**[Photograph your local culture, help Wikipedia and win!](https://commons.wikimedia.org/wiki/Special:MyLanguage/Commons:Wiki_Loves_Folklore_2023" \t "_blank)**

[Hide](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial)

Wikidata:SPARQL tutorial

[Jump to navigation](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#mw-head)[Jump to search](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#searchInput)

[Translate this page](https://www.wikidata.org/w/index.php?title=Special:Translate&group=page-Wikidata%3ASPARQL+tutorial&action=page&filter=)

**Other languages:**

* [Bahasa Indonesia](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial/id)

* [British English](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial/en-gb)

* [Deutsch](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial/de)

* **English**

* [Türkçe](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial/tr)

* [català](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial/ca)

* [dansk](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial/da)

* [eesti](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial/et)

* [español](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial/es)

* [français](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial/fr)

* [italiano](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial/it)

* [polski](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial/pl)

* [português do Brasil](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial/pt-br)

* [svenska](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial/sv)

* [čeština](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial/cs)

* [русский](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial/ru)

* [українська](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial/uk)

* [հայերեն](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial/hy)

* [العربية](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial/ar)

* [中文](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial/zh)

* [日本語](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial/ja)

WDQS, the [Wikidata Query Service](https://query.wikidata.org/), is a powerful tool to provide insight into Wikidata's content. This guide will teach you how to use it. See also the [interactive tutorial by Wikimedia Israel](https://wdqs-tutorial.toolforge.org/).

Before writing your own SPARQL query, look at {{[Item documentation](https://www.wikidata.org/wiki/Template:Item_documentation)}} or any other [generic SPARQL query template](https://www.wikidata.org/wiki/Category:Generic_SPARQL_query_templates) and see if your query is already included.



**Contents**

* [1Before we start](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#Before_we_start)
* [2SPARQL basics](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#SPARQL_basics)
* [3Our first query](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#Our_first_query)
  + [3.1Autocompletion](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#Autocompletion)
* [4Advanced triple patterns](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#Advanced_triple_patterns)
* [5Instances and classes](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#Instances_and_classes)
  + [5.1Property paths](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#Property_paths)
* [6Qualifiers](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#Qualifiers)
* [7ORDER and LIMIT](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#ORDER_and_LIMIT)
  + [7.1Exercise](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#Exercise)
    - [7.1.1Arthur Conan Doyle books](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#Arthur_Conan_Doyle_books)
    - [7.1.2Chemical elements](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#Chemical_elements)
    - [7.1.3Rivers that flow into the Mississippi](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#Rivers_that_flow_into_the_Mississippi)
    - [7.1.4Rivers that flow into the Mississippi II](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#Rivers_that_flow_into_the_Mississippi_II)
* [8OPTIONAL](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#OPTIONAL)
* [9Expressions, FILTER and BIND](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#Expressions,_FILTER_and_BIND)
  + [9.1Data types](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#Data_types)
  + [9.2Operators](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#Operators)
  + [9.3FILTER](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#FILTER)
  + [9.4BIND, BOUND, IF](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#BIND,_BOUND,_IF)
  + [9.5COALESCE](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#COALESCE)
* [10Grouping](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#Grouping)
  + [10.1City populations](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#City_populations)
  + [10.2Painting materials](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#Painting_materials)
  + [10.3Guns by manufacturer](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#Guns_by_manufacturer)
  + [10.4Publishers by number of pages](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#Publishers_by_number_of_pages)
    - [10.4.1HAVING](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#HAVING)
  + [10.5Aggregate functions summary](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#Aggregate_functions_summary)
  + [10.6wikibase:Label and aggregations](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#wikibase:Label_and_aggregations)
* [11VALUES](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#VALUES)
* [12And beyond…](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#And_beyond%E2%80%A6)
* [13See also](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#See_also)

Before we start[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=1)]

While this guide may look very long and intimidating, please don’t let that scare you away! Just learning the SPARQL basics will get you a long way – even if you stop reading after [#Our first query](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#Our_first_query), you will already understand enough to construct many interesting queries. Each section of this tutorial will empower you to write even more powerful queries.

If you have never heard of Wikidata, SPARQL, or WDQS before, here’s a short explanation of those terms:

* **Wikidata** is a knowledge database. It contains millions of statements, such as “the capital of Canada is Ottawa”, or “the Mona Lisa is painted in oil paint on poplar wood”, or “gold has a melting point of 1,064.18 degrees Celsius”.
* **SPARQL** is a language to formulate questions (queries) for knowledge databases. With the right database, a SPARQL query could answer questions like “[what is the most popular tonality in music?](https://twitter.com/WikidataFacts/status/707636944495353857)” or “[which character was portrayed by the most actors?](https://twitter.com/cdrini/status/720485653654462465)” or “[what’s the distribution of blood types?](https://twitter.com/WikidataFacts/status/722831311337951233)” or “[which authors’ works entered the public domain this year?](https://twitter.com/WikidataFacts/status/729073302652735488)”.
* **WDQS**, the [Wikidata Query Service](https://query.wikidata.org/), brings the two together: You enter a SPARQL query, it runs it against Wikidata’s dataset and shows you the result.

SPARQL basics[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=2)]

A simple SPARQL query looks like this:

**SELECT** ?a ?b ?c

**WHERE**

{

x y ?a.

m n ?b.

?b f ?c.

}

The SELECT clause lists variables that you want returned (variables start with a question mark), and the WHERE clause contains restrictions on them, mostly in the form of triples. All information in Wikidata (and similar knowledge databases) is stored in the form of triples; when you run the query, the query service tries to fill in the variables with actual values so that the resulting triples appear in the knowledge database, and returns one result for each combination of variables it finds.

A triple can be read like a sentence (which is why it ends with a period), with a *subject*, a *predicate*, and an *object*:

**SELECT** ?fruit

**WHERE**

{

?fruit hasColor yellow.

?fruit tastes sour.

}

The results for this query could include, for example, “lemon”. In Wikidata, most properties are “has”-kind properties, so the query might instead read:

**SELECT** ?fruit

**WHERE**

{

?fruit color yellow.

?fruit taste sour.

}

which reads like “?fruit *has* color ‘yellow’” (**not** “?fruit *is* the color *of* ‘yellow’” – keep this in mind for property pairs like “parent”/“child”!).

However, that’s not a good example for WDQS. Taste is subjective, so Wikidata doesn’t have a property for it. Instead, let’s think about parent/child relationships, which are mostly unambiguous.

Our first query[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=3)]

Suppose we want to list all children of the baroque composer Johann Sebastian Bach. Using pseudo-elements like in the queries above, how would you write that query?

Hopefully you got something like this:

**SELECT** ?child

**WHERE**

{

*# child "has parent" Bach*

?child parent Bach.

*# (note: everything after a ‘#’ is a comment and ignored by WDQS.)*

}

or this,

**SELECT** ?child

**WHERE**

{

*# child "has father" Bach*

?child father Bach.

}

or this,

**SELECT** ?child

**WHERE**

{

*# Bach "has child" child*

Bach child ?child.

}

The first two triples say that the ?child must have the parent/father Bach; the third says that Bach must have the child ?child. Let’s go with the second one for now.

So what remains to be done in order to turn this into a proper WDQS query? On Wikidata, items and properties are not identified by human-readable names like “father” (property) or “Bach” (item). (For good reason: “Johann Sebastian Bach” is also the name of a [German painter](https://www.wikidata.org/wiki/Q638391), and “Bach” might also refer to the [surname](https://www.wikidata.org/wiki/Q1409748), the [French commune](https://www.wikidata.org/wiki/Q936005), the [Mercury crater](https://www.wikidata.org/wiki/Q1072625), etc.) Instead, Wikidata items and properties are assigned an identifier. To find the identifier for an item, we [search](https://www.wikidata.org/wiki/Special:Search) for the item and copy the Q-number of the result that sounds like it’s the item we’re looking for (based on the description, for example). To find the identifier for a property, we do the same, but search for “P:search term” instead of just “search term”, which limits the search to properties. This tells us that the famous composer Johann Sebastian Bach is [Q1339](https://www.wikidata.org/wiki/Q1339), and the property to designate an item’s father is [P:P22](https://www.wikidata.org/wiki/Property:P22).

And last but not least, we need to include prefixes. For simple WDQS triples, items should be prefixed with wd:, and properties with wdt:. (But this only applies to fixed values – variables don’t get a prefix!)

Putting this together, we arrive at our first proper WDQS query:

**SELECT** ?child

**WHERE**

{

*# ?child father Bach*

?child **wdt**:**P22** **wd**:**Q1339**.

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fchild%0AWHERE%0A%7B%0A%23%20%3Fchild%20%20father%20%20%20Bach%0A%20%20%3Fchild%20wdt%3AP22%20wd%3AQ1339.%0A%7D)

Click that “Try it” link, then “Run” the query on the WDQS page. What do you get?

|  |
| --- |
| **child** |
| [wd:Q57225](https://www.wikidata.org/entity/Q57225) |
| [wd:Q76428](https://www.wikidata.org/entity/Q76428) |
| … |

Well that’s disappointing. You just see the identifiers. You can click on them to see their Wikidata page (including a human-readable label), but isn’t there a better way to see the results?

Well, as it happens, there is! (Aren’t rhetorical questions great?) If you include the magic text

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

somewhere within the WHERE clause, you get additional variables: For every variable ?foo in your query, you now also have a variable ?fooLabel, which contains the label of the item behind ?foo. If you add this to the SELECT clause, you get the item as well as its label:

**SELECT** ?child ?childLabel

**WHERE**

{

*# ?child father Bach*

?child **wdt**:**P22** **wd**:**Q1339**.

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fchild%20%3FchildLabel%0AWHERE%0A%7B%0A%23%20%3Fchild%20%20father%20%20%20Bach%0A%20%20%3Fchild%20wdt%3AP22%20wd%3AQ1339.%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%22.%20%7D%0A%7D)

Try running that query – you should see not only the item numbers, but also the names of the various children.

|  |  |
| --- | --- |
| **child** | **childLabel** |
| [wd:Q57225](https://www.wikidata.org/entity/Q57225) | Johann Christoph Friedrich Bach |
| [wd:Q76428](https://www.wikidata.org/entity/Q76428) | Carl Philipp Emanuel Bach |
| … |  |

**Autocompletion**[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=4)]

That SERVICE snippet looks tough to remember though, right? And going through the search function all the time while you’re writing the query is also tedious. Fortunately, WDQS offers a great solution to this: *autocompletion*. In the [query.wikidata.org](https://query.wikidata.org/) query editor, you can press Ctrl+Space (or Alt+Enter or Ctrl+Alt+Enter) at any point in the query and get suggestions for code that might be appropriate; select the right suggestion with the up/down arrow keys, and press Enter to select it.

For example, instead of writing out SERVICE wikibase:label { bd:serviceParam wikibase:language "en". } every time, you can just type SERV, hit Ctrl+Space, and the first suggestion will be that complete label service incantation, ready for use! Just hit Enter to accept it. (The formatting will be a bit different, but that doesn’t matter.)

