



# 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)

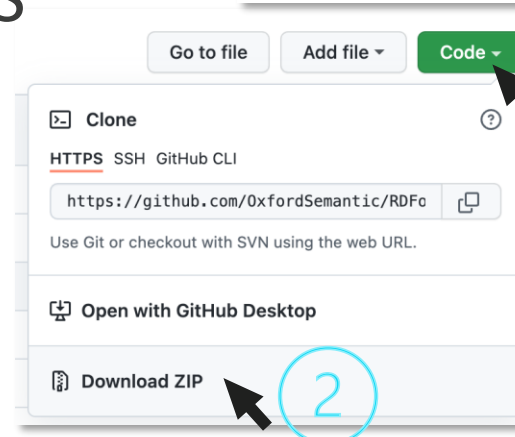
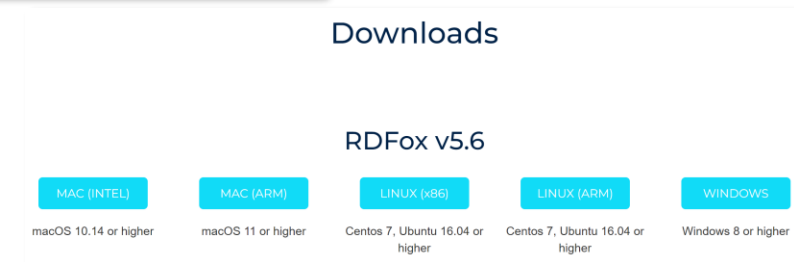
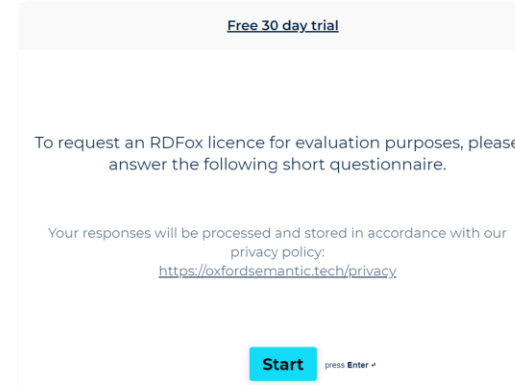
<https://www.oxfordsemantic.tech/downloads>

## 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**.



examples	✓	30/03/2022 11:23	File folder
include	✓	30/03/2022 11:23	File folder
lib	✓	30/03/2022 11:23	File folder
RDFox.lic	✓	30/03/2022 11:23	Text Document
RDFox	✓	30/03/2022 11:23	Application
readme	✓	30/03/2022 11:23	Text Document
version	✓	30/03/2022 11:23	Text Document

2

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



axioms	✓	24/05/2022 18:11	File folder
data	✓	24/05/2022 17:47	File folder
queries	✓	24/05/2022 17:47	File folder
RDFox-win64-x86_64-5.6	✓	26/05/2022 15:22	File folder
rules	✓	24/05/2022 15:05	File folder
.gitignore	✓	26/05/2022 15:23	Git Ignore Source ...
explorer	✓	24/05/2022 15:05	Text Document
RDFoxWorkshop-Reasoning	↻	10/05/2022 18:19	Chrome HTML Do...
RDFoxWorkshop-SPARQL	↻	10/05/2022 18:15	Chrome HTML Do...
README	✓	24/05/2022 15:05	Markdown Source...
start	✓	24/05/2022 15:05	Text Document
todo	✓	24/05/2022 15:05	Text Document

Without this setup you will need to 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.

