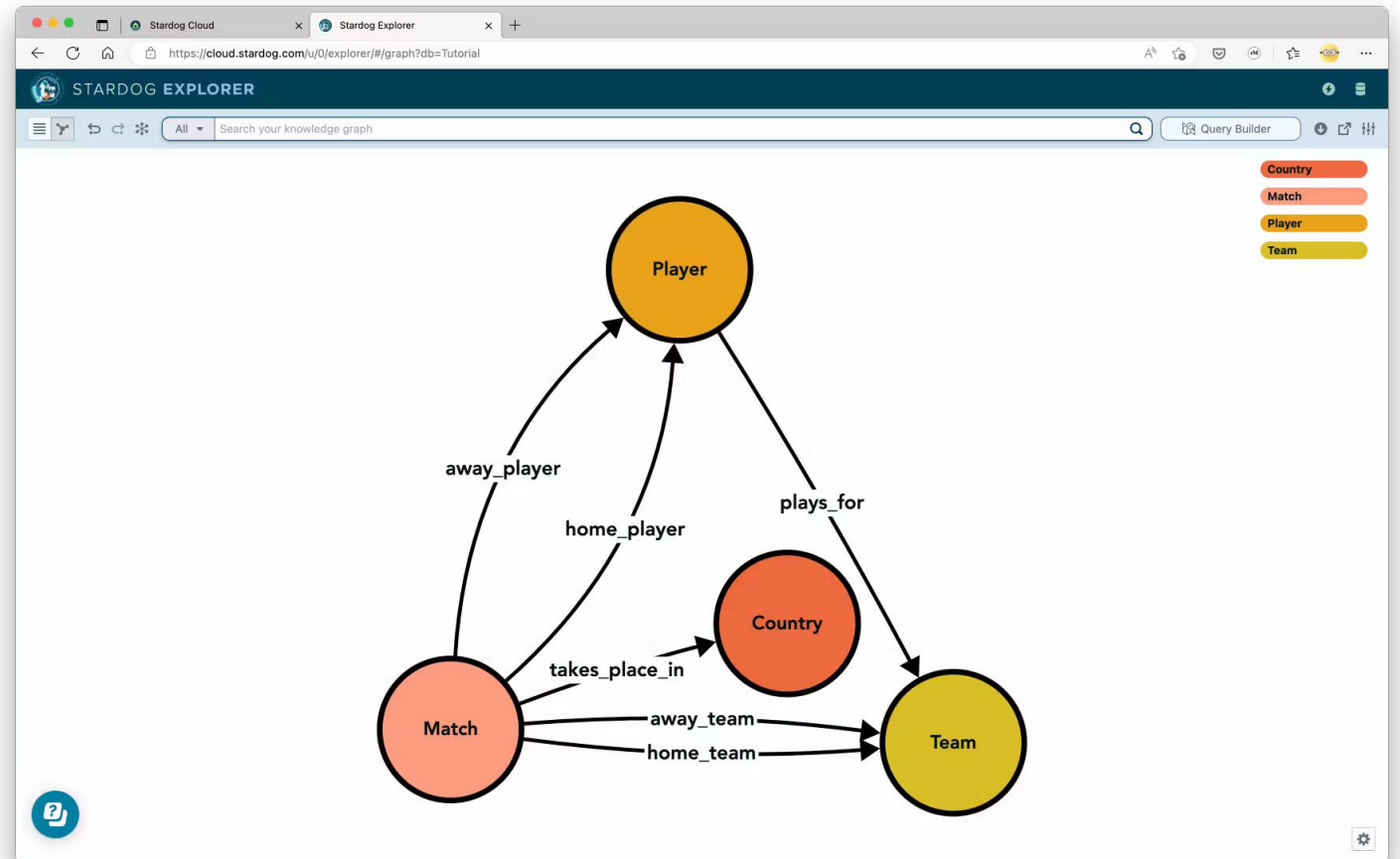
End result



Playing with the graph

Explore the data in Explorer

We can visualize the ontology and explore from there



Nodes properties

Clicking on a node gives access to a panel showing all the ingoing and outgoing edges for it

If the node is a concept we find a list of instances for it and the rest of the ontology predicates

The screenshot shows the Stardog Explorer interface. On the left, a graph visualization shows a 'Match' node connected to a 'Player' node via edges labeled 'away_player', 'home_player', and 'takes_play'. The 'Match' node also has edges labeled 'away' and 'home'. On the right, a panel titled 'Player' is open, showing a list of instances and outgoing edges.