And autocompletion can also search for you. If you type one of the Wikidata prefixes, like wd: or wdt:, and then just write text afterwards, Ctrl+Space will search for that text on Wikidata and suggest results. wd: searches for items, wdt: for properties. For example, instead of looking up the items for [Johann Sebastian Bach (Q1339)](https://www.wikidata.org/wiki/Q1339) and [father (P22)](https://www.wikidata.org/wiki/Property:P22), you can just type wd:Bach and wdt:fath and then just select the right entry from the autocompletion. (This even works with spaces in the text, e.g. wd:Johann Sebastian Bach.)

Advanced triple patterns[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=5)]

So now we’ve seen all children of Johann Sebastian Bach – more specifically: all items with the father Johann Sebastian Bach. But Bach had two wives, and so those items have two different mothers: what if we **only want to see the children of Johann Sebastian Bach with his first wife**, Maria Barbara Bach ([Q57487](https://www.wikidata.org/wiki/Q57487))? Try writing that query, based on the one above.

Done that? Okay, then onto the solution! The simplest way to do this is to add a second triple with that restriction:

**SELECT** ?child ?childLabel

**WHERE**

{

?child **wdt**:**P22** **wd**:**Q1339**.

?child **wdt**:**P25** **wd**:**Q57487**.

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fchild%20%3FchildLabel%0AWHERE%0A%7B%0A%20%20%3Fchild%20wdt%3AP22%20wd%3AQ1339.%0A%20%20%3Fchild%20wdt%3AP25%20wd%3AQ57487.%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%22.%20%7D%0A%7D)

In English, this reads:

Child has father Johann Sebastian Bach.

Child has mother Maria Barbara Bach.

That sounds a bit awkward, doesn’t it? In natural language, we’d abbreviate this:

Child has father Johann Sebastian Bach and mother Maria Barbara Bach.

In fact, it’s possible to express the same abbreviation in SPARQL as well: **if you end a triple with a semicolon (;) instead of a period, you can add another predicate-object pair**. This allows us to abbreviate the above query to:

**SELECT** ?child ?childLabel

**WHERE**

{

?child **wdt**:**P22** **wd**:**Q1339**;

**wdt**:**P25** **wd**:**Q57487**.

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fchild%20%3FchildLabel%0AWHERE%0A%7B%0A%20%20%3Fchild%20wdt%3AP22%20wd%3AQ1339%3B%0A%20%20%20%20%20%20%20%20%20wdt%3AP25%20wd%3AQ57487.%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%22.%20%7D%0A%7D)

which has the same results, but less repetition in the query.

Now suppose that, out of those results, we’re interested only in those children who were also composers and pianists. The relevant properties and items are [occupation (P106)](https://www.wikidata.org/wiki/Property:P106), [composer (Q36834)](https://www.wikidata.org/wiki/Q36834) and [pianist (Q486748)](https://www.wikidata.org/wiki/Q486748). Try updating the above query to add these restrictions!

Here’s my solution:

**SELECT** ?child ?childLabel

**WHERE**

{

?child **wdt**:**P22** **wd**:**Q1339**;

**wdt**:**P25** **wd**:**Q57487**;

**wdt**:**P106** **wd**:**Q36834**;

**wdt**:**P106** **wd**:**Q486748**.

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fchild%20%3FchildLabel%0AWHERE%0A%7B%0A%20%20%3Fchild%20wdt%3AP22%20wd%3AQ1339%3B%0A%20%20%20%20%20%20%20%20%20wdt%3AP25%20wd%3AQ57487%3B%0A%20%20%20%20%20%20%20%20%20wdt%3AP106%20wd%3AQ36834%3B%0A%20%20%20%20%20%20%20%20%20wdt%3AP106%20wd%3AQ486748.%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%22.%20%7D%0A%7D)

This uses the ; abbreviation two more times to add the two required occupations. But as you might notice, there’s still some repetition. This is as if we said:

Child has occupation composer and occupation pianist.

which we would usually abbreviate as:

Child has occupation composer and pianist.

And SPARQL has some syntax for that as well: just like a ; allows you to append a predicate-object pair to a triple (reusing the subject), **a , allows you to append another object to a triple (reusing both subject and predicate)**. With this, the query can be abbreviated to:

**SELECT** ?child ?childLabel

**WHERE**

{

?child **wdt**:**P22** **wd**:**Q1339**;

**wdt**:**P25** **wd**:**Q57487**;

**wdt**:**P106** **wd**:**Q36834**,

**wd**:**Q486748**.

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fchild%20%3FchildLabel%0AWHERE%0A%7B%0A%20%20%3Fchild%20wdt%3AP22%20wd%3AQ1339%3B%0A%20%20%20%20%20%20%20%20%20wdt%3AP25%20wd%3AQ57487%3B%0A%20%20%20%20%20%20%20%20%20wdt%3AP106%20wd%3AQ36834%2C%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20wd%3AQ486748.%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%22.%20%7D%0A%7D)

Note: indentation and other whitespaces don’t actually matter – they just make it more readable. You can also write this as:

**SELECT** ?child ?childLabel

**WHERE**

{

?child **wdt**:**P22** **wd**:**Q1339**;

**wdt**:**P25** **wd**:**Q57487**;

**wdt**:**P106** **wd**:**Q36834**, **wd**:**Q486748**.

*# both occupations in one line*

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fchild%20%3FchildLabel%0AWHERE%0A%7B%0A%20%20%3Fchild%20wdt%3AP22%20wd%3AQ1339%3B%0A%20%20%20%20%20%20%20%20%20wdt%3AP25%20wd%3AQ57487%3B%0A%20%20%20%20%20%20%20%20%20wdt%3AP106%20wd%3AQ36834%2C%20wd%3AQ486748.%0A%20%20%23%20both%20occupations%20in%20one%20line%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%22.%20%7D%0A%7D)

or, rather less readable:

**SELECT** ?child ?childLabel

**WHERE**

{

?child **wdt**:**P22** **wd**:**Q1339**;

**wdt**:**P25** **wd**:**Q57487**;

**wdt**:**P106** **wd**:**Q36834**,

**wd**:**Q486748**.

*# no indentation; makes it hard to distinguish between ; and ,*

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fchild%20%3FchildLabel%0AWHERE%0A%7B%0A%20%20%3Fchild%20wdt%3AP22%20wd%3AQ1339%3B%0A%20%20wdt%3AP25%20wd%3AQ57487%3B%0A%20%20wdt%3AP106%20wd%3AQ36834%2C%0A%20%20wd%3AQ486748.%0A%20%20%23%20no%20indentation%3B%20makes%20it%20hard%20to%20distinguish%20between%20%3B%20and%20%2C%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%22.%20%7D%0A%7D)

Luckily, the WDQS editor indents lines for you automatically, so you usually don’t have to worry about this.

Alright, let’s summarize here. We’ve seen that queries are structured like text. Each triple about a subject is terminated by a period. Multiple predicates about the same subject are separated by semicolons, and multiple objects for the same subject and predicate can be listed separated by commas.

**SELECT** ?s1 ?s2 ?s3

**WHERE**

{

?s1 p1 o1;

p2 o2;

p3 o31, o32, o33.

?s2 p4 o41, o42.

?s3 p5 o5;

p6 o6.

}

Now I want to introduce one more abbreviation that SPARQL offers. So if you’ll humor me for one more hypothetical scenario…

Suppose we’re not actually interested in Bach’s children. (Who knows, perhaps that’s actually true for you!) But we are interested in his *grand*children. (Hypothetically.) There’s one complication here: a grandchild may be related to Bach via the mother or the father. That’s two different properties, which is inconvenient. Instead, let’s flip the relation around: Wikidata also has a “child” property, [P:P40](https://www.wikidata.org/wiki/Property:P40), which points from parent to child and is gender-independent. With this information, can you write a query that returns Bach’s grandchildren?

Here’s my solution:

**SELECT** ?grandChild ?grandChildLabel

**WHERE**

{

**wd**:**Q1339** **wdt**:**P40** ?child.

?child **wdt**:**P40** ?grandChild.

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

[Run it!](https://query.wikidata.org/#SELECT%20%3FgrandChild%20%3FgrandChildLabel%0AWHERE%0A%7B%0A%20%20wd%3AQ1339%20wdt%3AP40%20%3Fchild.%0A%20%20%3Fchild%20wdt%3AP40%20%3FgrandChild.%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%22.%20%7D%0A%7D)

In natural language, this reads:

Bach has a child ?child.

?child has a child ?grandChild.

Once more, I propose that we abbreviate this English sentence, and then I want to show you how SPARQL supports a similar abbreviation. Observe how we actually don’t care about the child: we don’t use the variable except to talk about the grandchild. We could therefore abbreviate the sentence to:

Bach has as child someone who has a child ?grandChild.

Instead of saying who Bach’s child is, we just say “someone”: we don’t care who it is. But we can refer back to them because we’ve said “someone *who*”: this starts a relative clause, and within that relative clause we can say things about “someone” (e.g., that they “have a child ?grandChild”). In a way, “someone” is a variable, but a special one that’s only valid within this relative clause, and one that we don’t explicitly refer to (we say “someone who is this and does that”, not “someone who is this and someone who does that” – that’s two different “someone”s).

In SPARQL, this can be written as:

**SELECT** ?grandChild ?grandChildLabel

**WHERE**

{

**wd**:**Q1339** **wdt**:**P40** [ **wdt**:**P40** ?grandChild ].

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

[Run it!](https://query.wikidata.org/#SELECT%20%3FgrandChild%20%3FgrandChildLabel%0AWHERE%0A%7B%0A%20%20wd%3AQ1339%20wdt%3AP40%20%5B%20wdt%3AP40%20%3FgrandChild%20%5D.%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%22.%20%7D%0A%7D)

You can use a pair of brackets ([]) in place of a variable, which acts as an anonymous variable. Inside the brackets, you can specify predicate-object pairs, just like after a ; after a normal triple; the implicit subject is in this case the anonymous variable that the brackets represent. (Note: also just like after a ;, you can add more predicate-object pairs with more semicolons, or more objects for the same predicate with commas.)

And that’s it for triple patterns! There’s more to SPARQL, but as we’re about to leave the parts of it that are strongly analogous to natural language, I’d like to summarize that relationship once more:

|  |  |  |  |
| --- | --- | --- | --- |
| **natural language** | **example** | **SPARQL** | **example** |
| sentence | Juliet loves Romeo. | period | juliet loves romeo**.** |
| conjunction (clause) | Romeo loves Juliet **and** kills himself. | semicolon | romeo loves juliet**;** kills romeo. |
| conjunction (noun) | Romeo kills Tybalt **and** himself. | comma | romeo kills tybalt**,** romeo. |
| relative clause | Juliet loves **someone who** kills Tybalt. | brackets | juliet loves **[** kills tybalt **]**. |

Instances and classes[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=6)]

Earlier, I said that most Wikidata properties are “has” relations: *has* child, *has* father, *has* occupation. But sometimes (in fact, frequently), you also need to talk about what something *is*. But there are in fact two kinds of relations there:

* *Gone with the Wind* **is** a film.
* A film **is** a work of art.

