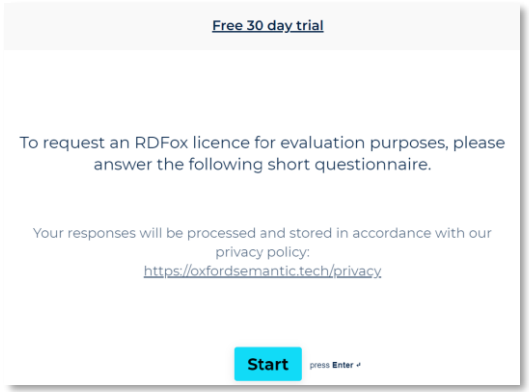
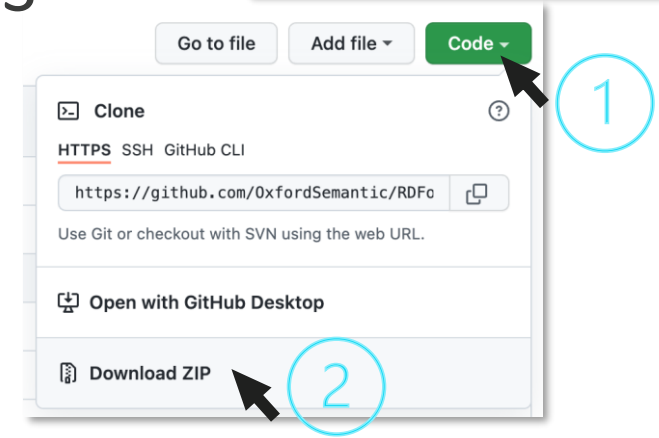
An Introduction to SPARQL

The world's most performant knowledge graph and semantic reasoning engine.

Requirements

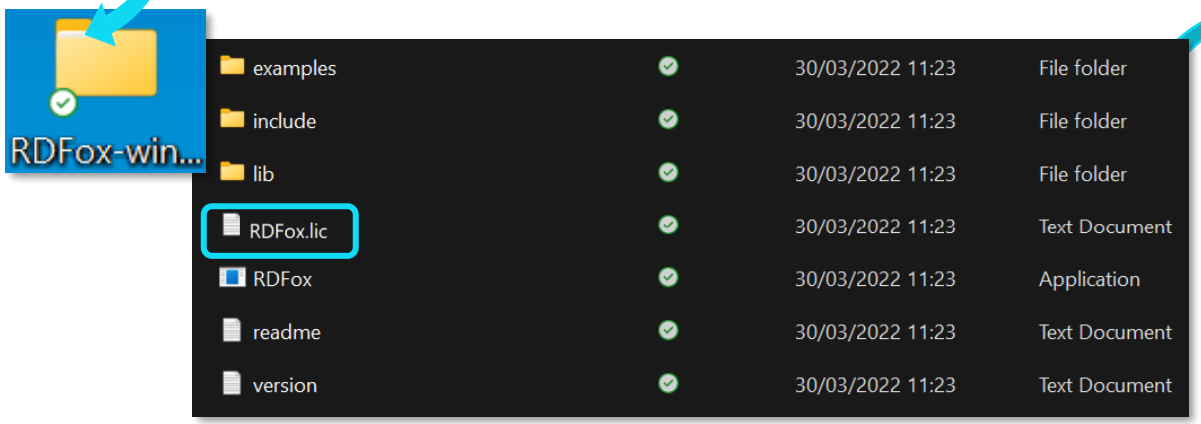
A. Get an RDFox License
   https://www.oxfordsemantic.tech/tryrdfoxforfree
B. Download RDFox (& unzip)
   https://www.oxfordsemantic.tech/downloads
C. Download the class materials from Github:
   https://github.com/OxfordSemantic/RDFoxWorkshop
D. *OPTIONAL* Get your IDE of choice ready (VS Code etc.)
   https://code.visualstudio.com/



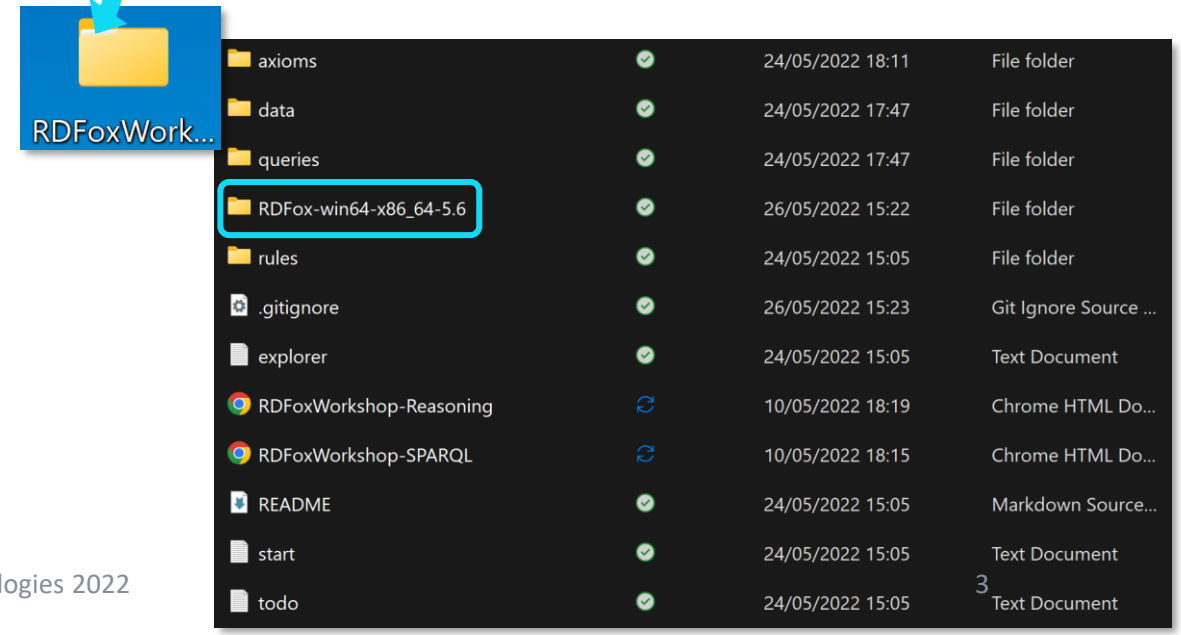


You should have…

1 Make sure to put your **license INSIDE** the **RDFox folder**.



2 Then drop your **RDFox folder INSIDE** the **Workshop folder**.



Without this setup you will need use different file paths in the commands we provide.