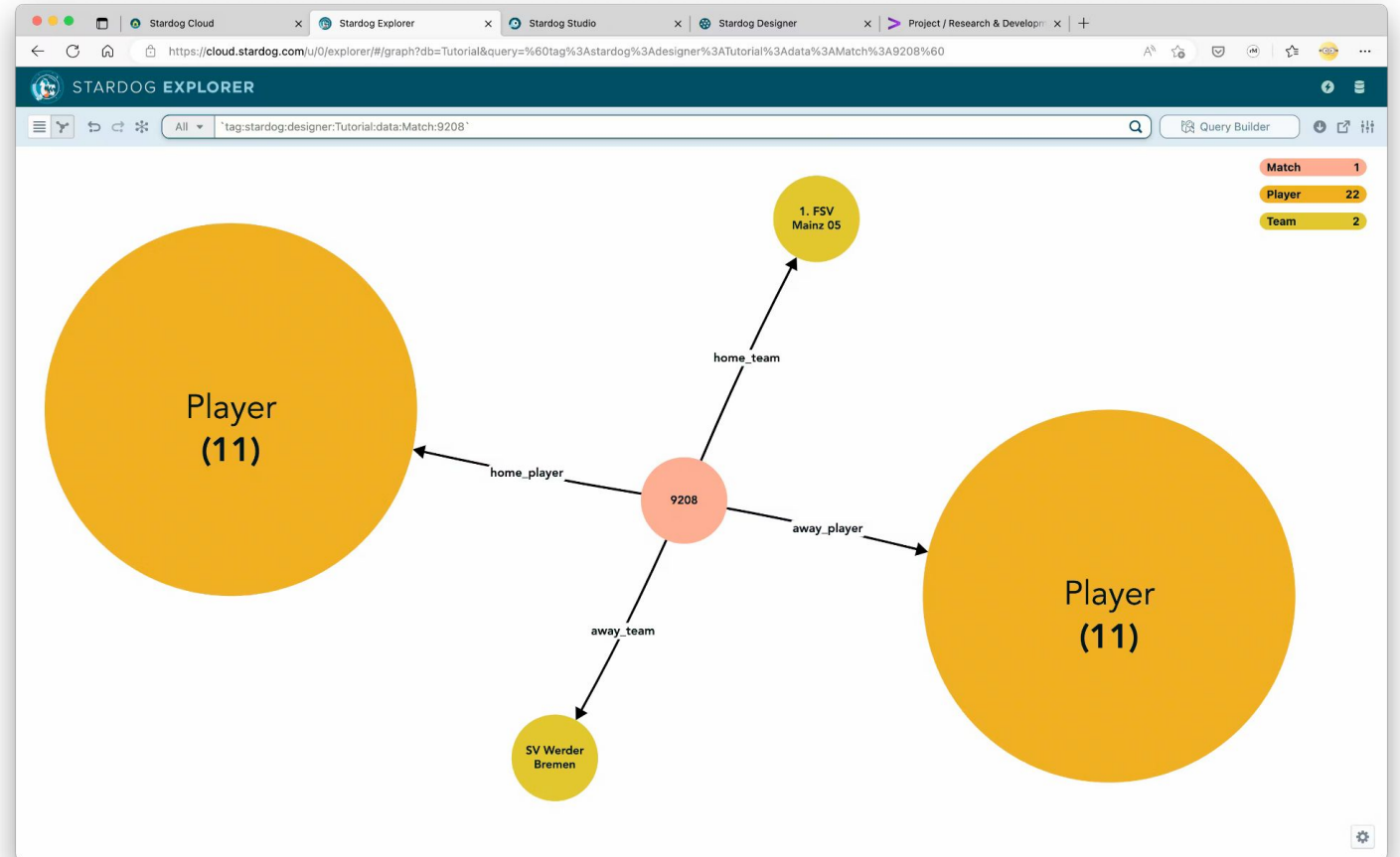
Player	
Instances	Aaron Appindangoye Aaron Cresswell Aaron Doran Aaron Galindo Aaron Hughes Aaron Hunt Aaron Kuhl Aaron Lennon Aaron Lennox Aaron Meijers
← away_player	Match
← home_player	Match
→ label	Player
→ plays_for	Team



Example of instance of Match

A match connects to two sets of 11 players as expected, as well as two teams (one home, one away)

This kind of manual validation could be automated using SHACL shapes but we will not get into that during this tutorial