*Gone with the Wind* is one particular film. It has a particular director (Victor Fleming), a specific duration (238 minutes), a list of cast members (Clark Gable, Vivien Leigh, …), and so on.

*Film* is a general concept. Films can have directors, durations, and cast members, but the concept “film” as such does not have any particular director, duration, or cast members. And although a film **is** a work of art, and a work of art usually has a creator, the concept of “film” itself does not have a creator – only particular *instances* of this concept do.

This difference is why there are two properties for “is” in Wikidata: [instance of (P31)](https://www.wikidata.org/wiki/Property:P31) and [subclass of (P279)](https://www.wikidata.org/wiki/Property:P279). *Gone with the Wind* is a particular instance of the class “film”; the class “film” is a subclass (more specific class; specialization) of the more general class “work of art”.

To help you to figure about the difference, you can try to use two different verbs: "is a" and "is a kind of". If "is a kind of" works (e.g. A film "is a kind of" work of art), it indicates that you are talking about a subclass, a specialization of a broader class and you should use [subclass of (P279)](https://www.wikidata.org/wiki/Property:P279). If "is a kind of" does not work (e.g. the sentence *Gone with the wind "is a kind of" film* does not make sense), it indicates that you are talking about a particular instance and you should use [instance of (P31)](https://www.wikidata.org/wiki/Property:P31).

So what does this mean for us when we’re writing SPARQL queries? When we want to search for “all works of art”, it’s not enough to search for all items that are directly instances of “work of art”:

**SELECT** ?work ?workLabel

**WHERE**

{

?work **wdt**:**P31** **wd**:**Q838948**. *# instance of work of art*

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fwork%20%3FworkLabel%0AWHERE%0A%7B%0A%20%20%3Fwork%20wdt%3AP31%20wd%3AQ838948.%20%23%20instance%20of%20work%20of%20art%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%22.%20%7D%0A%7D)

As I’m writing this (October 2016), that query only returns 2,815 results – obviously, there are more works of art than that! The problem is that this misses items like *Gone with the Wind*, which is only an instance of “film”, not of “work of art”. “film” is a subclass of “work of art”, but we need to tell SPARQL to take that into account when searching.

One possible solution to this is the [] syntax we talked about: *Gone with the Wind* is an instance of **some** subclass of “work of art”. (For exercise, try writing that query!) But that still has problems:

1. We’re no longer including items that are directly instances of work of art.
2. We’re still missing items that are instances of some subclass of some *other* subclass of “work of art” – for example, *Snow White and the Seven Dwarfs* is an animated film, which is a film, which is a work of art. In this case, we need to follow two “subclass of” statements – but it might also be three, four, five, any number really.

The solution: ?item wdt:P31/wdt:P279\* ?class. This means that there’s one “instance of” and then any number of “subclass of” statements between the item and the class.

**SELECT** ?work ?workLabel

**WHERE**

{

?work **wdt**:**P31**/**wdt**:**P279**\* **wd**:**Q838948**. *# instance of any subclass of work of art*

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fwork%20%3FworkLabel%0AWHERE%0A%7B%0A%20%20%3Fwork%20wdt%3AP31%2Fwdt%3AP279%2A%20wd%3AQ838948.%20%23%20instance%20of%20any%20subclass%20of%20work%20of%20art%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%22.%20%7D%0A%7D)

(I don’t recommend running that query. WDQS can handle it (just barely), but your browser might crash when trying to display the results because there are so many of them.)

Now you know how to search for all works of art, or all buildings, or all human settlements: the magic incantation wdt:P31/wdt:P279\*, along with the appropriate class. This uses some more SPARQL features that I haven’t explained yet, but quite honestly, this is almost the only relevant use of those features, so you don’t *need* to understand how it works in order to use WDQS effectively. If you want to know, I’ll explain it in a bit, but you can also just skip the next section and memorize or copy+paste wdt:P31/wdt:P279\* from here when you need it.

**Property paths**[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=7)]

Property paths are a way to very tersely write down a path of properties between two items. The simplest path is just a single property, which forms an ordinary triple:

?item **wdt**:**P31** ?class.

You can add path elements with a forward slash (/).

?item **wdt**:**P31**/**wdt**:**P279**/**wdt**:**P279** ?class.

This is equivalent to either of the following:

?item **wdt**:**P31** ?temp1.

?temp1 **wdt**:**P279** ?temp2.

?temp2 **wdt**:**P279** ?class.

?item **wdt**:**P31** [ **wdt**:**P279** [ **wdt**:**P279** ?class ] ].

Exercise: rewrite the “grandchildren of Bach” query from earlier to use this syntax.

An asterisk (\*) after a path element means “*zero* or more of this element”.

?item **wdt**:**P31**/**wdt**:**P279**\* ?class.

*# means:*

?item **wdt**:**P31** ?class

*# or*

?item **wdt**:**P31**/**wdt**:**P279** ?class

*# or*

?item **wdt**:**P31**/**wdt**:**P279**/**wdt**:**P279** ?class

*# or*

?item **wdt**:**P31**/**wdt**:**P279**/**wdt**:**P279**/**wdt**:**P279** ?class

*# or ...*

If there are no other elements in the path, ?a something\* ?b means that ?b might also just be ?a directly, with no path elements between them at all.

A plus (+) is similar to an asterisk, but means “*one* or more of this element”. The following query finds all descendants of Bach:

**SELECT** ?descendant ?descendantLabel

**WHERE**

{

**wd**:**Q1339** **wdt**:**P40**+ ?descendant.

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fdescendant%20%3FdescendantLabel%0AWHERE%0A%7B%0A%20%20wd%3AQ1339%20wdt%3AP40%2B%20%3Fdescendant.%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%22.%20%7D%0A%7D)

If we used an asterisk instead of a plus here, the query results would include Bach himself.

A question mark (?) is similar to an asterisk or a plus, but means “*zero or one* of this element”.

You can separate path elements with a vertical bar (|) instead of a forward slash; this means “either-or”: the path might use either of those properties. (But not combined – an either-or path segment always matches a path of length one.)

You can also group path elements with parentheses (()), and freely combine all these syntax elements (/|\*+?). This means that another way to find all descendants of Bach is:

**SELECT** ?descendant ?descendantLabel

**WHERE**

{

?descendant (**wdt**:**P22**|**wdt**:**P25**)+ **wd**:**Q1339**.

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fdescendant%20%3FdescendantLabel%0AWHERE%0A%7B%0A%20%20%3Fdescendant%20%28wdt%3AP22%7Cwdt%3AP25%29%2B%20wd%3AQ1339.%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%22.%20%7D%0A%7D)

Instead of using the “child” property to go from Bach to his descendants, we use the “father” and “mother” properties to go from the descendants to Bach. The path might include two mothers and one father, or four fathers, or father-mother-mother-father, or any other combination. (Though, of course, Bach can’t be the mother of someone, so the last element will always be father.)

Qualifiers[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=8)]

(Good news first: this section introduces no additional SPARQL syntax – yay! Take a quick breath and relax, this should be a piece of cake. Right?)

So far, we’ve only talked about simple statements: subject, property, object. But Wikidata statements are more than that: they can also have qualifiers and references. For example, the Mona Lisa ([Q12418](https://www.wikidata.org/wiki/Q12418)) has three [made from material (P186)](https://www.wikidata.org/wiki/Property:P186) statements:

1. [oil paint (Q296955)](https://www.wikidata.org/wiki/Q296955), the main material;
2. [poplar wood (Q291034)](https://www.wikidata.org/wiki/Q291034), with the qualifier [applies to part (P518)](https://www.wikidata.org/wiki/Property:P518)[painting support (Q861259)](https://www.wikidata.org/wiki/Q861259) – this is the material that the Mona Lisa was painted *on*; and
3. [wood (Q287)](https://www.wikidata.org/wiki/Q287), with the qualifiers [applies to part (P518)](https://www.wikidata.org/wiki/Property:P518)[stretcher (Q1737943)](https://www.wikidata.org/wiki/Q1737943) and [start time (P580)](https://www.wikidata.org/wiki/Property:P580) 1951 – this is a part that was added to the painting later.

Suppose we want to find all paintings with their painting surface, that is, those [made from material (P186)](https://www.wikidata.org/wiki/Property:P186) statements with a qualifier [applies to part (P518)](https://www.wikidata.org/wiki/Property:P518)[painting support (Q861259)](https://www.wikidata.org/wiki/Q861259). How do we do that? That’s more information than can be represented in a single triple.

The answer is: more triples! (Rule of thumb: Wikidata’s solution for almost everything is “more items”, and the corresponding WDQS rule is “more triples”. References, numeric precision, values with units, geocoordinates, etc., all of which we’re skipping here, also work like this.) So far, we’ve used the wdt: prefix for our statement triples, which points directly to the object of the statement. But there’s also another prefix: p:, which points not to the object, but to a *statement node*. This node then is the subject of other triples: the prefix ps: (for **p**roperty **s**tatement) points to the statement object, the prefix pq: (**p**roperty **q**ualifier) to qualifiers, and prov:wasDerivedFrom points to reference nodes (which we’ll ignore for now).

That was a lot of abstract text. Here’s a concrete example for the Mona Lisa:

**wd**:**Q12418** **p**:**P186** ?statement1. *# Mona Lisa: material used: ?statement1*

?statement1 **ps**:**P186** **wd**:**Q296955**. *# value: oil paint*

**wd**:**Q12418** **p**:**P186** ?statement2. *# Mona Lisa: material used: ?statement2*

?statement2 **ps**:**P186** **wd**:**Q291034**. *# value: poplar wood*

?statement2 **pq**:**P518** **wd**:**Q861259**. *# qualifier: applies to part: painting surface*

**wd**:**Q12418** **p**:**P186** ?statement3. *# Mona Lisa: material used: ?statement3*

?statement3 **ps**:**P186** **wd**:**Q287**. *# value: wood*

?statement3 **pq**:**P518** **wd**:**Q1737943**. *# qualifier: applies to part: stretcher bar*

?statement3 **pq**:**P580** 1951. *# qualifier: start time: 1951 (pseudo-syntax)*

We can abbreviate this a lot with the [] syntax, replacing the ?statement variables:

**wd**:**Q12418** **p**:**P186** [ **ps**:**P186** **wd**:**Q296955** ].

**wd**:**Q12418** **p**:**P186** [

**ps**:**P186** **wd**:**Q291034**;

**pq**:**P518** **wd**:**Q861259**

].

**wd**:**Q12418** **p**:**P186** [

**ps**:**P186** **wd**:**Q287**;

**pq**:**P518** **wd**:**Q1737943**;

**pq**:**P580** 1951

].

Can you use this knowledge to write a query for all paintings with their painting surface?