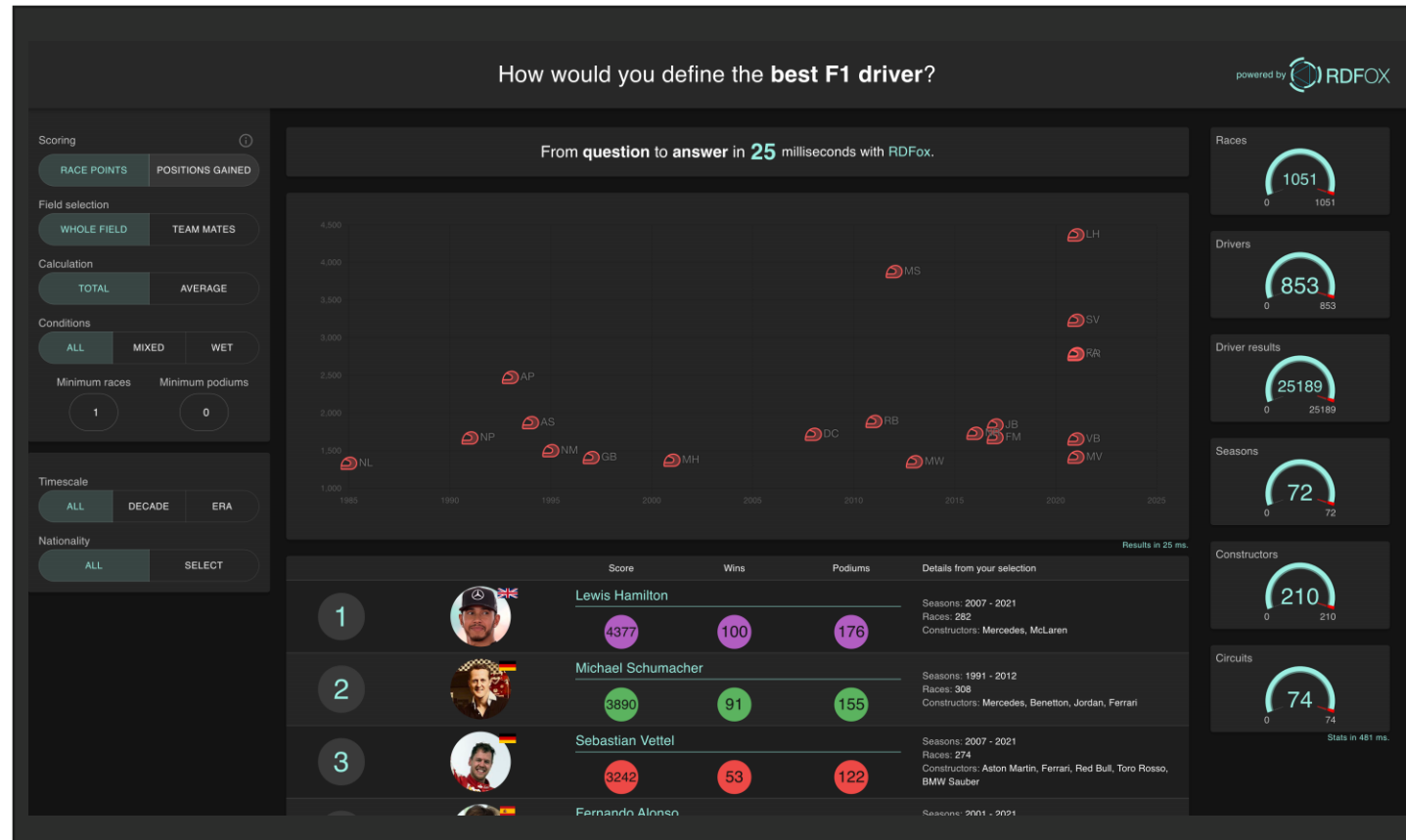


<https://docs.oxfordsemantic.tech/>

# Who is the Greatest Formula One Driver of All Time?



Controls  
and filters  
for the  
scoring  
system.



Try it for yourself!