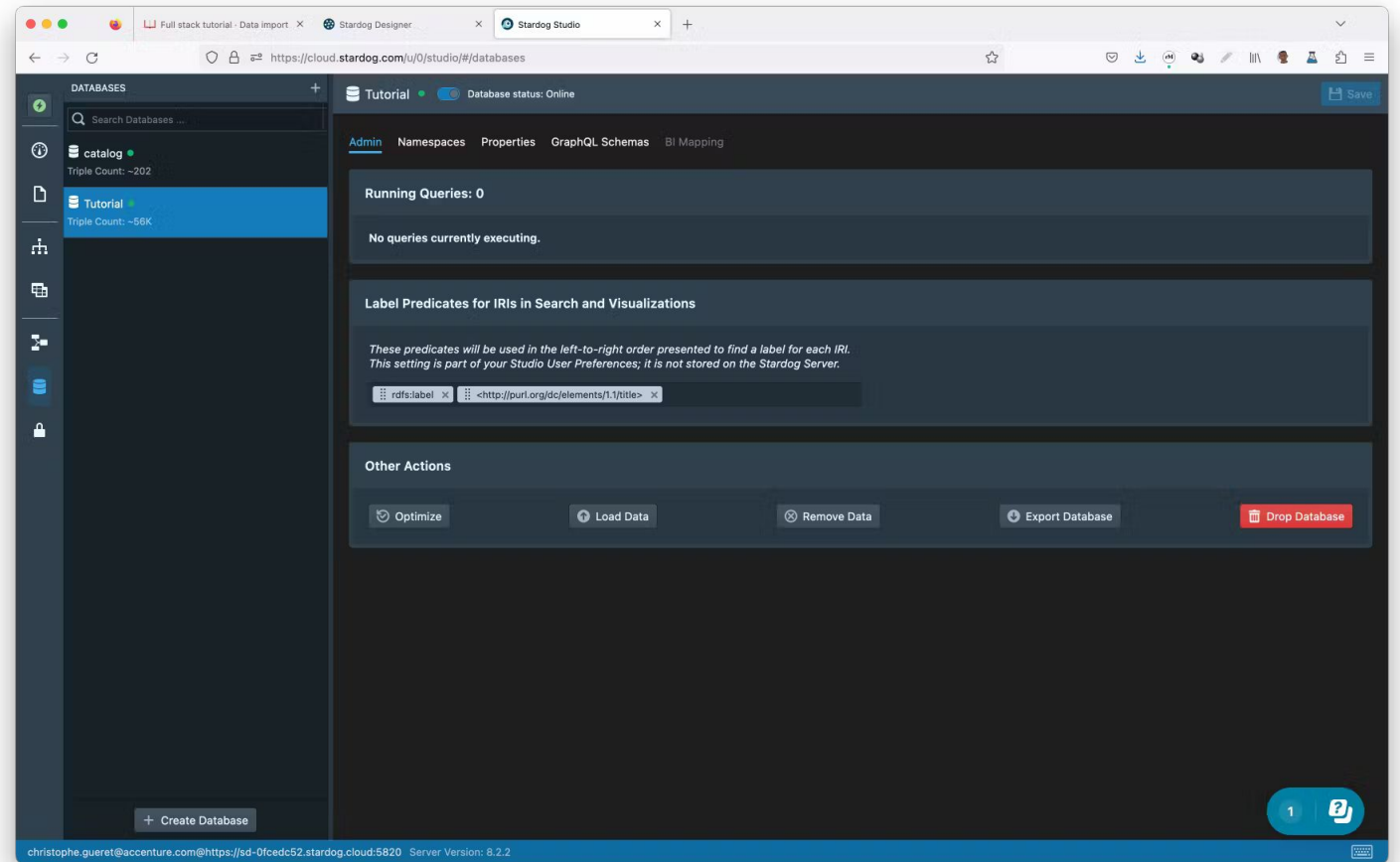
More advanced queries

Explorer allows for more advanced investigations via the visual query builder

The screenshot displays the Stardog Explorer interface. On the left, the 'Query Builder' window shows a graph query with nodes 'Country', 'Match', and 'Team'. The relationships are 'takes_place_in' (Country to Match) and 'away_team' (Match to Team). Below the query builder are 'AND' and 'OR' operators. On the right, a graph visualization shows a network of nodes and edges. A legend on the right side of the graph shows: Country (5), Match (1,762), and Team (98). At the bottom of the interface, there is an 'Expand graph results' toggle, a 'Clear' button, and a 'Search' button.

Switching to Studio

We can go to Studio to query the data



Simple query

A simple SPARQL query is a good start to see the kind of triple we have in the database

The screenshot shows the Stardog Studio interface. The query editor contains the following SPARQL query:

```
1 SELECT ?s ?p ?o WHERE {?s ?p ?o} LIMIT 10
2
3
4
```

The results pane shows 10 results in a table with columns s, p, and o. The results are as follows:

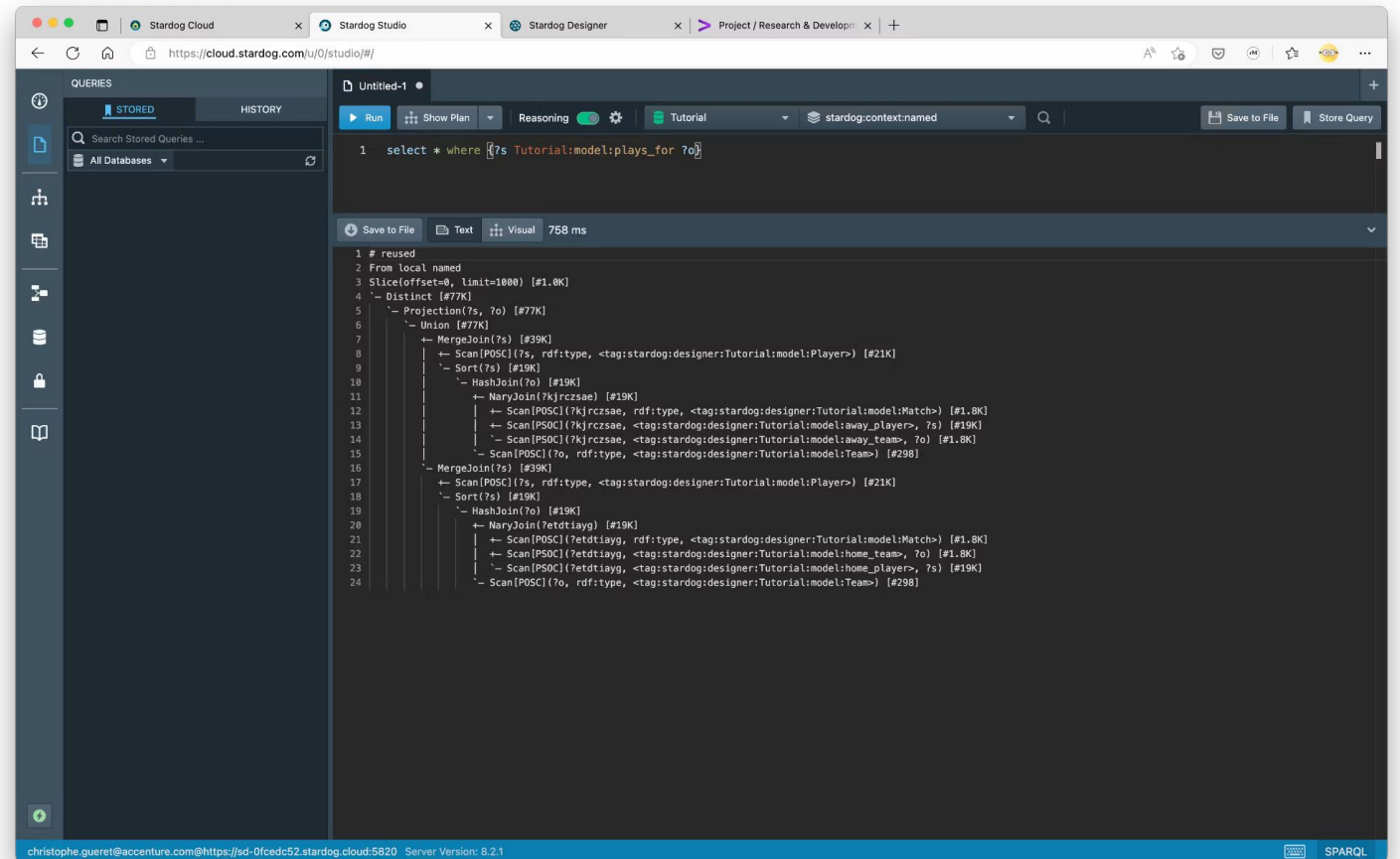
s	p	o
Tutorial:model	https://schema.org/datePublished	2023-01-24T18:01:21.909Z
Tutorial:data:Country	https://schema.org/datePublished	2023-01-24T18:01:21.909Z
Tutorial:data:Team	https://schema.org/datePublished	2023-01-24T18:01:21.909Z
Tutorial:data:Player	https://schema.org/datePublished	2023-01-24T18:01:21.909Z
Tutorial:data:Match	https://schema.org/datePublished	2023-01-24T18:01:21.909Z
Tutorial:data:Player_Attributes	https://schema.org/datePublished	2023-01-24T18:01:21.909Z
Tutorial:model:away_player	https://schema.org/rangeIncludes	Tutorial:model:Player
Tutorial:model:away_team	https://schema.org/rangeIncludes	Tutorial:model:Team
Tutorial:model:home_player	https://schema.org/rangeIncludes	Tutorial:model:Player
Tutorial:model:home_team	https://schema.org/rangeIncludes	Tutorial:model:Team

The interface also shows a sidebar with navigation icons, a top navigation bar with tabs for 'Stardog Cloud', 'Stardog Studio', and 'Stardog Explorer', and a bottom status bar with the user email 'christophe.gueret@accenture.com' and server version '8.2.1'.

Reasoning as query rewrite

Showing the query plan provides a view into what is happening under the hood for the rules we created in Designer

Their content is not executed to create the triples of the “THEN” part. Instead, these and the “IF” part are used to rewrite the query into the combination of condition triples



Extracting the graph

Figure out a target ontology

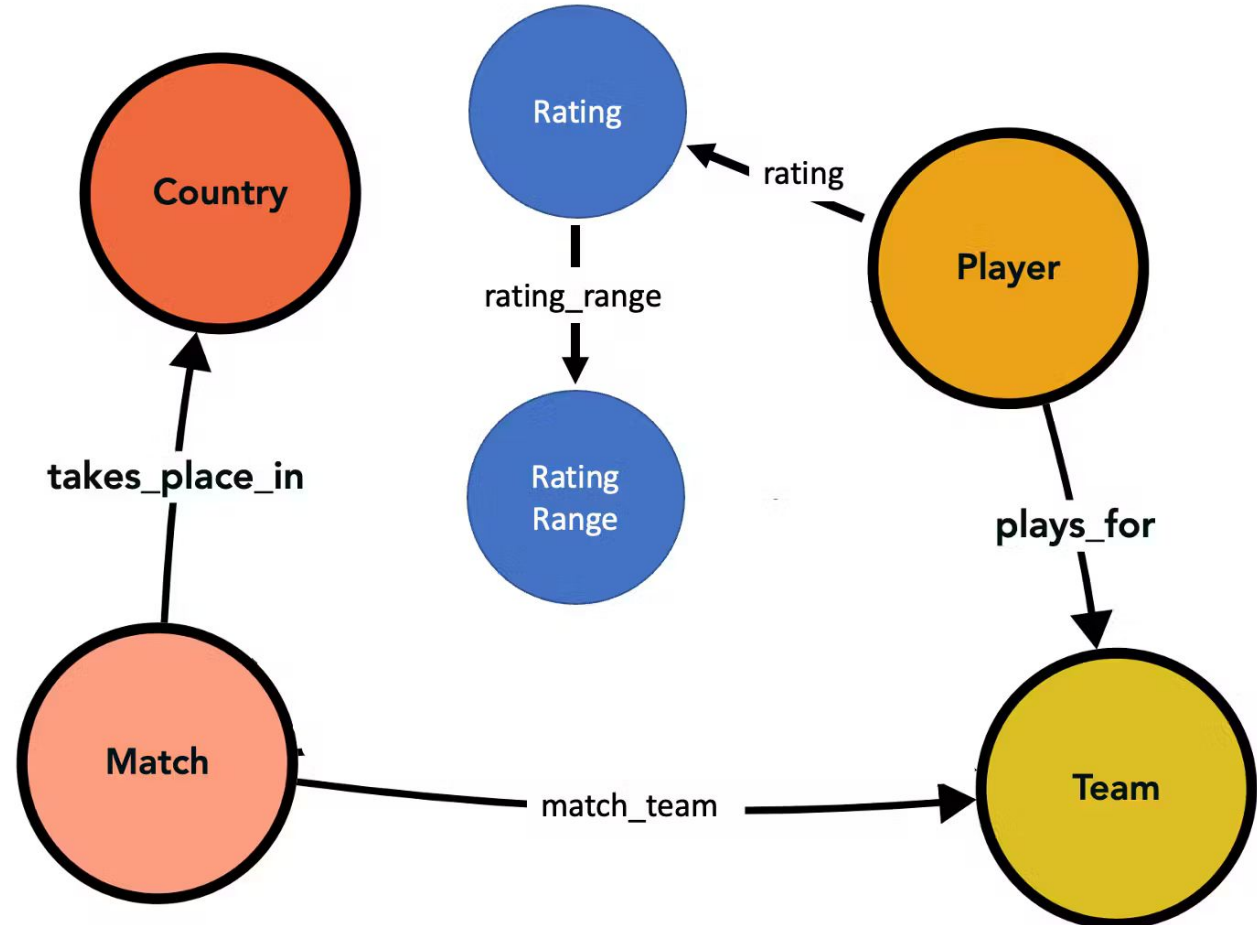
We need to design a graph optimised from machine learning