Here’s my solution:

**SELECT** ?painting ?paintingLabel ?material ?materialLabel

**WHERE**

{

?painting **wdt**:**P31**/**wdt**:**P279**\* **wd**:**Q3305213**;

**p**:**P186** [ **ps**:**P186** ?material; **pq**:**P518** **wd**:**Q861259** ].

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fpainting%20%3FpaintingLabel%20%3Fmaterial%20%3FmaterialLabel%0AWHERE%0A%7B%0A%20%20%3Fpainting%20wdt%3AP31%2Fwdt%3AP279%2A%20wd%3AQ3305213%3B%0A%20%20%20%20%20%20%20%20%20%20%20%20p%3AP186%20%5B%20ps%3AP186%20%3Fmaterial%3B%20pq%3AP518%20wd%3AQ861259%20%5D.%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%22.%20%7D%0A%7D)

First, we limit ?painting to all instances of [painting (Q3305213)](https://www.wikidata.org/wiki/Q3305213) or a subclass thereof. Then, we extract the material from the p:P186 statement node, limiting the statements to those that have an [applies to part (P518)](https://www.wikidata.org/wiki/Property:P518)[painting support (Q861259)](https://www.wikidata.org/wiki/Q861259) qualifier.

ORDER and LIMIT[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=9)]

We return to our regular scheduled program of more SPARQL features.

So far, we’ve only had queries where we were interested in all results. But it’s quite common to care only about a few results: those that are most extreme in some way – oldest, youngest, earliest, latest, highest population, lowest melting point, most children, most materials used, and so on. The common factor here is that the results are *ranked* in some way, and then we care about the first few results (those with the best rank).

This is controlled by two clauses, which are appended to the WHERE {} block (after the braces, not inside!): ORDER BY and LIMIT.

ORDER BY something sorts the results by something. something can be any expression – for now, the only kind of expression we know are simple variables (?something), but we’ll see some other kinds later. This expression can also be wrapped in either ASC() or DESC() to specify the sorting order (**asc**ending or **desc**ending). (If you don’t specify either, the default is ascending sort, so ASC(something) is equivalent to just something.)

LIMIT count cuts off the result list at count results, where count is any natural number. For example, LIMIT 10 limits the query to ten results. LIMIT 1 only returns a single result.

(You can also use LIMIT without ORDER BY. In this case, the results aren’t sorted, so you don’t have any guarantee which results you’ll get. Which is fine if you happen to know that there’s only a certain number of results, or you’re just interested in *some* result, but don’t care about which one. In either case, adding the LIMIT can significantly speed up the query, since WDQS can stop searching for results as soon as it’s found enough to fill the limit.)

Exercise time! Try to write a query that returns the ten most populous countries. (A country is a [sovereign state (Q3624078)](https://www.wikidata.org/wiki/Q3624078), and the property for population is [P:P1082](https://www.wikidata.org/wiki/Property:P1082).) You can start by searching for countries with their population, and then add the ORDER BY and LIMIT clauses.

Here’s my solution:

**SELECT** ?country ?countryLabel ?population

**WHERE**

{

?country **wdt**:**P31**/**wdt**:**P279**\* **wd**:**Q3624078**;

**wdt**:**P1082** ?population.

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "en". }

}

**ORDER BY** **DESC**(?population)

**LIMIT** 10

[Run it!](https://query.wikidata.org/#SELECT%20%3Fcountry%20%3FcountryLabel%20%3Fpopulation%0AWHERE%0A%7B%0A%20%20%3Fcountry%20wdt%3AP31%2Fwdt%3AP279%2A%20wd%3AQ3624078%3B%0A%20%20%20%20%20%20%20%20%20%20%20wdt%3AP1082%20%3Fpopulation.%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22en%22.%20%7D%0A%7D%0AORDER%20BY%20DESC%28%3Fpopulation%29%0ALIMIT%2010)

Note that if we want the *most* populous countries, we have to order by *descending* population, so that the first results will be the ones with the highest values.

**Exercise**[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=10)]

We’ve covered a lot of ground so far – I think it’s time for some exercises. (You can skip this section if you’re in a hurry.)

**Arthur Conan Doyle books**[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=11)]

Write a query that returns all books by Sir Arthur Conan Doyle.

|  |
| --- |
| **Expand**  **Hint** |

|  |
| --- |
| **Expand**  **Example solution** |

**Chemical elements**[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=12)]

Write a query that returns all chemical elements with their element symbol and atomic number, in order of their atomic number.

|  |
| --- |
| **Expand**  **Hint** |

|  |
| --- |
| **Expand**  **Example solution** |

**Rivers that flow into the Mississippi**[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=13)]

Write a query that returns all rivers that flow directly into the Mississippi River. (The main challenge is finding the correct property…)

|  |
| --- |
| **Expand**  **Hint** |

|  |
| --- |
| **Expand**  **Example solution** |

**Rivers that flow into the Mississippi II**[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=14)]

Write a query that returns all rivers that flow into the Mississippi River, directly or indirectly.

|  |
| --- |
| **Expand**  **Hint** |

|  |
| --- |
| **Expand**  **Example solution** |

OPTIONAL[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=15)]

In the exercises above, we had a query for all books by Sir Arthur Conan Doyle:

**SELECT** ?book ?bookLabel

**WHERE**

{

?book **wdt**:**P50** **wd**:**Q35610**.

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fbook%20%3FbookLabel%0AWHERE%0A%7B%0A%20%20%3Fbook%20wdt%3AP50%20wd%3AQ35610.%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%22.%20%7D%0A%7D)

But that’s a bit boring. There’s so much potential data about books, and we only show the label? Let’s try to craft a query that also includes the [title (P1476)](https://www.wikidata.org/wiki/Property:P1476), [illustrator (P110)](https://www.wikidata.org/wiki/Property:P110), [publisher (P123)](https://www.wikidata.org/wiki/Property:P123) and [publication date (P577)](https://www.wikidata.org/wiki/Property:P577).

A first attempt might look like this:

**SELECT** ?book ?title ?illustratorLabel ?publisherLabel ?published

**WHERE**

{

?book **wdt**:**P50** **wd**:**Q35610**;

**wdt**:**P1476** ?title;

**wdt**:**P110** ?illustrator;

**wdt**:**P123** ?publisher;

**wdt**:**P577** ?published.

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fbook%20%3Ftitle%20%3FillustratorLabel%20%3FpublisherLabel%20%3Fpublished%0AWHERE%0A%7B%0A%20%20%3Fbook%20wdt%3AP50%20wd%3AQ35610%3B%0A%20%20%20%20%20%20%20%20wdt%3AP1476%20%3Ftitle%3B%0A%20%20%20%20%20%20%20%20wdt%3AP110%20%3Fillustrator%3B%0A%20%20%20%20%20%20%20%20wdt%3AP123%20%3Fpublisher%3B%0A%20%20%20%20%20%20%20%20wdt%3AP577%20%3Fpublished.%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%22.%20%7D%0A%7D)

Run that query. As I’m writing this, it only returns two results – a bit meager! Why is that? We found over a hundred books earlier!

The reason is that to match this query, a potential result (a book) must match all the triples we listed: it must have a title, and an illustrator, and a publisher, and a publication date. If it has some of those properties, but not all of them, it won’t match. And that’s not what we want in this case: we primarily want a list of all the books – if additional data is available, we’d like to include it, but we don’t want that to limit our list of results.

The solution is to tell WDQS that those triples are *optional*:

**SELECT** ?book ?title ?illustratorLabel ?publisherLabel ?published

**WHERE**

{

?book **wdt**:**P50** **wd**:**Q35610**.

**OPTIONAL** { ?book **wdt**:**P1476** ?title. }

**OPTIONAL** { ?book **wdt**:**P110** ?illustrator. }

**OPTIONAL** { ?book **wdt**:**P123** ?publisher. }

**OPTIONAL** { ?book **wdt**:**P577** ?published. }

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fbook%20%3Ftitle%20%3FillustratorLabel%20%3FpublisherLabel%20%3Fpublished%0AWHERE%0A%7B%0A%20%20%3Fbook%20wdt%3AP50%20wd%3AQ35610.%0A%20%20OPTIONAL%20%7B%20%3Fbook%20wdt%3AP1476%20%3Ftitle.%20%7D%0A%20%20OPTIONAL%20%7B%20%3Fbook%20wdt%3AP110%20%3Fillustrator.%20%7D%0A%20%20OPTIONAL%20%7B%20%3Fbook%20wdt%3AP123%20%3Fpublisher.%20%7D%0A%20%20OPTIONAL%20%7B%20%3Fbook%20wdt%3AP577%20%3Fpublished.%20%7D%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%22.%20%7D%0A%7D)

This gives us the additional variables (?title, ?publisher etc.) if the appropriate statement exists, but if the statement doesn’t exist, the result isn’t discarded – the variable simply isn’t set.

Note: it’s very important to use separate OPTIONAL clauses here. If you put all the triples into a single clause, like here –

**SELECT** ?book ?title ?illustratorLabel ?publisherLabel ?published

**WHERE**

{

?book **wdt**:**P50** **wd**:**Q35610**.

**OPTIONAL** {

?book **wdt**:**P1476** ?title;

**wdt**:**P110** ?illustrator;

**wdt**:**P123** ?publisher;

**wdt**:**P577** ?published.

}

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "en". }

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fbook%20%3Ftitle%20%3FillustratorLabel%20%3FpublisherLabel%20%3Fpublished%0AWHERE%0A%7B%0A%20%20%3Fbook%20wdt%3AP50%20wd%3AQ35610.%0A%20%20OPTIONAL%20%7B%0A%20%20%20%20%3Fbook%20wdt%3AP1476%20%3Ftitle%3B%0A%20%20%20%20%20%20%20%20%20%20wdt%3AP110%20%3Fillustrator%3B%0A%20%20%20%20%20%20%20%20%20%20wdt%3AP123%20%3Fpublisher%3B%0A%20%20%20%20%20%20%20%20%20%20wdt%3AP577%20%3Fpublished.%0A%20%20%7D%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22en%22.%20%7D%0A%7D)

– you’ll notice that most of the results don’t include any extra information. This is because an optional clause with multiple triples only matches when all those triples can be satisfied. That is: if a book has a title, an illustrator, a publisher, and a publication date, then the optional clause matches, and those values are assigned to the appropriate variables. But if a book has, for example, a title but no illustrator, the entire optional clause doesn’t match, and although the result isn’t discarded, all four variables remain empty.

Expressions, FILTER and BIND[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=16)]

This section might seem a bit less organized than the other ones, because it covers a fairly wide and diverse topic. The basic concept is that we would like to do something with the values that, so far, we’ve just selected and returned indiscriminately. And *expressions* are the way to express these operations on values. There are many kinds of expressions, and a lot of things you can do with them – but first, let’s start with the basics: data types.