Objectives

Setting up RDFox

- ❑ License and executable
- ❑ IDE
- ❑ REST endpoint and UI

Loading data into RDFox

- ❑ Create datastore
- ❑ Import data

Exploring and querying with SPARQL

- ❑ Basics of SPARQL
- ❑ Useful queries to explore
- ❑ Aggregates
- ❑ Negation
- ❑ Filters
- ❑ Binds
- ❑ Optionals

Please feel free to ask questions at any time!

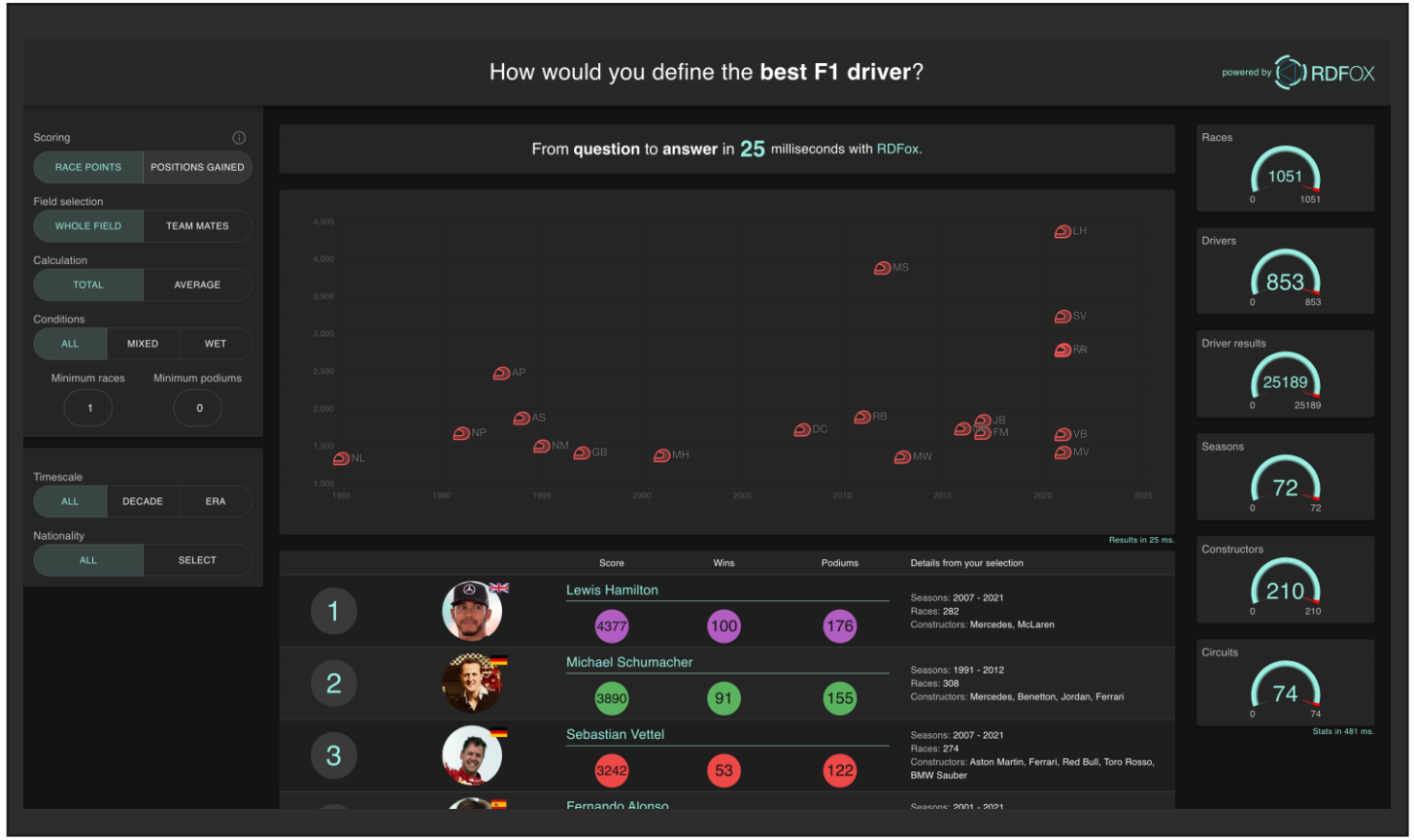
Exercises are scattered throughout.

Links are provided for reference.

https://docs.oxfordsemantic.tech/

Who is the Greatest Formula One Driver of All Time?

Controls and filters for the scoring system.



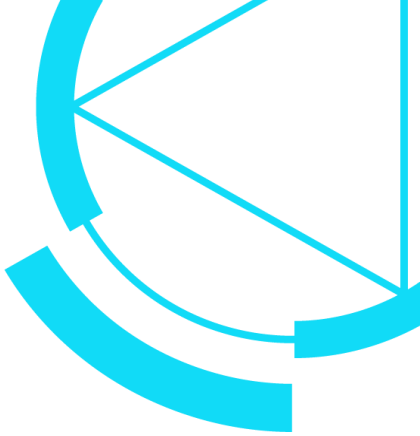
Statistics about the data used to form the results.

This matches the filters.

http://f1.rdfox.tech

Try it for yourself!