Right now Ampligraph would not know how to use the rating as an attribute.

We define a re-write of the graph with the following changes:

- Turn the rating attributes into resources
- Bucket the rating into ranges

We also simplify the relation “team”



Extract triples as is

The easiest is the triples we extract as is. One way to do this is to construct a set of triples and filter them on the predicates we want to keep

```
CONSTRUCT {?s ?p ?o}
WHERE {
  ?s ?p ?o
  VALUES ?p {Tutorial:model:takes_place_in Tutorial:model:plays_for}
}
```

Simplify a relationship

We construct a set of triples based on the ones we target, the trick is to change the predicate before it is returned

```
CONSTRUCT {?s ?p ?o}
WHERE {
  BIND (Tutorial:model:match_team AS ?p)
  ?s ?r ?o
  VALUES ?r {Tutorial:model:away_team Tutorial:model:home_team}
}
```


Turn a literal into a node

Simply using a cast to a URI we can turn a literal into a URI

It's important here not to forget to prefix the value with a distinct string (the range of the predicate is generally a good option) in order to avoid collisions

```
CONSTRUCT {?s ?p ?o}
WHERE {
  BIND (Tutorial:model:rating AS ?p)
  ?s ?p ?r
  BIND (URI(CONCAT("rating_",STR(?r))) AS ?o)
}
```

Adding buckets

Ampligraph has no notion of values for rating so the node rating_1 will be as similar as rating_2 as it is to rating_80

To help the machine learning machinery we create buckets to group values by some proximity criteria

In other contexts predicates like skos:broader can be used with the same aim

```
CONSTRUCT {?s ?p ?o}
WHERE {
    ?player Tutorial:model:rating ?r
    BIND (URI(CONCAT("rating_",STR(?r))) AS ?s)
    BIND (Tutorial:model:rating_bucket AS ?p)
    BIND ((INTEGER(FLOAT(?r) / 10.0)) AS ?r_bucket)
    BIND (URI(CONCAT("bucket_",STR(?r_bucket))) AS ?o)
}
```

Merge it all back together

The last step of the construction of the graph is to merge all those extractions together as one query. “UNION” is the perfect keyword to achieve that

In order to export the graph as a CSV file, which we will input into Ampligraph, we replace the “CONSTRUCT” by a “SELECT”

```
SELECT DISTINCT ?s ?p ?o
{
  {
    ?s ?p ?o
    VALUES ?p {Tutorial:model:takes_place_in Tutorial:model:plays_for}
  }
  UNION {
    BIND (Tutorial:model:match_team AS ?p)
    ?s ?r ?o
    VALUES ?r {Tutorial:model:away_team Tutorial:model:home_team}
  }
  UNION {
    BIND (Tutorial:model:rating AS ?p)
    ?s ?p ?r
    BIND (URI(CONCAT("rating_",STR(?r))) AS ?o)
  }
  UNION {
    ?player Tutorial:model:rating ?r
    BIND (URI(CONCAT("rating_",STR(?r))) AS ?s)
    BIND (Tutorial:model:rating_bucket AS ?p)
    BIND ((INTEGER(FLOAT(?r) / 10.0)) AS ?r_bucket)
    BIND (URI(CONCAT("bucket_",STR(?r_bucket))) AS ?o)
  }
}
```

Running predictions

Getting started

Open a Jupyter notebook somewhere (Google Colab, local machine, ...) and import pandas as well as Ampligraph

```
%%capture  
!pip install ampligraph  
import pandas as pd  
import ampligraph
```

You can verify that all went fine by printing out the version number.