<http://f1.rdfox.tech>



# What can Reasoning do for you?

```
1 #Query 11
2
3
4 # Drivers with their win percentage, ordered by win percentage
5 SELECT ?forename ?surname ?raceCount ?raceWins ?percentage
6 WHERE {
7   # First get the drivers
8   ?driver :driver_forename ?forename ;
9           :driver_surname ?surname .
10
11 # Then get the race count for each driver with an innery query...
12 {SELECT ?driver (COUNT(?race) AS ?raceCount)
13   WHERE {
14     ?result :result_driver ?driver ;
15            :result_race ?race .
16   }
17   GROUP BY ?driver}
18
19 # ... and get the *win* count for each driver with another inner query.
20 {SELECT ?driver (COUNT(?race) AS ?raceWins)
21   WHERE {
22     ?result :result_driver ?driver ;
23            :result_race ?race ;
24            :result_positionOrder 1 .
25   }
26   GROUP BY ?driver}
27
28 # Finally use the two aggregate variables to compute a percentage
29 # with the BIND keyword.
30 BIND(?raceWins/?raceCount AS ?percentage)
31
32 }
33 ORDER BY DESC(?percentage)
```

Fetches 108 answers in 0.011 s. ✓

Datalog Rules

```
1 #Query 19
2
3
4 SELECT ?forename ?surname ?percentage
5 WHERE {
6   ?driver :driver_forename ?forename;
7           :driver_surname ?surname;
8           :hasWinPercentage ?percentage;
9           :hasRaceCount ?count .
10 }
11 ORDER BY DESC(?percentage)
```

Fetches 108 answers in 0.003 s. ✓

# 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-arm64-6.2/RDFox sandbox`
  - › MacOS INTEL: `./RDFox-macOS-x86_64-6.2/RDFox sandbox`
  - › Windows: `./RDFox-win64-x86_64-6.2/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 Technologies 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:

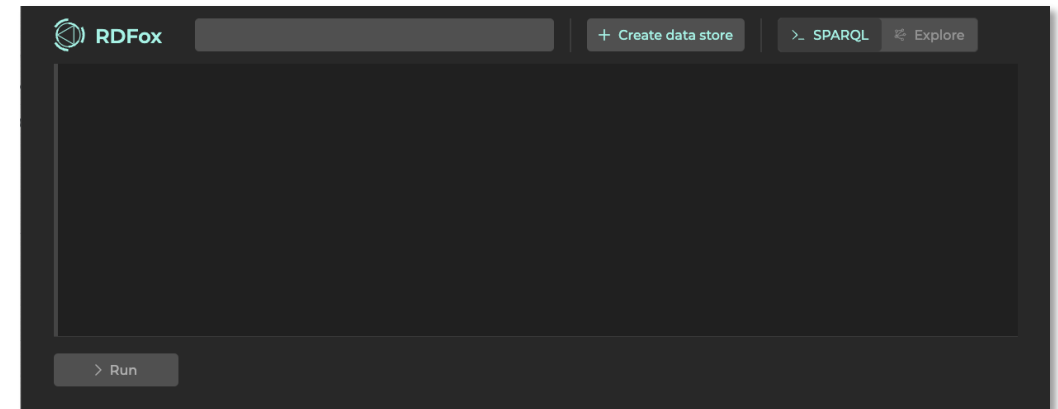
`endpoint start`

```
> 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](http://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

The image shows a sequence of three screenshots from the RDFOx web interface, illustrating the steps to create a data store:

- The first screenshot shows the RDFOx header with a button labeled "+ Create data store" highlighted by a red box.
- The second screenshot shows the "Create data store" dialog box. The "Name" field contains "f1". The "Create data store" button at the bottom right is highlighted by a red box.
- The third screenshot shows a confirmation message "Data store has been created" with a green checkmark icon. Below the message, the "Import content" button is highlighted by a red box.

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



The screenshot shows the 'Add content' dialog in RDFox. The dialog has a title bar with a close button (X). Below the title bar, it says 'Add facts, rules, or OWL axioms to your data store.' There is a text input field for 'Default graph name' with the value 'Default Graph'. Below this is a large dashed box containing an upload icon and the text 'Drag files to upload'. Below the dashed box is the word 'Or' and a button labeled 'Choose file'. Below the 'Choose file' button is a list of supported formats: 'Formats application/n-triples, text/turtle, application/trig, application/x.data-log and text/owl-functional are supported.' At the bottom of the dialog is a text input field containing 'upTo2020.ttl' and a close button (X). Below this is a toggle switch for 'Update prefixes' which is currently set to 'on'. At the very bottom are two buttons: 'Cancel' and 'Add content'.

Annotations on the left side of the image:

- A folder icon labeled 'upTo2020.ttl' with the text 'unzip' below it.