© Oxford Semantic Technologies 2022



Setting Up RDFox

Setting up RDFox

- ❑License and executable
- ❑IDE
- ❑REST endpoint and UI

Loading data into RDFox

- ❑Create datastore
- ❑Import data

Exploring and querying with SPARQL

- ❑Basics of SPARQL
- ❑Useful queries to explore
- ❑Aggregates
- ❑Negation
- ❑Filters

Setting up RDFox

- We recommend using an IDE (e.g. VS Code)
- Open a terminal, navigate to the workshop folder (or open it in VS Code)
  `cd <path_to_workshop_folder>`
- From there run:
  - MacOS ARM: `./RDFox-macOS-arm64-5.6/RDFox sandbox`
  - MacOS INTEL: `./RDFox-macOS-x86_64-5.6/RDFox sandbox`
  - Windows: `./RDFox-win64-x86_64-5.6/RDFox.exe sandbox`
- The RDFox server should now be running.

```
Source code for RDFox v1.0 Copyright 2013 Oxford University Innovation Limited and subsequent improvements Copyright 2017-2021 by Oxford Semantic Techn
ologies Limited.
This copy of RDFox is licensed for Developer use to Tom Vout (tom.vout@oxfordsemantic.tech) of OST until 07-Jun-2022 16:01:44
This system is equipped with 16.9 GB of RAM, and RDFox is configured to use at most 15.2 GB (89.9% of the total).
Currently, 2.8 GB (18.4% of the amount allocated to RDFox) appear to be available on the system.
Since RDFox is a RAM-based system, its performance can suffer when other running processes use a lot of memory.


A new server connection was opened as role 'guest' and stored with name 'sc1'.
>
```

The REST Endpoint and UI

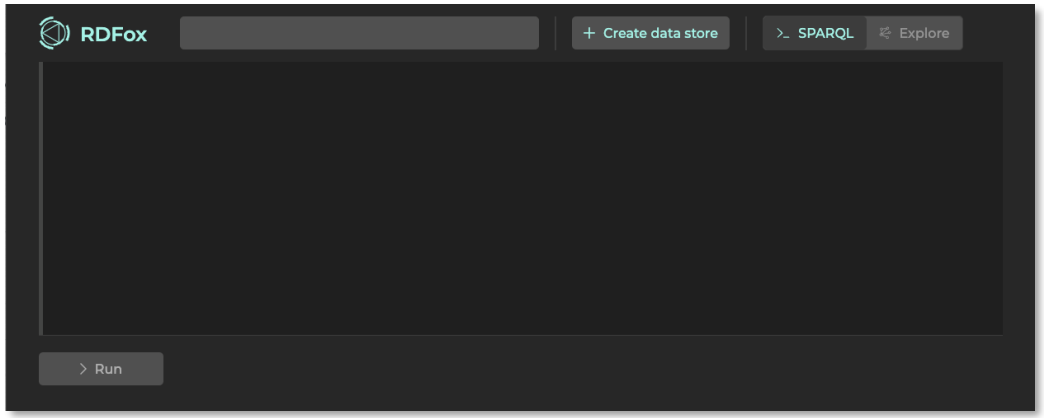
- In the terminal, run:
  `endpoint start`

```
> endpoint start

WARNING: The RDFox endpoint is running with no transport layer security (TLS). This could allow attackers to
steal information including role passwords.
         See the endpoint.channel variable and related variables in the description of the RDFox endpoint for
 details of how to set up TLS.

The REST endpoint was successfully started at port number/service name 12110 with 7 threads.
>
```

- The RDFox endpoint is now running, so we can use the web UI
- Open a browser and go to: localhost:12110/console
- This will show an empty console



https://docs.oxfordsemantic.tech/command-line-reference.html#endpoint

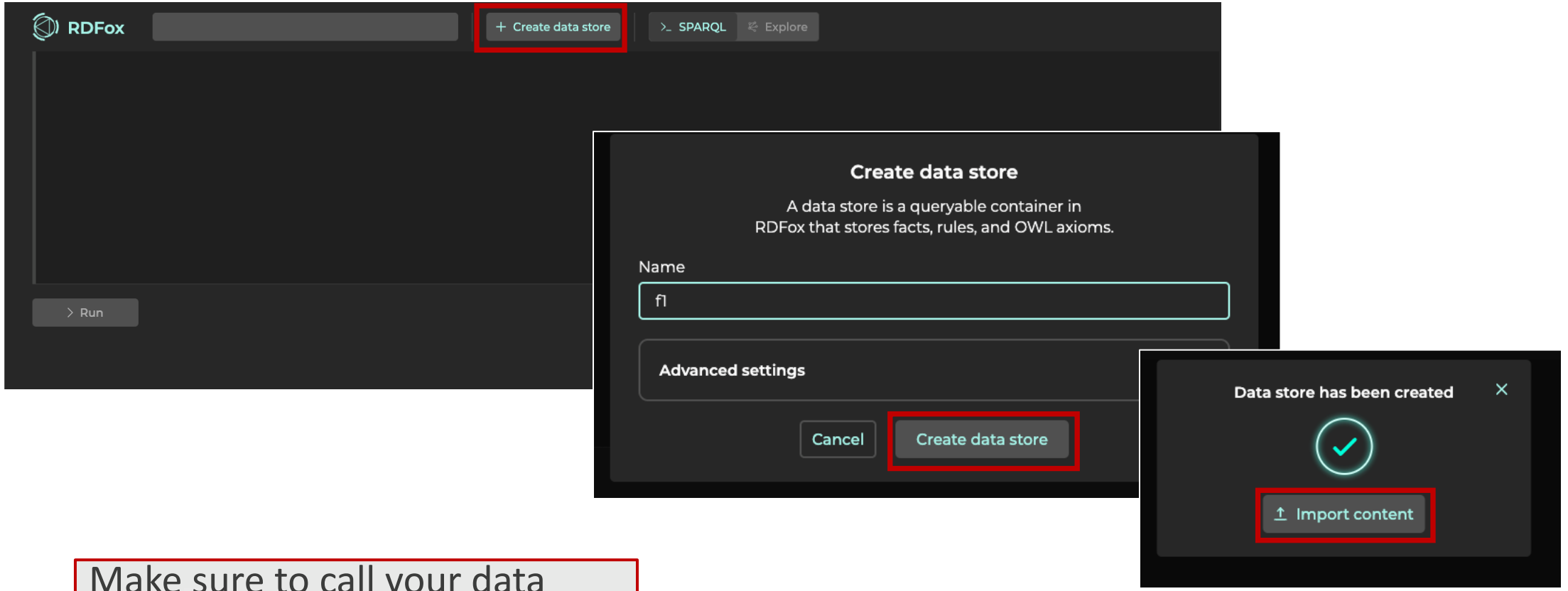
Loading Data into RDFox

Setting up RDFox
- ✓ License and executable
- ✓ IDE
- ✓ REST endpoint and UI

Loading data into RDFox
- ❑ Create datastore
- ❑ Import data

Exploring and querying with SPARQL
- ❑ Basics of SPARQL
- ❑ Useful queries to explore
- ❑ Aggregates
- ❑ Negation
- ❑ Filters
- ❑ Binds
- ❑ Optionals