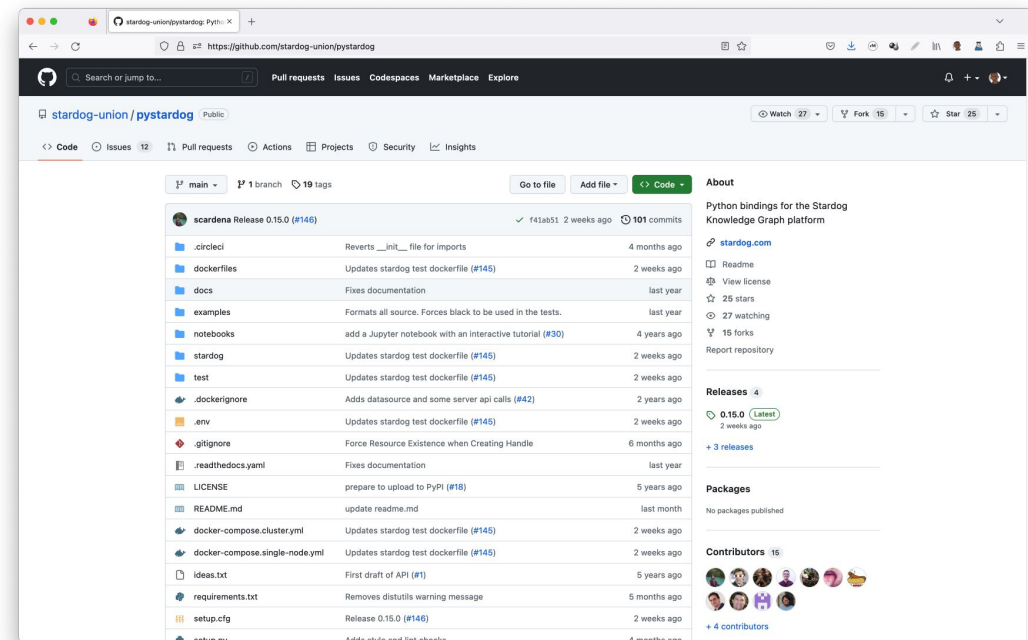
```
print(ampligraph.__version__)
```

Load the data

Load the data we just exported from Stardog

PS: we apply here a distinct split (export/save/load) but it could also be possible to instead use PyStardog to directly connect to Stardog from the notebook and run the SPARQL query from it

```
df = pd.read_csv("graph_content.csv.gz").dropna()  
df.head()
```



Preparing a training dataset

In a common ML pattern, we first prepare a train and validation set to train the ML and validate if something has actually been learnt

```
from ampligraph.evaluation import train_test_split_no_unseen
test_size = int(df.shape[0]*0.10)
X_train, X_valid = train_test_split_no_unseen(np.array(df),
test_size=test_size)
print('Train set size: ', X_train.shape)
print('Test set size: ', X_valid.shape)
```

Compile and train the model

The KGE models available in Ampligraph are defined as a combination of three blocks:

- A regularizer
- A loss function
- An optimizer

For each of them there are several options available, the code proposed here proposes a set experimentally found to work well for this tutorial

```
from ampligraph.latent_features import ScoringBasedEmbeddingModel
from ampligraph.latent_features.regularizers import get as
get_regularizer
from ampligraph.latent_features.loss_functions import get as get_loss
import tensorflow as tf

# create the model
model = ScoringBasedEmbeddingModel(k=300, eta=20,
scoring_type='Complex', seed=0)

# create regularizer, loss function and optimizer
reg = get_regularizer('LP', hyperparams={'p':3, 'lambda':1e-3})
loss = get_loss('pairwise', hyperparams={'margin':5})
optim = tf.keras.optimizers.Adam(learning_rate = 1e-4)

# compile the model
model.compile(loss=loss, optimizer=optim,
entity_relation_regularizer=reg)
model.fit(X_train, batch_size=5000, epochs=500)
```


Validate the model

Once the model is trained we can validate its learning using some common metrics

The learning goal is for the model to be able to make the difference between a fact in the graph and a fact that is not in the graph:

- MR(R) looks at the predictive performance between the two
- Hits@N is “if we generate N triples at random, what is the percentage of them which are in the graph?”

```
from ampligraph.evaluation import mr_score, mrr_score, hits_at_n_score
ranks = model.evaluate(X_valid,
                       batch_size=5000,
                       use_filter={'train': X_train, 'test': X_valid},
                       verbose=True)

print('MR:', mr_score(ranks))
print('MRR:', mrr_score(ranks))
print('Hits@1:', hits_at_n_score(ranks, 1))
print('Hits@3:', hits_at_n_score(ranks, 3))
print('Hits@10:', hits_at_n_score(ranks, 10))
print('Hits@100:', hits_at_n_score(ranks, 100))
```

```
MR: 445.92871287128713
MRR: 0.22918282737102727
Hits@1: 0.18663366336633663
Hits@3: 0.22574257425742575
Hits@10: 0.32425742574257427
Hits@100: 0.5272277227722773
```



Creating the triples to test

Using the set of players and the set of teams we can create all the triples to test