- A file icon labeled 'upTo2020' with an arrow pointing from the 'unzip' folder to it.
- A file icon labeled 'upTo2020.ttl.zip' with an arrow pointing from it to the 'upTo2020' file.
- A red arrow points from the 'upTo2020' file to the 'Choose file' button in the dialog.

Annotation on the right side of the image:

- A red box contains the text 'Remember to import prefixes!' followed by an information icon (i). A red arrow points from this box to the 'Update prefixes' toggle switch in the dialog.



# The data for today's class

Formula 1 data from the fan-run  
Ergast database:

- Drivers
- Races
- Race results
- Constructors

## Ergast Developer API

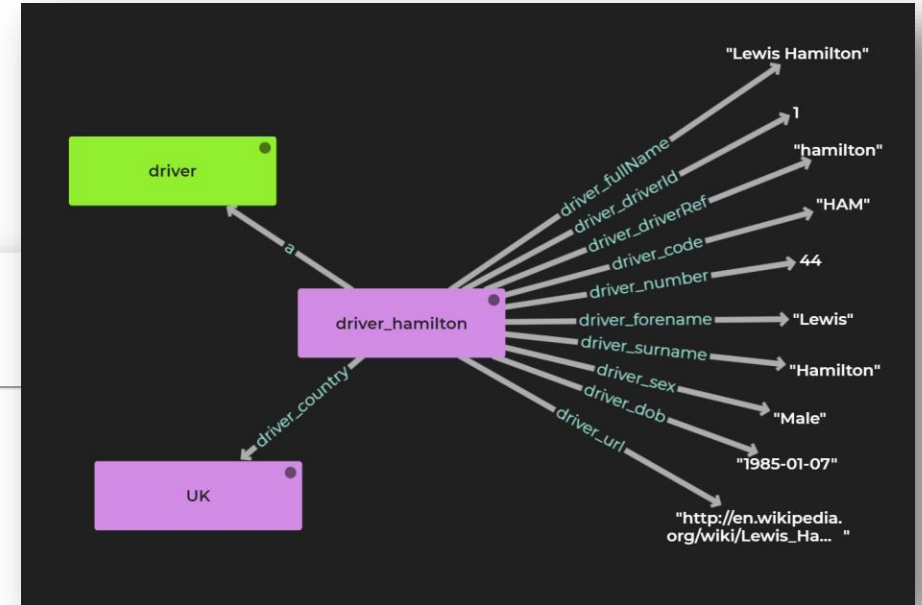
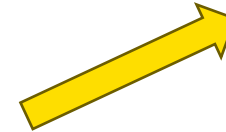
### Query Results

#### Query Details

Series	Page	Results
f1	1 of 86	857

#### Driver Table

Driver Name	Permanent Number	Nationality	DOB	Information
Carlo Abate		Italian	1932-07-10	<a href="#">Biography</a>
George Abecassis		British	1913-03-21	<a href="#">Biography</a>
Kenny Acheson		British	1957-11-27	<a href="#">Biography</a>

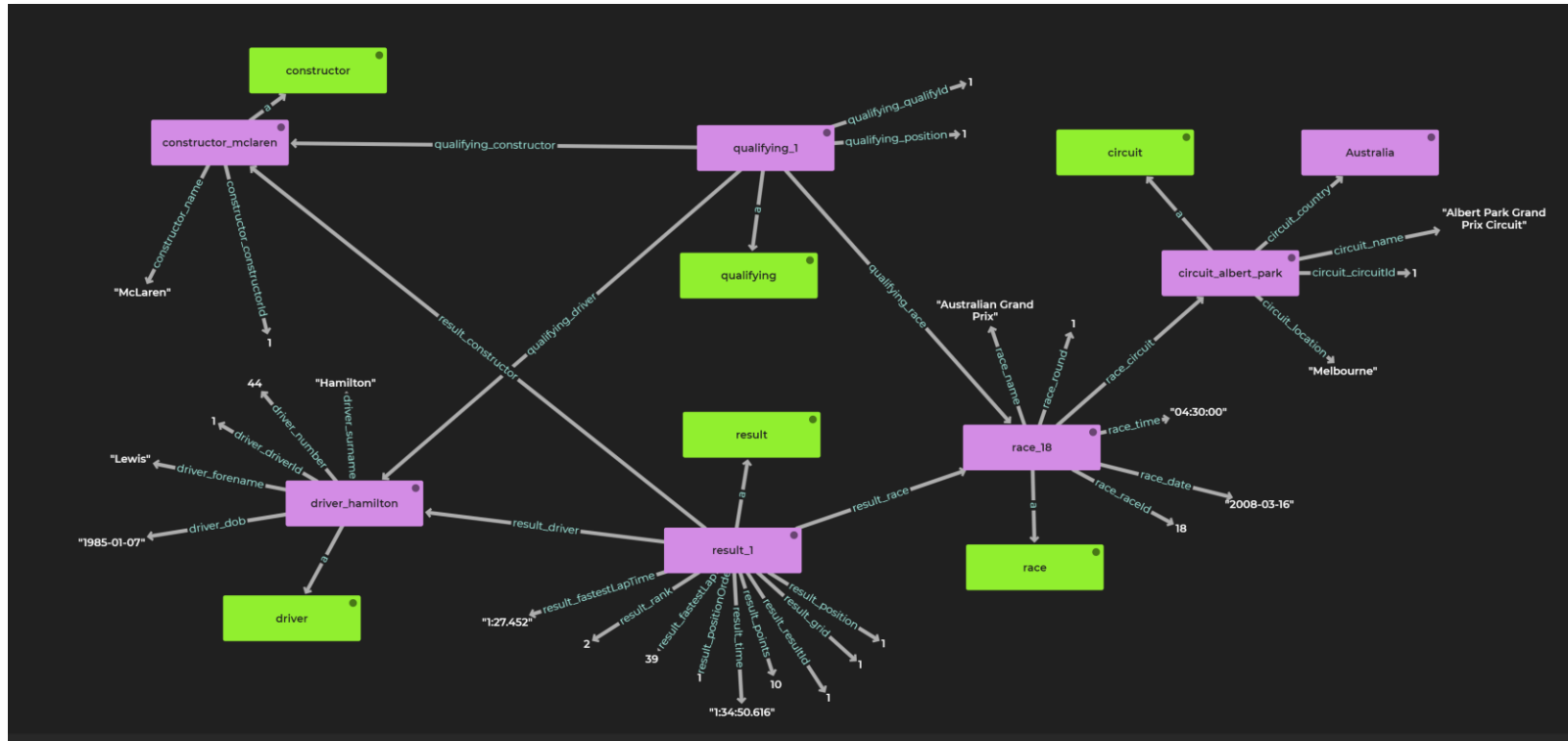


# Exploring the data

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.

# Exploring and Querying with SPARQL

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

## Loading data into RDFox

- ✓ Create datastore
- ✓ Import data

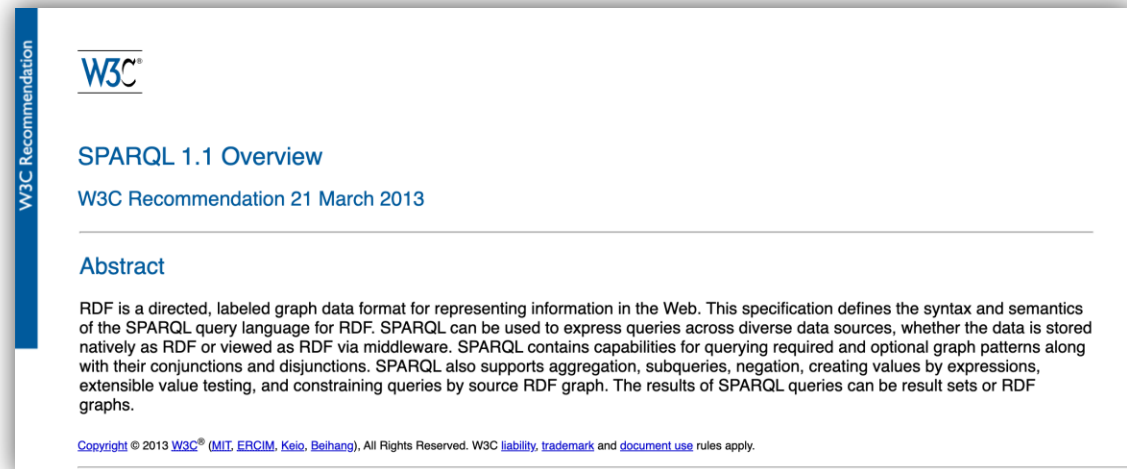
## Exploring and querying with SPARQL

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

# SPARQL “used to express queries”



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







# Query 1 – Basic SELECT

**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
3 # This query matches all the triples
4 # and returns all the variables from the matched results
5 SELECT ?S ?P ?O
6 WHERE {
7   ?S ?P ?O
8 }
```