**Data types**[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=17)]

Each value in SPARQL has a type, which tells you what kind of value it is and what you can do with it. The most important types are:

* item, like wd:Q42 for [Douglas Adams (Q42)](https://www.wikidata.org/wiki/Q42).
* boolean, with the two possible values true and false. Boolean values aren’t stored in statements, but many expressions return a boolean value, e.g. 2 < 3 (true) or "a" = "b" (false).
* string, a piece of text. String literals are written in double quotes.
* monolingual text, a string with a language tag attached. In a literal, you can add the language tag after the string with an @ sign, e.g. "Douglas Adams"@en.
* numbers, either integers (1) or decimals (1.23).
* dates. Date literals can be written by adding ^^xsd:dateTime (case sensitive – ^^xsd:date**t**ime won’t work!) to an [ISO 8601](https://en.wikipedia.org/wiki/ISO_8601) date string: "2012-10-29"^^xsd:dateTime.

**Operators**[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=18)]

The familiar mathematical operators are available: +, -, \*, / to add, subtract, multiply or divide numbers, <, >, =, <=, >= to compare them. The inequality test ≠ is written !=. Comparison is also defined for other types; for example, "abc" < "abd" is true (lexical comparison), as is "2016-01-01"^^xsd:dateTime > "2015-12-31"^^xsd:dateTime and wd:Q4653 != wd:Q283111. And boolean conditions can be combined with && (logical and: a && b is true if both a and b are true) and || (logical or: a || b is true if either (or both) of a and b is true).

**FILTER**[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=19)]

 **Info** For a sometimes faster alternative to FILTER, you might also look at MINUS, see [example](https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries#Excluding_subsets).

FILTER(condition). is a clause you can insert into your SPARQL query to filter the results. Inside the parentheses, you can put any expression of boolean type, and only those results where the expression returns true are used.

For example, to get a list of all humans born in 2015, we first get all humans with their date of birth –

**SELECT** ?person ?personLabel ?dob

**WHERE**

{

?person **wdt**:**P31** **wd**:**Q5**;

**wdt**:**P569** ?dob.

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "en". }

}

– and then filter that to only return the results where the year of the date of birth is 2015. There are two ways to do that: extract the year of the date with the YEAR function, and test that it’s 2015 –

**FILTER**(YEAR(?dob) = 2015).

– or check that the date is between Jan. 1st (inclusive), 2015 and Jan. 1st, 2016 (exclusive):

**FILTER**("2015-01-01"^^**xsd**:**dateTime** <= ?dob && ?dob < "2016-01-01"^^**xsd**:**dateTime**).

I’d say that the first one is more straightforward, but it turns out the second one is much faster, so let’s use that:

**SELECT** ?person ?personLabel ?dob

**WHERE**

{

?person **wdt**:**P31** **wd**:**Q5**;

**wdt**:**P569** ?dob.

**FILTER**("2015-01-01"^^**xsd**:**dateTime** <= ?dob && ?dob < "2016-01-01"^^**xsd**:**dateTime**).

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fperson%20%3FpersonLabel%20%3Fdob%0AWHERE%0A%7B%0A%20%20%3Fperson%20wdt%3AP31%20wd%3AQ5%3B%0A%20%20%20%20%20%20%20%20%20%20wdt%3AP569%20%3Fdob.%0A%20%20FILTER%28%222015-01-01%22%5E%5Exsd%3AdateTime%20%3C%3D%20%3Fdob%20%26%26%20%3Fdob%20%3C%20%222016-01-01%22%5E%5Exsd%3AdateTime%29.%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%22.%20%7D%20%0A%7D)

Another possible use of FILTER is related to labels. The label service is very useful if you just want to display the label of a variable. But if you want to do stuff with the label – for example: check if it starts with “Mr. ” – you’ll find that it doesn’t work:

**SELECT** ?human ?humanLabel

**WHERE**

{

?human **wdt**:**P31** **wd**:**Q15632617**.

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "en". }

*#This FILTER does not work!*

**FILTER**(STRSTARTS(?humanLabel, "Mr. ")).

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fhuman%20%3FhumanLabel%0AWHERE%0A%7B%0A%20%20%3Fhuman%20wdt%3AP31%20wd%3AQ15632617.%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22en%22.%20%7D%0A%20%20%23This%20FILTER%20does%20not%20work%21%0A%20%20FILTER%28STRSTARTS%28%3FhumanLabel%2C%20%22Mr.%20%22%29%29.%0A%7D)

This query finds all instances of [fictional human (Q15632617)](https://www.wikidata.org/wiki/Q15632617) and tests if their label starts with "Mr. " (STRSTARTS is short for “string starts [with]”; there’s also STRENDS and CONTAINS). The reason why this doesn’t work is that the label service adds its variables very late during query evaluation; at the point where we try to filter on ?humanLabel, the label service hasn’t created that variable yet.

Fortunately, the label service isn’t the only way to get an item’s label. Labels are also stored as regular triples, using the predicate rdfs:label. Of course, this means all labels, not just English ones; if we only want English labels, we’ll have to filter on the language of the label:

**FILTER**(LANG(?label) = "en").

The LANG function returns the language of a monolingual string, and here we only select those labels that are in English. The full query is:

**SELECT** ?human ?label

**WHERE**

{

?human **wdt**:**P31** **wd**:**Q15632617**;

**rdfs**:**label** ?label.

**FILTER**(LANG(?label) = "[AUTO\_LANGUAGE]").

**FILTER**(STRSTARTS(?label, "Mr. ")).

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fhuman%20%3Flabel%0AWHERE%0A%7B%0A%20%20%3Fhuman%20wdt%3AP31%20wd%3AQ15632617%3B%0A%20%20%20%20%20%20%20%20%20rdfs%3Alabel%20%3Flabel.%0A%20%20FILTER%28LANG%28%3Flabel%29%20%3D%20%22%5BAUTO_LANGUAGE%5D%22%29.%0A%20%20FILTER%28STRSTARTS%28%3Flabel%2C%20%22Mr.%20%22%29%29.%0A%7D)

We get the label with the ?human rdfs:label ?label triple, restrict it to English labels, and then check if it starts with “Mr. ”.

One can also use FILTER with a regular expression. In the following example

**SELECT** ?item ?itemLabel ?bblid

**WHERE** {

?item **wdt**:**P2580** ?bblid .

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE],en" }

**FILTER**(!REGEX(STR(?bblid), "[\**\**.q]"))

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fitem%20%3FitemLabel%20%3Fbblid%0AWHERE%20%7B%20%20%0A%20%20%20%20%3Fitem%20wdt%3AP2580%20%3Fbblid%20.%0A%20%20%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%2Cen%22%20%7D%20%20%0A%20%20%20%20FILTER%28%21REGEX%28STR%28%3Fbblid%29%2C%20%22%5B%5C%5C.q%5D%22%29%29%20%0A%7D)

If the format constraint for an ID is [A-Za-z][-.0-9A-Za-z]{1,}:

**SELECT** ?item ?itemLabel ?bblid

**WHERE** {

?item **wdt**:**P2580** ?bblid .

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE],en" }

**FILTER**(!REGEX(STR(?bblid), "^[A-Za-z][-.0-9A-Za-z]{1,}$"))

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fitem%20%3FitemLabel%20%3Fbblid%0AWHERE%20%7B%20%20%0A%20%20%20%20%3Fitem%20wdt%3AP2580%20%3Fbblid%20.%0A%20%20%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%2Cen%22%20%7D%20%20%0A%20%20%20%20FILTER%28%21REGEX%28STR%28%3Fbblid%29%2C%20%22%5E%5BA-Za-z%5D%5B-.0-9A-Za-z%5D%7B1%2C%7D%24%22%29%29%0A%7D)

It is possible to filter out specific elements like this

**FILTER** ( ?item **not in** ( **wd**:**Q4115189**,**wd**:**Q13406268**,**wd**:**Q15397819** ) )

It is possible to filter and have elements that aren't filled:

**FILTER** ( NOT EXISTS { ?item **wdt**:**P21** [] } )

**BIND, BOUND, IF**[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=20)]

These three features are often used in conjunction, so I’ll first explain all three of them and then show you some examples.

A BIND(expression AS ?variable). clause can be used to assign the result of an expression to a variable (usually a new variable, but you can also overwrite existing ones).

BOUND(?variable) tests if a variable has been bound to a value (returns true or false). It’s mostly useful on variables that are introduced in an OPTIONAL clause.

IF(condition,thenExpression,elseExpression) evaluates to thenExpression if condition evaluates to true, and to elseExpression if condition evaluates to false. That is, IF(true, "yes", "no") evaluates to "yes", and IF(false, "great", "terrible") evaluates to "terrible".

BIND can be used to bind the results of some calculation to a new variable. This can be an intermediate result of a larger calculation or just directly a result of the query. For example, to get the age of victims of capital punishment:

**SELECT** ?person ?personLabel ?age

**WHERE**

{

?person **wdt**:**P31** **wd**:**Q5**;

**wdt**:**P569** ?born;

**wdt**:**P570** ?died;

**wdt**:**P1196** **wd**:**Q8454**.

**BIND**(?died - ?born **AS** ?ageInDays).

**BIND**(?ageInDays/365.2425 **AS** ?ageInYears).

**BIND**(FLOOR(?ageInYears) **AS** ?age).

*# or, as one expression:*

*#BIND(FLOOR((?died - ?born)/365.2425) AS ?age).*

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fperson%20%3FpersonLabel%20%3Fage%0AWHERE%0A%7B%0A%20%20%3Fperson%20wdt%3AP31%20wd%3AQ5%3B%0A%20%20%20%20%20%20%20%20%20%20wdt%3AP569%20%3Fborn%3B%0A%20%20%20%20%20%20%20%20%20%20wdt%3AP570%20%3Fdied%3B%0A%20%20%20%20%20%20%20%20%20%20wdt%3AP1196%20wd%3AQ8454.%0A%20%20BIND%28%3Fdied%20-%20%3Fborn%20AS%20%3FageInDays%29.%0A%20%20BIND%28%3FageInDays%2F365.2425%20AS%20%3FageInYears%29.%0A%20%20BIND%28FLOOR%28%3FageInYears%29%20AS%20%3Fage%29.%0A%20%20%23%20or%2C%20as%20one%20expression%3A%0A%20%20%23BIND%28FLOOR%28%28%3Fdied%20-%20%3Fborn%29%2F365.2425%29%20AS%20%3Fage%29.%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%22.%20%7D%0A%7D)

BIND can also be used to simply bind constant values to variables in order to increase readability. For example, a query that finds all female priests:

**SELECT** ?woman ?womanLabel

**WHERE**

{

?woman **wdt**:**P31** **wd**:**Q5**;

**wdt**:**P21** **wd**:**Q6581072**;

**wdt**:**P106** **wd**:**Q42603**.

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fwoman%20%3FwomanLabel%0AWHERE%0A%7B%0A%20%20%3Fwoman%20wdt%3AP31%20wd%3AQ5%3B%0A%20%20%20%20%20%20%20%20%20wdt%3AP21%20wd%3AQ6581072%3B%0A%20%20%20%20%20%20%20%20%20wdt%3AP106%20wd%3AQ42603.%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%22.%20%7D%0A%7D)

can be rewritten like this:

**SELECT** ?woman ?womanLabel

**WHERE**

{

**BIND**(**wdt**:**P31** **AS** ?instanceOf).

**BIND**(**wd**:**Q5** **AS** ?human).

**BIND**(**wdt**:**P21** **AS** ?sexOrGender).

**BIND**(**wd**:**Q6581072** **AS** ?female).

**BIND**(**wdt**:**P106** **AS** ?occupation).

**BIND**(**wd**:**Q42603** **AS** ?priest).

?woman ?instanceOf ?human;

?sexOrGender ?female;

?occupation ?priest.