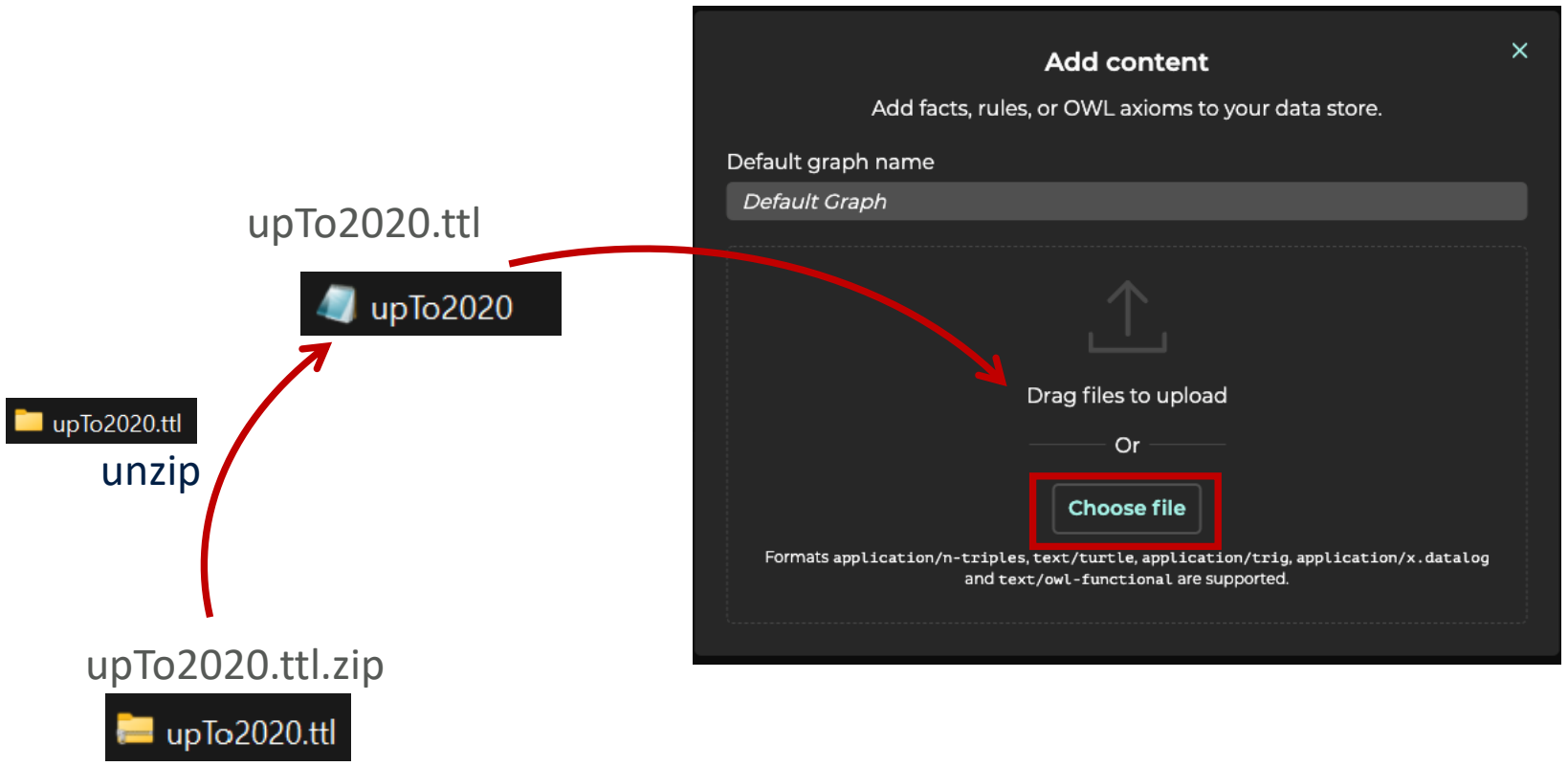
Loading data into RDFox – Create data store



Make sure to call your data store **f1** so that the links in this guide work.

https://docs.oxfordsemantic.tech/getting-started.html#getting-started-with-the-web-console