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

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
3 # What kinds of edges are in the data?
4 SELECT DISTINCT ?P
5 WHERE {
6   ?S ?P ?O
7 }
```

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

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 universally as the predicate for specifying the type of an entity.

```
1 # Query 3
2
3 # What classes of nodes are there in the data?
4 SELECT DISTINCT ?type
5 WHERE {
6   ?S a ?type
7 }
```

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

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

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.

The data store holds a list of prefixes. For example, ':' is equivalent to <https://rdfox.com/examples/f1/>

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

✓ 9 prefixes in data store

```
1 # Query 4
2
3 # What edges are there specifically for drivers?
4 SELECT DISTINCT ?P
5 WHERE {
6   ?S a :driver ;
7     ?P ?O .
8 }
```

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

Subjects can be shared across multiple lines.

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

# 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

Find all drivers along with their forename and surname.

```
1 # Query 5
2
3 # What are the names of my drivers?
4 SELECT DISTINCT ?driver ?forename ?surname
5 WHERE {
6   ?driver a ___ ;
7   :driver_forename ___ ;
8   :driver_surname ___ .
9 }
```

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
3 # Can you return everything about a specific driver, say, Lewis Hamilton?
4 SELECT ?driver ?p ?o
5 WHERE {
6   ?driver a :driver ;
7   --- "Lewis" ; ←
8   --- "Hamilton" ;
9   --- --- .
10 }
```

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

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

```
1 # Query 7
2
3 # Number of races Lewis Hamilton has raced in
4 SELECT (COUNT(?race) AS ?raceCount)
5 WHERE {
6
7   ?driver :driver_forename "Lewis" ;
8           :driver_surname "Hamilton" .
9
10  ?result :result_driver ___ ;
11         :result_race ___ .
12 }
```

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





# Query 8 – All Drivers' Races

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

```
1 # Query 8
2
3 # Number of races per driver, ordered
4 SELECT ?driver (COUNT(?race) AS ?raceCount)
5 WHERE {
6   ?result :result_driver ___ ;
7   :result_race ?race .
8 }
9 GROUP BY ___
10 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 all 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

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

```
1 # Query 9
2
3 # Drivers who never got a podium
4 SELECT --- ---
5 WHERE {
6 # First, get all the drivers with their names
7   ?driver a :driver ;
8   :driver_forename ?forename ;
9   :driver_surname ?surname .
10
11 # Then make sure that they have never achieved a podium (i.e. positions 3, 2, or 1)
12   FILTER NOT EXISTS {
13     ?result :result_driver ?driver ;
14     :result_positionOrder ?positionOrder .
15     FILTER(?positionOrder IN (1, 2, 3))
16   }
17 }
```

The **FILTER** function contains properties that are used to restrict the results that are returned.

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

The **NOT EXISTS** 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.



# Query 10 – Most Races Without a Podium

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’.