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fwoman%20%3FwomanLabel%0AWHERE%0A%7B%0A%20%20BIND%28wdt%3AP31%20AS%20%3FinstanceOf%29.%0A%20%20BIND%28wd%3AQ5%20AS%20%3Fhuman%29.%0A%20%20BIND%28wdt%3AP21%20AS%20%3FsexOrGender%29.%0A%20%20BIND%28wd%3AQ6581072%20AS%20%3Ffemale%29.%0A%20%20BIND%28wdt%3AP106%20AS%20%3Foccupation%29.%0A%20%20BIND%28wd%3AQ42603%20AS%20%3Fpriest%29.%0A%20%20%3Fwoman%20%3FinstanceOf%20%3Fhuman%3B%0A%20%20%20%20%20%20%20%20%20%3FsexOrGender%20%3Ffemale%3B%0A%20%20%20%20%20%20%20%20%20%3Foccupation%20%3Fpriest.%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%22.%20%7D%0A%7D)

The meaningful part of the query, from ?woman to ?priest., is now probably more readable. However, the large BIND block right in front of it is pretty distracting, so this technique should be used sparingly. (In the WDQS user interface, you can also hover your mouse over any term like wd:Q123 or wdt:P123 and see the label and description for the entity, so ?female is only more readable than wd:Q6581072 if you ignore that feature.)

IF expressions are often used with condition-expressions built with BOUND. For example, suppose you have a query that shows some humans, and instead of just showing their label, you’d like to display their [pseudonym (P742)](https://www.wikidata.org/wiki/Property:P742) if they have one, and only use the label if a pseudonym doesn’t exist. For this, you select the pseudonym in an OPTIONAL clause (it has to be optional – you don’t want to throw out results that don’t have a pseudonym), and then use BIND(IF(BOUND(… to select either the pseudonym or the label.

**SELECT** ?writer ?label

**WHERE**

{

*# French writer born in the second half of the 18th century*

?writer **wdt**:**P31** **wd**:**Q5**;

**wdt**:**P27** **wd**:**Q142**;

**wdt**:**P106** **wd**:**Q36180**;

**wdt**:**P569** ?dob.

**FILTER**("1751-01-01"^^**xsd**:**dateTime** <= ?dob && ?dob < "1801-01-01"^^**xsd**:**dateTime**).

*# get the English label*

?writer **rdfs**:**label** ?writerLabel.

**FILTER**(LANG(?writerLabel) = "en").

*# get the pseudonym, if it exists*

**OPTIONAL** { ?writer **wdt**:**P742** ?pseudonym. }

*# bind the pseudonym, or if it doesn’t exist the English label, as ?label*

**BIND**(IF(BOUND(?pseudonym),?pseudonym,?writerLabel) **AS** ?label).

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fwriter%20%3Flabel%0AWHERE%0A%7B%0A%20%20%23%20French%20writer%20born%20in%20the%20second%20half%20of%20the%2018th%20century%0A%20%20%3Fwriter%20wdt%3AP31%20wd%3AQ5%3B%0A%20%20%20%20%20%20%20%20%20%20wdt%3AP27%20wd%3AQ142%3B%0A%20%20%20%20%20%20%20%20%20%20wdt%3AP106%20wd%3AQ36180%3B%0A%20%20%20%20%20%20%20%20%20%20wdt%3AP569%20%3Fdob.%0A%20%20FILTER%28%221751-01-01%22%5E%5Exsd%3AdateTime%20%3C%3D%20%3Fdob%20%26%26%20%3Fdob%20%3C%20%221801-01-01%22%5E%5Exsd%3AdateTime%29.%0A%20%20%23%20get%20the%20English%20label%0A%20%20%3Fwriter%20rdfs%3Alabel%20%3FwriterLabel.%0A%20%20FILTER%28LANG%28%3FwriterLabel%29%20%3D%20%22en%22%29.%0A%20%20%23%20get%20the%20pseudonym%2C%20if%20it%20exists%0A%20%20OPTIONAL%20%7B%20%3Fwriter%20wdt%3AP742%20%3Fpseudonym.%20%7D%0A%20%20%23%20bind%20the%20pseudonym%2C%20or%20if%20it%20doesn%E2%80%99t%20exist%20the%20English%20label%2C%20as%20%3Flabel%0A%20%20BIND%28IF%28BOUND%28%3Fpseudonym%29%2C%3Fpseudonym%2C%3FwriterLabel%29%20AS%20%3Flabel%29.%0A%7D)

Other properties that may be used in this way include [nickname (P1449)](https://www.wikidata.org/wiki/Property:P1449), [posthumous name (P1786)](https://www.wikidata.org/wiki/Property:P1786), and [taxon common name (P1843)](https://www.wikidata.org/wiki/Property:P1843) – anything where some sort of “fallback” makes sense.

You can also combine BOUND with FILTER to ensure that at least one of several OPTIONAL blocks has been fulfilled. For example, let’s get all astronauts that went to the moon, as well as the members of [Apollo 13 (Q182252)](https://www.wikidata.org/wiki/Q182252) (close enough, right?). That restriction can’t be expressed as a single property path, so we need one OPTIONAL clause for “member of some moon mission” and another one for “member of Apollo 13”. But we only want to select those results where at least one of those conditions is true.

**SELECT** ?astronaut ?astronautLabel

**WHERE**

{

?astronaut **wdt**:**P31** **wd**:**Q5**;

**wdt**:**P106** **wd**:**Q11631**.

**OPTIONAL** {

?astronaut **wdt**:**P450** ?mission.

?mission **wdt**:**P31** **wd**:**Q495307**.

}

**OPTIONAL** {

?astronaut **wdt**:**P450** **wd**:**Q182252**.

**BIND**(**wd**:**Q182252** **AS** ?mission).

}

**FILTER**(BOUND(?mission)).

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fastronaut%20%3FastronautLabel%0AWHERE%0A%7B%0A%20%20%3Fastronaut%20wdt%3AP31%20wd%3AQ5%3B%0A%20%20%20%20%20%20%20%20%20%20%20%20%20wdt%3AP106%20wd%3AQ11631.%0A%20%20OPTIONAL%20%7B%0A%20%20%20%20%3Fastronaut%20wdt%3AP450%20%3Fmission.%0A%20%20%20%20%3Fmission%20wdt%3AP31%20wd%3AQ495307.%0A%20%20%7D%0A%20%20OPTIONAL%20%7B%0A%20%20%20%20%3Fastronaut%20wdt%3AP450%20wd%3AQ182252.%0A%20%20%20%20BIND%28wd%3AQ182252%20AS%20%3Fmission%29.%0A%20%20%7D%0A%20%20FILTER%28BOUND%28%3Fmission%29%29.%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%22.%20%7D%0A%7D)

**COALESCE**[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=21)]

The COALESCE function can be used as an abbreviation of the BIND(IF(BOUND(?x), ?x, ?y) AS ?z). pattern for fallbacks mentioned above: it takes a number of expressions and returns the first one that evaluates without error. For example, the above “pseudonym” fallback

**BIND**(IF(BOUND(?pseudonym),?pseudonym,?writerLabel) **AS** ?label).

can be written more concisely as

**BIND**(COALESCE(?pseudonym, ?writerLabel) **AS** ?label).

and it’s also easy to add another fallback label in case the ?writerLabel isn’t defined either:

**BIND**(COALESCE(?pseudonym, ?writerLabel, "<no label>") **AS** ?label).

Grouping[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=22)]

So far, all the queries we’ve seen were queries that found all items satisfying some conditions; in some cases, we also included extra statements on the item (paintings with materials, Arthur Conan Doyle books with title and illustrator).

But it’s very common that we don’t want a long list of all results. Instead, we might ask questions like this:

* *How many* paintings were painted on canvas / poplar wood / etc.?
* What is the *highest* population of each country’s cities?
* What is the *total number* of guns produced by each manufacturer?
* Who publishes, *on average*, the longest books?

**City populations**[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=23)]

Let’s look at the second question for now. It’s fairly simple to write a query that lists all cities along with their population and country, ordered by country:

**SELECT** ?country ?city ?population

**WHERE**

{

?city **wdt**:**P31**/**wdt**:**P279**\* **wd**:**Q515**;

**wdt**:**P17** ?country;

**wdt**:**P1082** ?population.

}

**ORDER BY** ?country

[Run it!](https://query.wikidata.org/#SELECT%20%3Fcountry%20%3Fcity%20%3Fpopulation%0AWHERE%0A%7B%0A%20%20%3Fcity%20wdt%3AP31%2Fwdt%3AP279%2A%20wd%3AQ515%3B%0A%20%20%20%20%20%20%20%20wdt%3AP17%20%3Fcountry%3B%0A%20%20%20%20%20%20%20%20wdt%3AP1082%20%3Fpopulation.%0A%7D%0AORDER%20BY%20%3Fcountry)

(Note: that query returns a *lot* of results, which might cause trouble for your browser. You might want to add a LIMIT clause.)

Since we’re ordering the results by country, all cities belonging to a country form one contiguous block in the results. To find the highest population within that block, we want to consider the block as a *group*, and *aggregate* all the individual population values into one value: the maximum. This is done with a GROUP BY clause below the WHERE block, and an aggregate function (MAX) in the SELECT clause.

**SELECT** ?country (MAX(?population) **AS** ?maxPopulation)

**WHERE**

{

?city **wdt**:**P31**/**wdt**:**P279**\* **wd**:**Q515**;

**wdt**:**P17** ?country;

**wdt**:**P1082** ?population.

}

**GROUP BY** ?country

[Run it!](https://query.wikidata.org/#SELECT%20%3Fcountry%20%28MAX%28%3Fpopulation%29%20AS%20%3FmaxPopulation%29%0AWHERE%0A%7B%0A%20%20%3Fcity%20wdt%3AP31%2Fwdt%3AP279%2A%20wd%3AQ515%3B%0A%20%20%20%20%20%20%20%20wdt%3AP17%20%3Fcountry%3B%0A%20%20%20%20%20%20%20%20wdt%3AP1082%20%3Fpopulation.%0A%7D%0AGROUP%20BY%20%3Fcountry)

We’ve replaced the ORDER BY with a GROUP BY. The effect of this is that all results with the same ?country are now grouped together into a single result. This means that we have to change the SELECT clause as well. If we kept the old clause SELECT ?country ?city ?population, which ?city and ?population would be returned? Remember, there are many results in this one result; they all have the same ?country, so we can select that, but since they can all have a different ?city and ?population, we have to tell WDQS which of those values to select. That’s the job of the *aggregate function*. In this case, we’ve used MAX: out of all the ?population values, we select the maximum one for the group result. (We also have to give that value a new name with the AS construct, but that’s just a minor detail.)