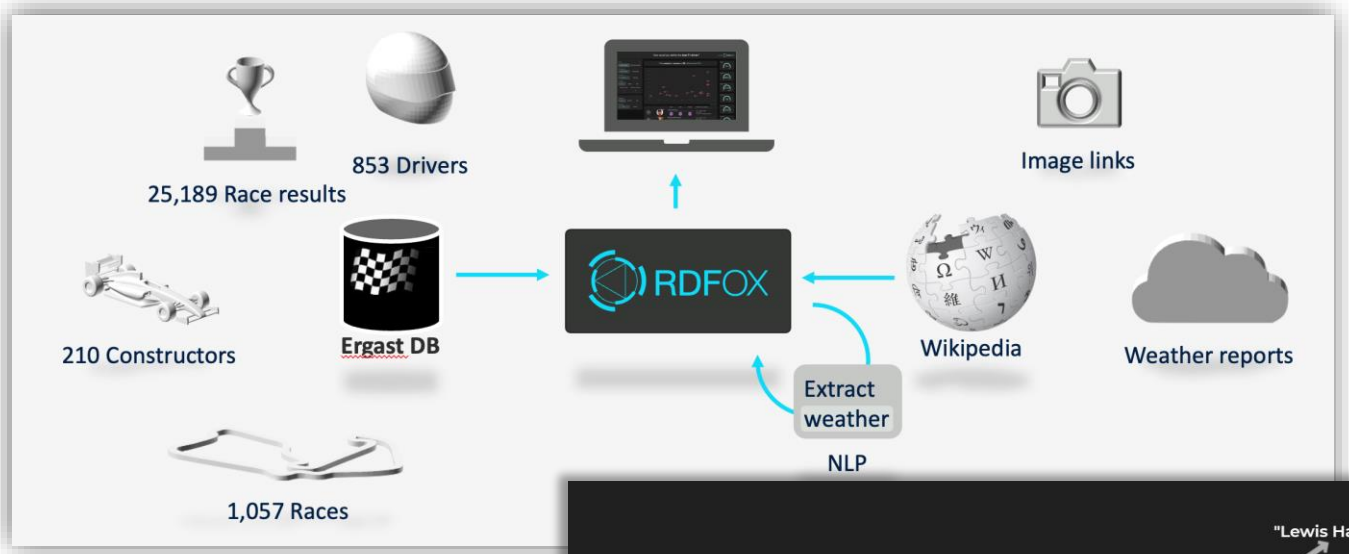
Loading data into RDFox – Load files

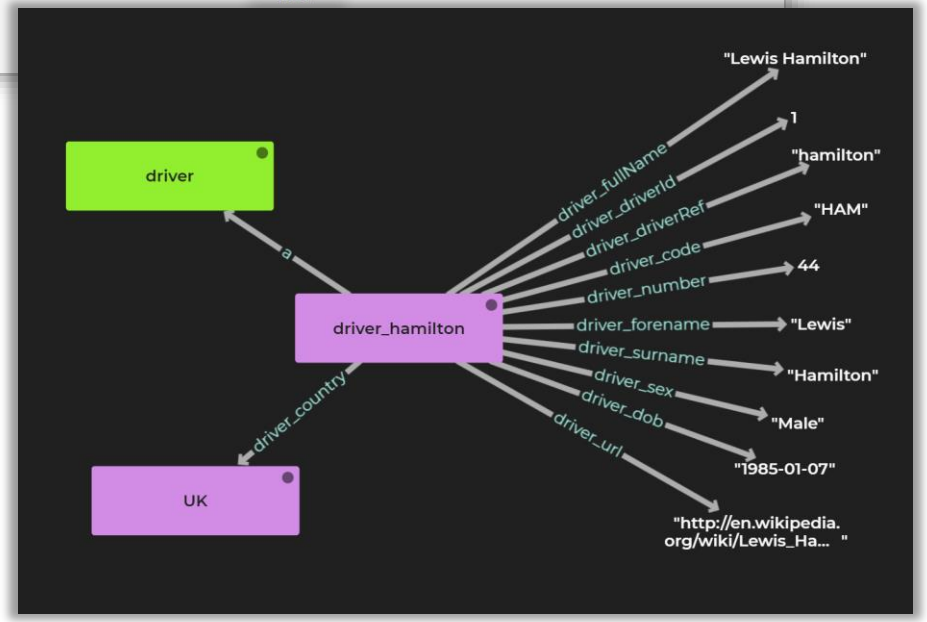


RDF

The data for today's class

- Race data from the fan-run Ergast database
  - Drivers
  - Races
  - Race results
  - Constructors
- Combined with additional information from Wikidata
  - Driver images
  - Race weather reports