```

1 # Query 10
2
3 # Most races run without ever getting a podium
4 SELECT ?forename ?surname ?raceCount
5 WHERE {
6   # First, get all the drivers with their names.
7   # The order of the query atoms is mostly not important, as RDFS will automatically
8   # rearrange it internally in order to have the best performance.
9     ?driver a :driver ;
10      :driver_forename ?forename ;
11      :driver_surname ?surname .
12
13 # Then make sure that they have never achieved a podium (i.e. positions 3, 2, or 1)
14   FILTER NOT EXISTS {
15     ?result2 :result_driver ?driver ;
16     :result_positionOrder ?positionOrder .
17     FILTER(?positionOrder IN (1, 2, 3))
18   }
19
20 # Finally find out how many races they have raced in
21 # This is done in an 'inner query'
22 {
23   SELECT ___ (COUNT(___)) AS ?raceCount
24   WHERE {
25     ?result :result_driver ?driver ;
26     :result_race ?race .
27   }
28   GROUP BY ___
29 }
30 }
31 ORDER BY DESC(___)
```



# Query 11 – Driver Win Percentage

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.

```
1 # Query 11
2
3 # Drivers with their win percentage, ordered by win percentage
4 SELECT ?forename ?surname ?raceCount ?raceWins ?percentage
5 WHERE {
6 # First get the drivers
7 ?driver :driver_forename ?forename ;
8         :driver_surname ?surname .
9
10 # Then get the race count for each driver with an innery query...
11 {
12     SELECT ?driver (COUNT(?race) AS ?raceCount)
13     WHERE {
14         ?result :result_driver ?driver ;
15                 :result_race ?race .
16     }
17     GROUP BY ?driver
18 }
19
20 # ... and get the *win* count for each driver with another inner query.
21 {
22     SELECT ___ (COUNT(?race) AS ___)
23     WHERE {
24         ?result :result_driver ___ ;
25                 :result_race ___ ;
26                 :result_positionOrder 1 .
27     }
28     GROUP BY ?driver
29 }
30
31 # Finally use the two aggregate variables to compute a percentage
32 # with the BIND keyword.
33 BIND(?raceWins/___ AS ?percentage)
34 }
35 ORDER BY DESC(?percentage)
```



# Query 12 – Correcting Win Percentage

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.

```

1 # Query 12
2
3 SELECT ?driverName ?raceCount ?raceWinsFinal ?percentage
4 WHERE {
5   # First get the race count for each driver.
6   {
7     SELECT ?driver (COUNT(?race) AS ?raceCount)
8     WHERE {
9       ?result :result_driver ?driver ;
10        :result_race ?race .
11     }
12     GROUP BY ?driver
13   }
14
15   # Then get the *win* count for each driver.
16   OPTIONAL {
17     SELECT ?driver (COUNT(?race) AS ?raceWins)
18     WHERE {
19       ?result :result_driver ?driver ;
20        :result_race ?race ;
21        :result_positionOrder 1 .
22     }
23     GROUP BY ?driver
24   }
25
26   # Use the forename and surname to make the full name
27   # This is not strictly necessary...
28   ?driver :driver_forename ?forename ;
29   :driver_surname ?surname .
30   BIND(CONCAT(?forename, " ", ?surname) AS ?driverName)
31
32   # Using the COALESCE keyword, we make sure that when the number of wins is undefined
33   # it is 'bound' as 0
34   BIND(COALESCE(____, 0) AS ?raceWinsFinal)
35   # And use the two aggregate variables to compute a percentage
36   BIND(____/?raceCount AS ?percentage)
37 }
38 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.

`answer <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 : <https://rdfox.com/examples/f1/>
```

- d. Run **query 6** again.

If you are running RDFox from **RDFoxWorkshop** directory, then the command should be  

```
answer 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 R<sup>D</sup>Fox Advanced Reasoning Workshop, please keep R<sup>D</sup>Fox running.

Otherwise, stop your R<sup>D</sup>Fox 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>