This is the general pattern for writing group queries: write a normal query that returns the data you want (not grouped, with many results per “group”), then add a GROUP BY clause and add an aggregate function all the non-grouped variables in the SELECT clause.

**Painting materials**[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=24)]

Let’s try it out with another question: How many paintings were painted on each material? First, write a query that just returns all paintings along with their painting material. (Take care to only use those [made from material (P186)](https://www.wikidata.org/wiki/Property:P186) statements with an [applies to part (P518)](https://www.wikidata.org/wiki/Property:P518)[painting support (Q861259)](https://www.wikidata.org/wiki/Q861259) qualifier.)

**SELECT** ?material ?painting

**WHERE**

{

?painting **wdt**:**P31**/**wdt**:**P279**\* **wd**:**Q3305213**;

**p**:**P186** [ **ps**:**P186** ?material; **pq**:**P518** **wd**:**Q861259** ].

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fmaterial%20%3Fpainting%0AWHERE%0A%7B%0A%20%20%3Fpainting%20wdt%3AP31%2Fwdt%3AP279%2A%20wd%3AQ3305213%3B%0A%20%20%20%20%20%20%20%20%20%20%20%20p%3AP186%20%5B%20ps%3AP186%20%3Fmaterial%3B%20pq%3AP518%20wd%3AQ861259%20%5D.%0A%7D)

Next, add a GROUP BY clause on the ?material, and then an aggregate function on the other selected variable (?painting). In this case, we are interested in the number of paintings; the aggregate function for that is COUNT.

**SELECT** ?material (COUNT(?painting) **AS** ?count)

**WHERE**

{

?painting **wdt**:**P31**/**wdt**:**P279**\* **wd**:**Q3305213**;

**p**:**P186** [ **ps**:**P186** ?material; **pq**:**P518** **wd**:**Q861259** ].

}

**GROUP BY** ?material

[Run it!](https://query.wikidata.org/#SELECT%20%3Fmaterial%20%28COUNT%28%3Fpainting%29%20AS%20%3Fcount%29%0AWHERE%0A%7B%0A%20%20%3Fpainting%20wdt%3AP31%2Fwdt%3AP279%2A%20wd%3AQ3305213%3B%0A%20%20%20%20%20%20%20%20%20%20%20%20p%3AP186%20%5B%20ps%3AP186%20%3Fmaterial%3B%20pq%3AP518%20wd%3AQ861259%20%5D.%0A%7D%0AGROUP%20BY%20%3Fmaterial)

One problem with this is that we don’t have the label for the materials, so the results are a bit inconvenient to interpret. If we just add the label variable, we’ll get an error:

**SELECT** ?material ?materialLabel (COUNT(?painting) **AS** ?count)

**WHERE**

{

?painting **wdt**:**P31**/**wdt**:**P279**\* **wd**:**Q3305213**;

**p**:**P186** [ **ps**:**P186** ?material; **pq**:**P518** **wd**:**Q861259** ].

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

**GROUP BY** ?material

[Run it!](https://query.wikidata.org/#SELECT%20%3Fmaterial%20%3FmaterialLabel%20%28COUNT%28%3Fpainting%29%20AS%20%3Fcount%29%0AWHERE%0A%7B%0A%20%20%3Fpainting%20wdt%3AP31%2Fwdt%3AP279%2A%20wd%3AQ3305213%3B%0A%20%20%20%20%20%20%20%20%20%20%20%20p%3AP186%20%5B%20ps%3AP186%20%3Fmaterial%3B%20pq%3AP518%20wd%3AQ861259%20%5D.%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%22.%20%7D%0A%7D%0AGROUP%20BY%20%3Fmaterial)

**Bad aggregate**

“Bad aggregate” is an error message you’ll probably see a lot when working with group queries; it means that one of the selected variables needs an aggregate function but doesn’t have one, or it has an aggregate function but isn’t supposed to have one. In this case, WDQS thinks that there might be multiple ?materialLabels per ?material (even though we know that can’t happen), and so it complains that you’re not specifying an aggregate function for that variable.

One solution is to group over multiple variables. If you list multiple variables in the GROUP BY clause, there’s one result for each combination of those variables, and you can select all those variables without aggregate function. In this case, we’ll group over both ?material and ?materialLabel.

**SELECT** ?material ?materialLabel (COUNT(?painting) **AS** ?count)

**WHERE**

{

?painting **wdt**:**P31**/**wdt**:**P279**\* **wd**:**Q3305213**;

**p**:**P186** [ **ps**:**P186** ?material; **pq**:**P518** **wd**:**Q861259** ].

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

**GROUP BY** ?material ?materialLabel

[Run it!](https://query.wikidata.org/#SELECT%20%3Fmaterial%20%3FmaterialLabel%20%28COUNT%28%3Fpainting%29%20AS%20%3Fcount%29%0AWHERE%0A%7B%0A%20%20%3Fpainting%20wdt%3AP31%2Fwdt%3AP279%2A%20wd%3AQ3305213%3B%0A%20%20%20%20%20%20%20%20%20%20%20%20p%3AP186%20%5B%20ps%3AP186%20%3Fmaterial%3B%20pq%3AP518%20wd%3AQ861259%20%5D.%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%22.%20%7D%0A%7D%0AGROUP%20BY%20%3Fmaterial%20%3FmaterialLabel)

We’re almost done with the query – just one more improvement: we’d like to see the most-used materials first. Fortunately, we’re allowed to use the new, aggregated variables from the SELECT clause (here, ?count) in an ORDER BY clause, so this is very simple:

**SELECT** ?material ?materialLabel (COUNT(?painting) **AS** ?count)

**WHERE**

{

?painting **wdt**:**P31**/**wdt**:**P279**\* **wd**:**Q3305213**;

**p**:**P186** [ **ps**:**P186** ?material; **pq**:**P518** **wd**:**Q861259** ].

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

**GROUP BY** ?material ?materialLabel

**ORDER BY** **DESC**(?count)

[Run it!](https://query.wikidata.org/#SELECT%20%3Fmaterial%20%3FmaterialLabel%20%28COUNT%28%3Fpainting%29%20AS%20%3Fcount%29%0AWHERE%0A%7B%0A%20%20%3Fpainting%20wdt%3AP31%2Fwdt%3AP279%2A%20wd%3AQ3305213%3B%0A%20%20%20%20%20%20%20%20%20%20%20%20p%3AP186%20%5B%20ps%3AP186%20%3Fmaterial%3B%20pq%3AP518%20wd%3AQ861259%20%5D.%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%22.%20%7D%0A%7D%0AGROUP%20BY%20%3Fmaterial%20%3FmaterialLabel%0AORDER%20BY%20DESC%28%3Fcount%29)

As an exercise, let’s do the other queries too.

**Guns by manufacturer**[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=25)]

What is the total number of guns produced by each manufacturer?

|  |
| --- |
| **Expand**  **Hint** |

|  |
| --- |
| **Expand**  **Example solution** |

**Publishers by number of pages**[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=26)]

What is the average (function: AVG) number of pages of books by each publisher?

|  |
| --- |
| **Expand**  **Hint** |

|  |
| --- |
| **Expand**  **Example solution** |

**HAVING**[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=27)]

A small addendum to that last query – if you look at the results, you might notice that the top result has an outrageously large average, over ten times that of the second place. A bit of investigation reveals that this is because that publisher ([UTET (Q4002388)](https://www.wikidata.org/wiki/Q4002388)) only published a single book with a [number of pages (P1104)](https://www.wikidata.org/wiki/Property:P1104) statement, [Grande dizionario della lingua italiana (Q3775610)](https://www.wikidata.org/wiki/Q3775610), which skews the results a bit. To remove outliers like that, we could try to select only publishers that published at least two books with [number of pages (P1104)](https://www.wikidata.org/wiki/Property:P1104) statements on Wikidata.

How do we do that? Normally, we restrict results with a FILTER clause, but in this case we want to restrict based on the group (the number of books), not any individual result. This is done with a HAVING clause, which can be placed right after a GROUP BY clause and takes an expression just like FILTER does:

**SELECT** ?publisher ?publisherLabel (AVG(?pages) **AS** ?avgPages)

**WHERE**

{

?book **wdt**:**P123** ?publisher;

**wdt**:**P1104** ?pages.

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE]". }

}

**GROUP BY** ?publisher ?publisherLabel

**HAVING**(COUNT(?book) > 1)

**ORDER BY** **DESC**(?avgPages)

[Run it!](https://query.wikidata.org/#SELECT%20%3Fpublisher%20%3FpublisherLabel%20%28AVG%28%3Fpages%29%20AS%20%3FavgPages%29%0AWHERE%0A%7B%0A%20%20%3Fbook%20wdt%3AP123%20%3Fpublisher%3B%0A%20%20%20%20%20%20%20%20wdt%3AP1104%20%3Fpages.%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%22.%20%7D%0A%7D%0AGROUP%20BY%20%3Fpublisher%20%3FpublisherLabel%0AHAVING%28COUNT%28%3Fbook%29%20%3E%201%29%0AORDER%20BY%20DESC%28%3FavgPages%29)

**Aggregate functions summary**[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=28)]

Here’s a short summary of the available aggregate functions:

* COUNT: the number of elements. You can also write COUNT(\*) to simply count all results.
* SUM, AVG: the sum or average of all elements, respectively. If the elements aren’t numbers, you’ll get weird results.
* MIN, MAX: the minimum or maximum value of all elements, respectively. This works for all value types; numbers are sorted numerically, strings and other types lexically.
* SAMPLE: any element. This is occasionally useful if you know there’s only one result, or if you don’t care which one is returned.
* GROUP\_CONCAT: concatenates all elements. Useful for example if you want only one result for an item but you want to include informations for a property that may have several statements for this item, such as the occupations of a person. The different occupations may be regrouped and concatenated to appear all in only one variable instead of several lines in the results. If you’re curious, you can look it up in the [SPARQL specification](https://www.w3.org/TR/2013/REC-sparql11-query-20130321/#defn_aggGroupConcat).

Additionally, you can add a DISTINCT modifier for any of these functions to eliminate duplicate results. For example, if there are two results but they both have the same value in ?var, then COUNT(?var) will return 2 but COUNT(DISTINCT ?var) will only return 1. You often have to use DISTINCT when your query can return the same item multiple times – this can happen if, for example, you use ?item wdt:P31/wdt:P279\* ?class, and there are multiple paths from ?item to ?class: you will get a new result for each of those paths, even though all the values in the result are identical. (If you’re not grouping, you can also eliminate those duplicate results by starting the query with SELECT DISTINCT instead of just SELECT.)