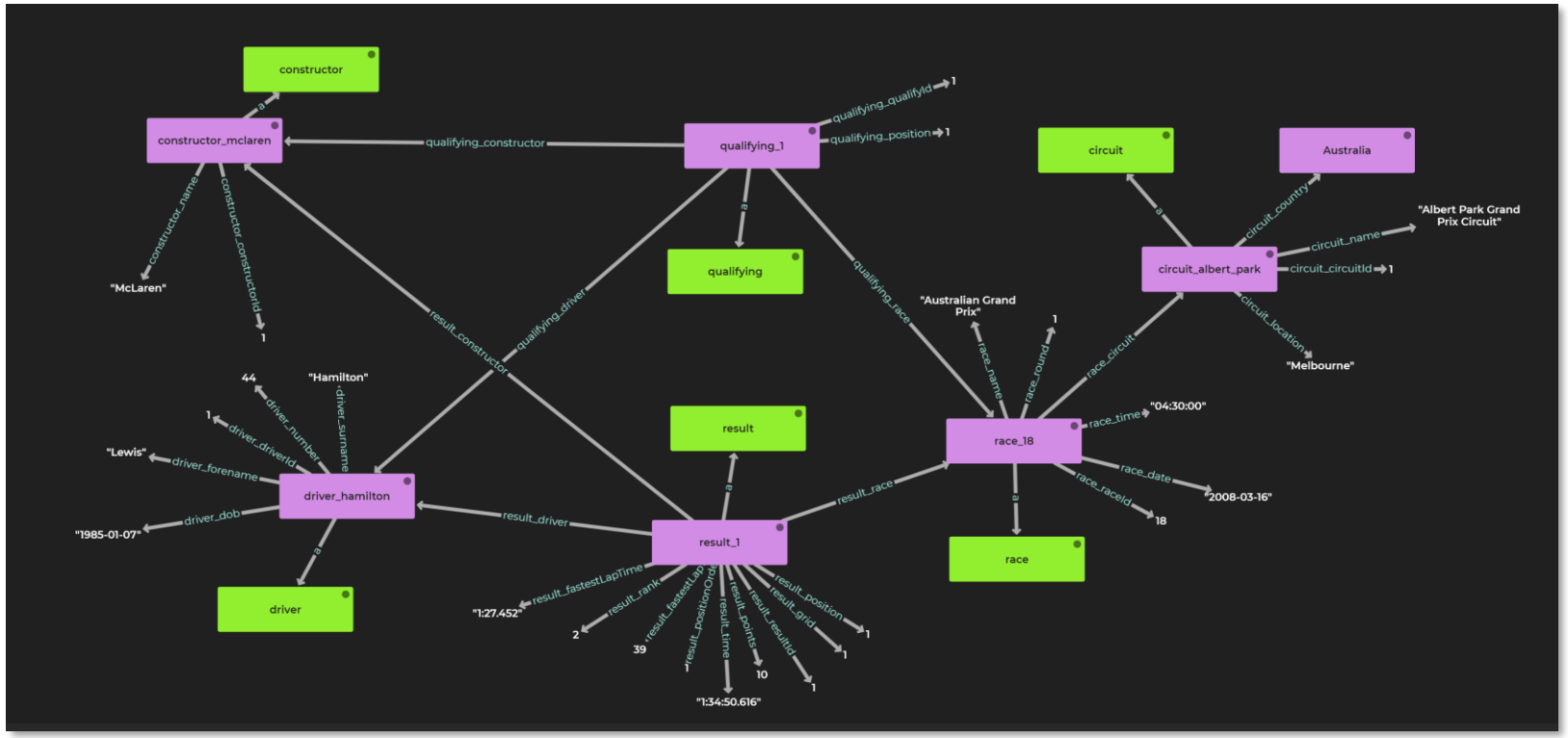




Exploring the data

Console Visualisation

Click this link to open the console.



The graph shows a small sample of the data.

Classes are green, instances of those classes are purple, and properties of those instances have no boxes.

✓ License and executable
✓ IDE
✓ REST endpoint and UI

Loading data into RDFox

✓ Create datastore
✓ Import data

Exploring and querying with SPARQL

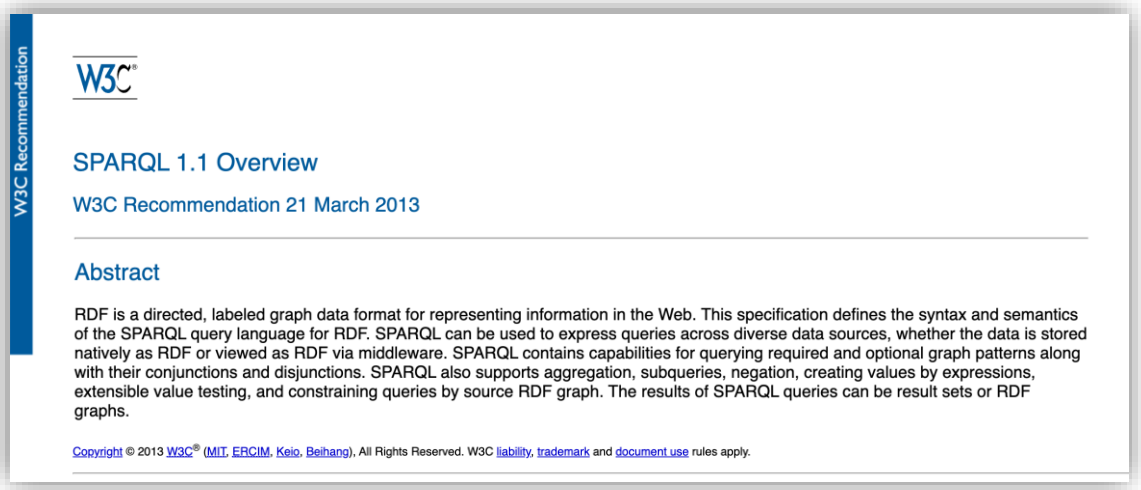
- ❑ Basics of SPARQL
- ❑ Useful queries to explore
- ❑ Aggregates
- ❑ Negation
- ❑ Filters
- ❑ Binds
- ❑ Optionals

Exploring and Querying with SPARQL

SPARQL “used to express queries”

SPARQL 1.1

https://www.w3.org/TR/sparql11-overview/



W3C Recommendation

W3C

SPARQL 1.1 Overview

W3C Recommendation 21 March 2013

Abstract

RDF is a directed, labeled graph data format for representing information in the Web. This specification defines the syntax and semantics of the SPARQL query language for RDF. SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. SPARQL also supports aggregation, subqueries, negation, creating values by expressions, extensible value testing, and constraining queries by source RDF graph. The results of SPARQL queries can be result sets or RDF graphs.

Copyright © 2013 W3C® (MIT, ERCIM, Keio, Beihang), All Rights Reserved. W3C liability, trademark and document use rules apply.

Query 1 – Basic SELECT

Console Query

SELECT queries are used to extract results from the dataset and can be modified to match whatever pattern you desire.

This is the simplest SELECT query. It returns all the triples in your dataset.

The SELECT keyword determines the type of query.

SELECT simply returns values.

```
1 #Query 1
2 PREFIX : <http://www.oxfordsemantic.tech/f1demo/>
3
4 # This query matches all the triples,
5   and returns all the variables from the matched results
6 SELECT ?S ?P ?O
7 WHERE {
8   ?S ?P ?O
9 }
```

The output variables ?S ?P and ?O tell the SELECT query what to return from the variables that are found by the WHERE clause.

Here we're asking for everything.

The WHERE clause describes what the query is looking for.

In this case we want all triples, so we write the variables ?S ?P and ?O, showing we want cases where a subject, predicate, and object exist, with any value in any position.

You can call your variables whatever you like but upper case ?S ?P and ?O have some special properties that we will make use of later.

Query 2 – DISTINCT SELECT

Console Query

Often you'll want to be more specific.

This time, we don't want all the triples to be returned, just their predicates.

Using SELECT alone would return duplicates, but we don't want that either.

The DISTINCT modifier tells the query to disregard duplicate results.

```
1 #Query 2
2 PREFIX : <http://www.oxfordsemantic.tech/f1demo/>
3
4 # What kinds of edges are in the data
5 SELECT DISTINCT ?P
6 WHERE {
7   ?S ?P ?O
8 }
```

This time we only ask for ?P to be returned.

We still have to tell the query that the variable ?P should come from the predicate of a triple, and that we want to consider all triples.

Therefore, we still need ?S ?P ?O inside the WHERE clause.