Every single of those triple is an hypothesis: an edge we think may make sense and of which the AI model will assess the likelihood

```
players = set(df[df['s'].str.contains('Player:')] ['s'].values) | \
            set(df[df['o'].str.contains('Player:')] ['o'].values)
teams = set(df[df['s'].str.contains('Team:')] ['s'].values) | \
        set(df[df['o'].str.contains('Team:')] ['o'].values)
print('Number of players: ', len(players))
print('Number of teams: ', len(teams))
```

```
pairs = []
for player in players:
    for team in teams:
        pairs.append([player,
                    'tag:stardog:designer:Tutorial:model:plays_for', team])
df_to_test = pd.DataFrame(pairs, columns=['s', 'p', 'o'])
df_to_test.head()
```

Getting the result

Now that the set of hypothesis is ready we need to remove all the links that are already in the graph (unless we want to check that they indeed rank high) and ask for the prediction scores

The scores are not calibrated so they have to absolute meaning and can be only used to rank the list

```
df_to_test = pd.merge(df_to_test, df, indicator=True, how='outer')\
                .query('_merge=="left_only"').drop('_merge', axis=1)
```

```
scores = model.predict(np.array(df_to_test))
```

```
df_to_test['scores'] = scores
df_to_test.sort_values(by='scores', ascending=False)
```

Adding labels

Adding some labels

URIs are not very intuitive nor convenient to read and they are not expected to be so

To make the results more user friendly we first need to connect to Stardog

```
{  
  "database": "YOUR DATABASE (e.g. Tutorial)",  
  "endpoint": "YOUR ENDPOINT (e.g. https://sd-  
0fcedc52.stardog.cloud:5820)",  
  "username": "YOUR USERNAME (e.g. christophe.gueret@accenture.com)",  
  "password": "YOUR PASSWORD (e.g. Passw0rd)",  
}
```

```
connection_details = json.load(open('connection_details.json'))  
conn = stardog.Connection(**connection_details)
```

Get the labels

Once connected it is possible to extract with a SPARQL query all the labels matching the URIs

```
query = """
SELECT ?identifier ?label FROM stardog:context:all WHERE {
  ?identifier rdfs:label ?label .
}
"""

csv_results = conn.select(query, content_type='text/csv')
df_labels =
pd.read_csv(io.BytesIO(csv_results)).set_index('identifier')
labels = df_labels.to_dict()['label']
print ('Loaded {} labels'.format(len(labels)))
```

```
df_to_test.replace(labels)
```

Outline

Objective:

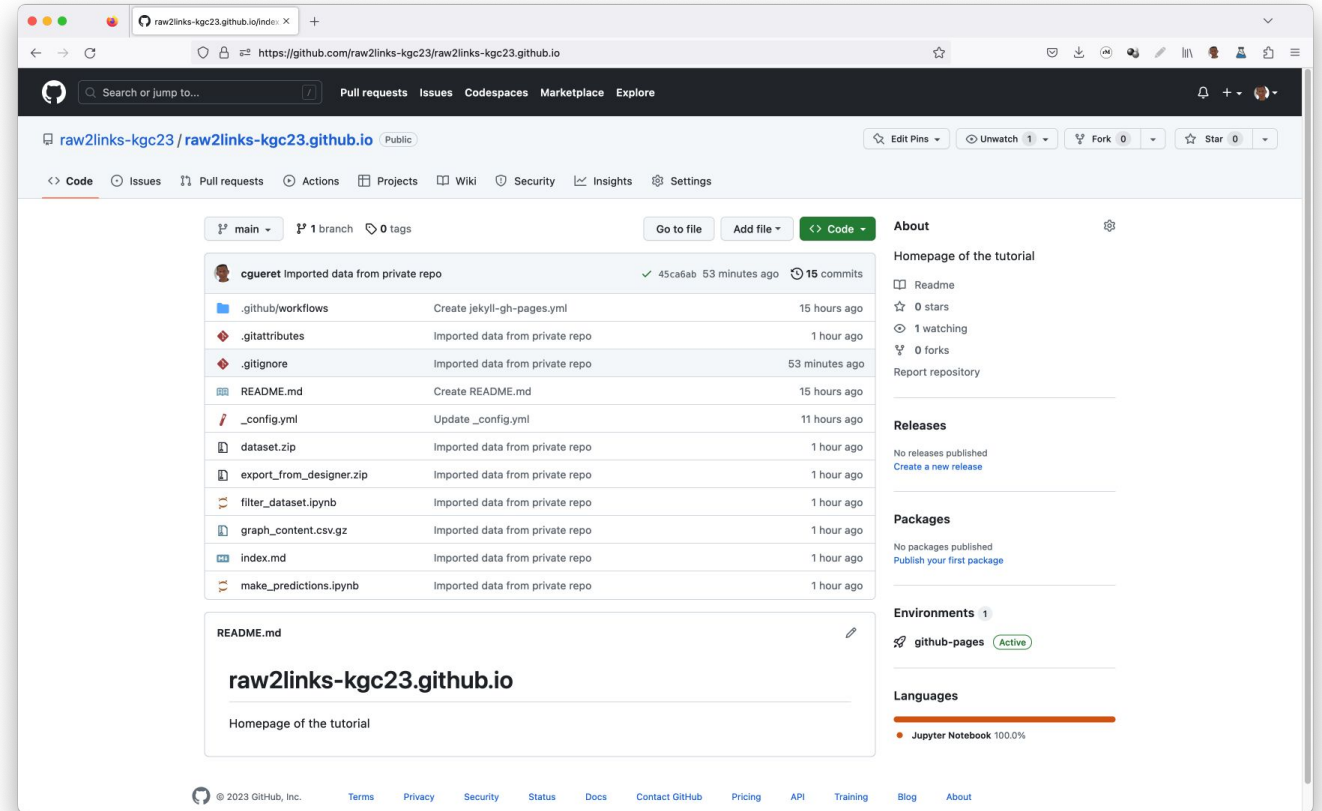
Answer questions you may still have and share your insights