Query 3 – Finding Classes

Console Query

We want to SELECT all the DISTINCT classes that exist in our data to find the all the types of things we're storing.

In SPARQL, the keyword 'a' is used universal as the predicate for specifying the type of an entity.

We want to find all instances of triples WHERE nodes have the 'a' property.

```
1 #Query 3
2 PREFIX : <http://www.oxfordsemantic.tech/f1demo/>
3
4 # What classes of nodes are there in the data?
5 SELECT DISTINCT ?type
6 WHERE {
7   ?S a ?type
8 }
```

We are only interested in the object of these triples – their class.

We have defined the variable ?type to help us.

Query 4 – Class Properties

Console Query

This is one of the most important queries we'll cover today.

We SELECT a class of interest, and then return all its properties—not their values, just the properties it possesses.

This whole line declares a prefix. This creates a substitute, so every time you see ':', it is equivalent to `http://www.oxfordsemantic.tech/f1demo/`.

We want to find all instances of the class `:driver`.

```
1 #Query 4
2 PREFIX : <http://www.oxfordsemantic.tech/f1demo/>
3
4 # What edges are there specifically for drivers?
5 SELECT DISTINCT ?p
6 WHERE {
7   ?S a :driver ;
8     ?p ?o .
9 }
```

Subjects can be shared across multiple lines.

To denote this, all lines except the last must end with a ';'.

The last ends with a '.'.

And to consider all triples that share their subjects, ?s.

Exercise 1

Run through **queries 1-4** in the **web console**.

Try them out yourself. See what they do and how they do it.

Query 5 – Finding Drivers

Incomplete Query

Answer

Find all drivers along with their forename and surname.

```
1 #Query 5
2 PREFIX : <http://www.oxfordsemantic.tech/f1demo/>
3
4 # What are the names of my drivers?
5 SELECT DISTINCT ?driver ?forename ?surname
6 WHERE {
7   ?driver a ___ ;
8           :driver_forename ___ ;
9           :driver_surname ___ .
10 }
```

We have introduced new variables to the output that must appear in the WHERE clause.

Query 6 – A Driver's Properties

To build on what we've got so far, let's find all the associated properties and corresponding values of a specific driver. Say, Lewis Hamilton.

```
1 #Query 6
2 PREFIX : <http://www.oxfordsemantic.tech/f1demo/>
3
4 # Can you return everything about a specific driver, say Lewis Hamilton?
5 SELECT ?driver ?p ?o
6 WHERE {
7   ?driver a :driver ;
8           ___ "Lewis" ;
9           ___ "Hamilton" ;
10          ___ ___ .
11 }
```

Literals, or the values attached to properties, can be specified in queries too.

Here we're looking for a specific string so we use the '""' characters to identify it.

Query 7 – Hamilton's Races

Incomplete Query

Answer

Find the total number of races Hamilton has taken part in.

```
1  #Query 7
2  PREFIX : <http://www.oxfordsemantic.tech/f1demo/>
3
4  # Number of races Lewis Hamilton raced
5  SELECT (COUNT(?race) AS ?raceCount)
6  WHERE {
7
8    ?driver :driver_forename "Lewis" ;
9            :driver_surname "Hamilton" .
10
11   ?result :result_driver ___ ;
12           :result_race ___ .
13
14 }
```

The COUNT function counts the number of instance of a variable, ?race, and binds the resulting value as another variable, ?raceCount.

Query 8 – All Drivers' Races

Incomplete Query

Answer

Find the total number of races each driver individually has taken part in.

```
1 #Query 8
2 PREFIX : <http://www.oxfordsemantic.tech/f1demo/>
3
4 # Number of races per driver, ordered
5 SELECT ?driver (COUNT(?race) AS ?raceCount)
6 WHERE {
7   ?result :result_driver ___ ;
8           :result_race ?race .
9 }
10 GROUP BY ___
11 ORDER BY DESC(___)
```

The GROUP BY clause states how the results should be grouped when returned. Aggregate functions, such as COUNT, will now be performed for each group.

The ORDER BY clause determines the order of the query results.

The DESC modifier tells the ORDER BY clause the variable indicated should be ranked in descending order.

Exercise 2

Complete **Queries 5-8.**

HINTS:

It might help you to have **several tabs open** with the previous queries in each one—they will help you fill in the gaps!

You can click on Incomplete Query at any time to see an editable version of the query for you to play around with.

Sometimes there will be parts missing for you to fill in. You can click on Answer for the query in full for guidance.

Exercise 2 – Bonus Questions

BONUS Qs

Display the **forename and surname** of the driver (**instead of the IRI**) next to their race count.

Answer

Using the `GROUP_CONCAT` aggregate function, find the list of teammates that each driver has ever had.

Answer

HINTS:

Two people are teammates if they competed in the same **race** for the same **constructor**. Notice every result is associated with a constructor and race.

You can put `DISTINCT` inside of an aggregate function to deduplicate.

Query 9 – Drivers Without Podiums

Incomplete Query

Answer

Find all of the drivers who have **never** finished on the podium.

The FILTER function contains properties that are used to restrict the results that are returned to just those that match.

```
#Query 9
PREFIX : <http://www.oxfordsemantic.tech/f1demo/>

# Drivers who never got a podium
SELECT ___ ___
WHERE {
# First, get all the drivers with their names.
    ?driver a :driver ;
        :driver_forename ?forename ;
        :driver_surname ?surname .

# Then make sure that they have never achieved a podium (i.e. positions 3, 2, or 1)
    FILTER NOT EXISTS{
        ?result :result_driver ?driver ;
                :result_positionOrder ?positionOrder .
        FILTER(?positionOrder IN(1, 2, 3))
        }
    }
```

The NOT EXIST operator is our first glimpse of negation. It returns true if its conditions are **not** matched.

In effect, in this case the conditions of the FILTER are reversed, so results are limited to those that **do not** match what is stated.

The IN operator checks to see if the variable on the left-hand side appears in the list.

Query 10 – Most Races Without a Podium

Incomplete Query

Answer

Rank all of the drivers who have **never** finished on the podium from most to least races without a podium.

Here we have used a query within a query—an ‘inner query’, or ‘subquery’.

```
#Query 10
PREFIX : <http://www.oxfordsemantic.tech/f1demo/>

# Most races run without ever getting a podium
SELECT ?forename ?surname ?raceCount
WHERE {
# First, get all the drivers with their names.
# The order of the query atoms is mostly not important, as RDFox will rearrange it
# internally in order to have the best performance automatically.
    ?driver a :driver ;
        :driver_forename ?forename ;
        :driver_surname ?surname .

# Then make sure that they have never achieved a podium (i.e. positions 3, 2, or 1)
    FILTER NOT EXISTS{
        ?result2 :result_driver ?driver ;
                 :result_positionOrder ?positionOrder .
        FILTER(?positionOrder IN(1, 2, 3))
    }

# Finally find out how many races they have raced in.
# This is done in an 'inner query'
    {SELECT ___ (COUNT(___) AS ?raceCount)
    WHERE {
        ?result :result_driver ?driver ;
            :result_race ?race .
    }
    GROUP BY ___
    }
}
ORDER BY DESC(___) # Don't forget to order
```

Query 11 – Driver Win Percentage

Incomplete Query

Answer

Calculate the **win percentage** of each driver individually, ranking them from most to least successful.

The BIND function sets a variable to a specific value.

```
#Query 11
PREFIX : <http://www.oxfordsemantic.tech/f1demo/>

# Drivers with their win percentage, ordered by win percentage
SELECT ?forename ?surname ?raceCount ?raceWins ?percentage
WHERE {
# First get the drivers
?driver :driver_forename ?forename ;
        :driver_surname ?surname .

# Then get the race count for each driver with an innery query...
    {SELECT ?driver (COUNT(?race) AS ?raceCount)
    WHERE {
    ?result :result_driver ?driver ;
            :result_race ?race .
    }
    GROUP BY ?driver}

# ... and get the *win* count for each driver with another inner query.
    {SELECT ___ (COUNT(?race) AS ___)
    WHERE {
    ?result :result_driver ___ ;
            :result_race ___ ;
            :result_positionOrder 1 .
    }
    GROUP BY ?driver}

# Finally use the two aggregate variables to compute a percentage
# with the BIND keyword.
BIND(?raceWins / ___ AS ?percentage)

}
ORDER BY DESC(?percentage)
```

Query 12 – Correcting Win Percentage

Incomplete Query

Answer

Correct the calculation of win percentages to include even those drivers who never won.

The OPTIONAL keyword states that if, for a given case, the subsequent result cannot be evaluated, skip over it and leave it undefined.

```
#Query 12
PREFIX : <http://www.oxfordsemantic.tech/f1demo/>

SELECT ?driverName ?raceCount ?raceWinsFinal ?percentage
WHERE {

# First get the race count for each driver.
    {SELECT ?driver (COUNT(?race) AS ?raceCount)
    WHERE {
    ?result :result_driver ?driver ;
            :result_race ?race .
    }
    GROUP BY ?driver}

# Then get the *win* count for each driver.
OPTIONAL {SELECT ?driver (COUNT(?race) AS ?raceWins)
    WHERE {
    ?result :result_driver ?driver ;
            :result_race ?race ;
            :result_positionOrder 1 .
    }
    GROUP BY ?driver}

# Use the forename and surname to make the full name
# This is not strictly necessary...
?driver :driver_forename ?forename ;
        :driver_surname ?surname .
BIND(CONCAT(?forename, " ", ?surname) AS ?driverName)

# Using the COALESCE keyword, we make sure that when the number of wins is undefined
# it is 'bound' as 0.
BIND(COALESCE(___,0) AS ?raceWinsFinal)
# And use the two aggregatee variables to compute a percentage
BIND(___ / ?raceCount AS ?percentage)

}
ORDER BY DESC(___)
```

The COALESCE function returns the first value in its list that does not throw up an error (including undefined).

It is bound to the stated variable.

Exercise 3

Complete **Queries 9-12.**

HINT:

To finish on the podium you must come in positions 1, 2, or 3.

Every result will be associated with a position number which tells us where they finished in the race.

Exercise 3 – Bonus Question

Use FILTER to restrict this list to only drivers who **won at least 5** races.

Answer

Running SPARQL from the command line

As well as running SPARQL queries from the web console, queries can be run from the command line.

Here are some of the commands that you will need to set this up:

`active <data store name>`

- The active command sets the active data store for subsequent commands.

`set output out`

- The `set` command allows you to change internal variables used by the RDFox shell.
- `set output out` sets the output variable so that query results are sent to the terminal.

`evaluate <query file name>`

- Runs the query with that filename.

https://docs.oxfordsemantic.tech/command-line-reference.html

Running SPARQL from the command line

From the RDFox command line:

a. Make the `f1` data store active.
   `active f1`

b. Set the output to the terminal.
   `set output out`

c. Then specify a prefix.
   `prefix: <http://www.oxfordsemantic.tech/f1demo/>`

d. Run **query 6** again.
   If you are running RDFox from `RDFoxWorkshop-SPARQL` directory then the command should be
   `evaluate queries/q6.rq`

Congratulations!

A look back at what you've achieved.

Setting up RDFox

- ✓ License and executable
- ✓ IDE
- ✓ REST endpoint and UI

Loading data into RDFox

- ✓ Create datastore
- ✓ Import data

Exploring and querying with SPARQL

- ✓ Basics of SPARQL
- ✓ Useful queries to explore
- ✓ Aggregates
- ✓ Negation
- ✓ Filters
- ✓ Binds
- ✓ Optionals

Closing RDFox

If you are continuing with the RDFox Advanced Reasoning Workshop, please keep RDFox running.

Otherwise, stop your RDFox server with the `quit` command.

Further resources

Our website
https://www.oxfordsemantic.tech

Request an evaluation license
https://www.oxfordsemantic.tech/tryrdfoxforfree

Read the documentation
https://docs.oxfordsemantic.tech/

Our blog
https://www.oxfordsemantic.tech/the-blog