- 1** Overview (15 min)
 - 1.2** Data integration
 - 1.3** Graph machine learning
- 2** Hands-on session (60 min)
 - 2.1** Putting a Knowledge Graph together
 - 2.2** Using graph machine learning
- 3** Closing and Q&A (15 min)

Share your results!

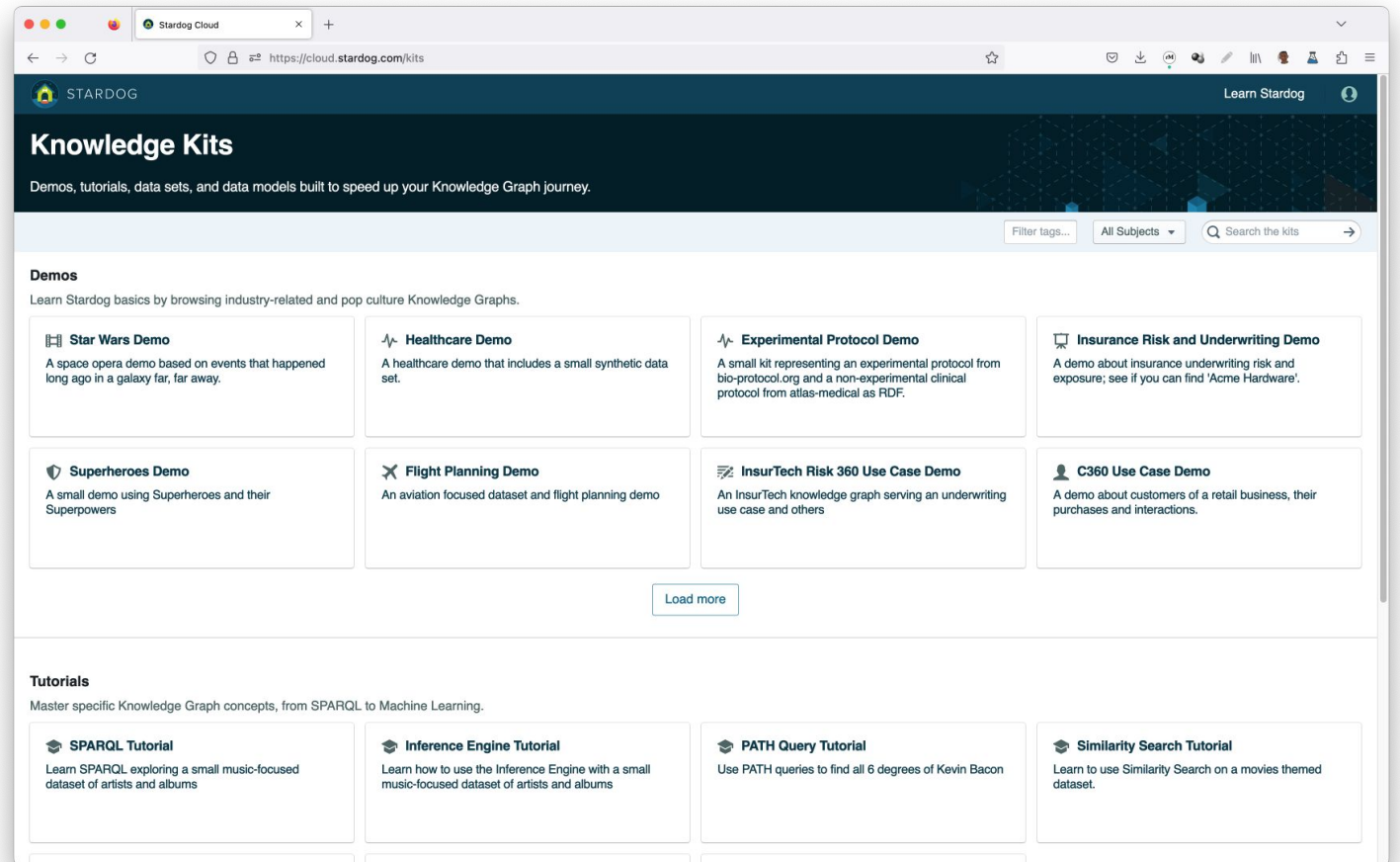
Use different random seeds and share them among yourselves and with us!

All the content of this tutorial is on github so you can fork the repository, propose PR, etc.



Why not test some Knowledge Kits?

To explore further the capabilities of Stardog and get inspiration for more use-cases, you can have a go at the many Knowledge Kits available via the menu in “Learn Stardog”



To infinity and beyond!

As you may have noticed, recent developments in the field of Large Language Models (LLMs) is rocketing AI capabilities into new horizons

Stardog recently introduced Voicebox to make what we just did in Designer even easier to do. And that is only the beginning so stay tuned for more features to come ;-)