**wikibase:Label and aggregations**[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=29)]

A query such as the following, which searches all academic persons with more than two countries of citizenships in Wikidata, does not show the names of those countries in the ?citizenships column:

**select** ?person ?personLabel (group\_concat(?citizenshipLabel;separator="/") **as** ?citizenships) {

*# find all academics*

?person **wdt**:**P106** **wd**:**Q3400985** ;

**wdt**:**P27** ?citizenship .

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE],en". }

} **group by** ?person ?personLabel **having** (count(?citizenship) > 2)

[Run it!](https://query.wikidata.org/#select%20%3Fperson%20%3FpersonLabel%20%28group_concat%28%3FcitizenshipLabel%3Bseparator%3D%22%2F%22%29%20as%20%3Fcitizenships%29%20%7B%0A%20%20%23%20find%20all%20academics%0A%20%20%3Fperson%20wdt%3AP106%20wd%3AQ3400985%20%3B%20%20%20%0A%20%20%20%20%20%20%20%20%20%20wdt%3AP27%20%20%3Fcitizenship%20.%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%2Cen%22.%20%7D%0A%7D%20group%20by%20%3Fperson%20%3FpersonLabel%20having%20%28count%28%3Fcitizenship%29%20%3E%202%29)

To show the ?citizenships, explicitely name the ?personLabel and ?citizenshipLabel in the wikibase:label service call like this:

**SERVICE** **wikibase**:**label** {

**bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE],en".

?citizenship **rdfs**:**label** ?citizenshipLabel .

?person **rdfs**:**label** ?personLabel .

}

The following query works as expected:

**select** ?person ?personLabel (group\_concat(?citizenshipLabel;separator="/") **as** ?citizenships) {

?person **wdt**:**P106** **wd**:**Q3400985** ;

**wdt**:**P27** ?citizenship .

**SERVICE** **wikibase**:**label** {

**bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE],en".

?citizenship **rdfs**:**label** ?citizenshipLabel .

?person **rdfs**:**label** ?personLabel .

}

} **group by** ?person ?personLabel **having** (count(?citizenship) > 2)

[Run it!](https://query.wikidata.org/#select%20%3Fperson%20%3FpersonLabel%20%28group_concat%28%3FcitizenshipLabel%3Bseparator%3D%22%2F%22%29%20as%20%3Fcitizenships%29%20%7B%0A%20%20%3Fperson%20wdt%3AP106%20wd%3AQ3400985%20%3B%0A%20%20%20%20%20%20%20%20%20%20wdt%3AP27%20%20%3Fcitizenship%20.%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20%0A%20%20%20%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%2Cen%22.%20%0A%20%20%20%20%3Fcitizenship%20rdfs%3Alabel%20%3FcitizenshipLabel%20.%0A%20%20%20%20%3Fperson%20%20%20%20%20%20rdfs%3Alabel%20%3FpersonLabel%20.%0A%20%20%7D%0A%7D%20group%20by%20%3Fperson%20%3FpersonLabel%20having%20%28count%28%3Fcitizenship%29%20%3E%202%29)

VALUES[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=30)]

One can select items based on a list of items:

**SELECT** ?item ?itemLabel ?mother ?motherLabel **WHERE** {

**VALUES** ?item { **wd**:**Q937** **wd**:**Q1339** }

**OPTIONAL** { ?item **wdt**:**P25** ?mother. }

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE],en". }

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fitem%20%3FitemLabel%20%3Fmother%20%3FmotherLabel%20WHERE%20%7B%0A%20%20VALUES%20%3Fitem%20%7B%20wd%3AQ937%20wd%3AQ1339%20%7D%0A%20%20OPTIONAL%20%7B%20%3Fitem%20wdt%3AP25%20%3Fmother.%20%7D%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%2Cen%22.%20%7D%0A%7D)

One can also select based on a list of values of a specific property:

**SELECT** ?item ?itemLabel ?mother ?motherLabel ?ISNI **WHERE** {

**VALUES** ?ISNI { "0000 0001 2281 955X" "0000 0001 2276 4157" }

?item **wdt**:**P213** ?ISNI.

**OPTIONAL** { ?item **wdt**:**P25** ?mother. }

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE],en". }

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fitem%20%3FitemLabel%20%3Fmother%20%3FmotherLabel%20%3FISNI%20WHERE%20%7B%0A%20%20VALUES%20%3FISNI%20%7B%20%220000%200001%202281%20955X%22%20%220000%200001%202276%204157%22%20%7D%0A%20%20%3Fitem%20wdt%3AP213%20%3FISNI.%0A%20%20OPTIONAL%20%7B%20%3Fitem%20wdt%3AP25%20%3Fmother.%20%7D%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%2Cen%22.%20%7D%0A%7D)

VALUES can also do more and build enumerations of values possible for a couple (or a tuple) of variables. For example say you want to use (known) custom labels for the persons enumerated in the first « value » example. It’s then possible to use a « values » clause such as VALUES (?item ?customItemLabel) { (wd:Q937 "Einstein") (wd:Q1339 "Bach") } which ensures that whenever ?item has value wd:Q937 in a result, ?customItemLabel own value is Einstein and whenever ?item has value wd:Q1339, ?customItemLabel’s value is Bach.

**SELECT** ?item ?customItemLabel ?mother ?motherLabel **WHERE** {

**VALUES** (?item ?customItemLabel) { (**wd**:**Q937** "Einstein") (**wd**:**Q1339** "Bach") }

**OPTIONAL** { ?item **wdt**:**P25** ?mother. }

**SERVICE** **wikibase**:**label** { **bd**:**serviceParam** **wikibase**:**language** "[AUTO\_LANGUAGE],en". }

}

[Run it!](https://query.wikidata.org/#SELECT%20%3Fitem%20%3FcustomItemLabel%20%3Fmother%20%3FmotherLabel%20WHERE%20%7B%0A%20%20VALUES%20%28%3Fitem%20%3FcustomItemLabel%29%20%7B%20%28wd%3AQ937%20%22Einstein%22%29%20%28wd%3AQ1339%20%22Bach%22%29%20%7D%0A%20%20OPTIONAL%20%7B%20%3Fitem%20wdt%3AP25%20%3Fmother.%20%7D%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%2Cen%22.%20%7D%0A%7D)

And beyond…[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=31)]

This guide ends here. SPARQL doesn’t: there’s still a lot that I haven’t shown you – I never promised this was going to be a complete guide! If you got this far, you already know a lot about WDQS and should be able to write some very powerful queries. But if you want to learn even more, here are some things you can look at:

* Subqueries. You add another entire query in curly brackets ({ SELECT ... WHERE { ... } LIMIT 10 }), and the results are visible in the outer query. (If you’re familiar with SQL, you’ll have to rethink the concept a bit – SPARQL subqueries are purely “bottom-up” and can’t use values from the outer query, like SQL “correlated subqueries” can.)
* MINUS lets you select results that *don’t* fit some graph pattern. FILTER NOT EXISTS is mostly equivalent (see the [SPARQL spec](https://www.w3.org/TR/sparql11-query/#neg-notexists-minus) for an example where they differ), but – at least on WDQS – usually slower by quite a bit.

Your main reference for these and other topics is the [SPARQL specification](https://www.w3.org/TR/sparql11-query/).

Also, you can take a look at [SPARQL tutorial](https://en.wikibooks.org/wiki/en:SPARQL) on Wikibooks and [this tutorial by data.world](https://docs.data.world/tutorials/sparql/).

And of course, there are some parts of Wikidata still missing as well, such as references, numeric precision (100±2.5), values with units (two kilograms), geocoordinates, sitelinks, statements on properties, and more. You can see how those are modeled as triples under [mw:Wikibase/Indexing/RDF Dump Format](https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format).

See also[[edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit&section=32)]

* [Wikidata:SPARQL query service](https://www.wikidata.org/wiki/Special:MyLanguage/Wikidata:SPARQL_query_service)
* [SPARQL wikibook](https://en.wikipedia.org/wiki/wikibooks:SPARQL)

[Category](https://www.wikidata.org/wiki/Special:Categories):

* [SPARQL](https://www.wikidata.org/wiki/Category:SPARQL)

Navigation menu

* [English](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial)
* Not logged in
* [Talk](https://www.wikidata.org/wiki/Special:MyTalk)
* [Contributions](https://www.wikidata.org/wiki/Special:MyContributions)
* [Create account](https://www.wikidata.org/w/index.php?title=Special:CreateAccount&returnto=Wikidata%3ASPARQL+tutorial)
* [Log in](https://www.wikidata.org/w/index.php?title=Special:UserLogin&returnto=Wikidata%3ASPARQL+tutorial)
* [Project page](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial)
* [Discussion](https://www.wikidata.org/wiki/Wikidata_talk:SPARQL_tutorial)
* [Read](https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial)
* [Edit](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=edit)
* [View history](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=history)

Top of Form



Bottom of Form

* [Main page](https://www.wikidata.org/wiki/Wikidata:Main_Page)
* [Community portal](https://www.wikidata.org/wiki/Special:MyLanguage/Project:Community_portal)
* [Project chat](https://www.wikidata.org/wiki/Wikidata:Project_chat)
* [Create a new Item](https://www.wikidata.org/wiki/Special:NewItem)
* [Recent changes](https://www.wikidata.org/wiki/Special:RecentChanges)
* [Random Item](https://www.wikidata.org/wiki/Special:Random)
* [Query Service](https://query.wikidata.org/)
* [Nearby](https://www.wikidata.org/wiki/Special:Nearby)
* [Help](https://www.wikidata.org/wiki/Special:MyLanguage/Help:Contents)
* [Donate](https://donate.wikimedia.org/wiki/Special:FundraiserRedirector?utm_source=donate&utm_medium=sidebar&utm_campaign=C13_wikidata.org&uselang=en)

Lexicographical data

* [Create a new Lexeme](https://www.wikidata.org/wiki/Special:NewLexeme)
* [Recent changes](https://www.wikidata.org/wiki/Special:RecentChanges?namespace=146%3B147)
* [Random Lexeme](https://www.wikidata.org/wiki/Special:Random/Lexeme)

Tools

* [What links here](https://www.wikidata.org/wiki/Special:WhatLinksHere/Wikidata:SPARQL_tutorial)
* [Related changes](https://www.wikidata.org/wiki/Special:RecentChangesLinked/Wikidata:SPARQL_tutorial)
* [Special pages](https://www.wikidata.org/wiki/Special:SpecialPages)
* [Permanent link](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&oldid=1785728898)
* [Page information](https://www.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&action=info)

In Wikipedia

[Add links](https://www.wikidata.org/wiki/Special:NewItem?site=wikidatawiki&page=Wikidata%3ASPARQL+tutorial)

* This page was last edited on 6 December 2022, at 21:46.
* All structured data from the main, Property, Lexeme, and EntitySchema namespaces is available under the [Creative Commons CC0 License](https://creativecommons.org/publicdomain/zero/1.0/); text in the other namespaces is available under the [Creative Commons Attribution-ShareAlike License](https://creativecommons.org/licenses/by-sa/3.0/); additional terms may apply. By using this site, you agree to the [Terms of Use](https://foundation.wikimedia.org/wiki/Terms_of_Use) and [Privacy Policy.](https://foundation.wikimedia.org/wiki/Privacy_policy)
* [Privacy policy](https://foundation.wikimedia.org/wiki/Privacy_policy)
* [About Wikidata](https://www.wikidata.org/wiki/Special:MyLanguage/Project:Introduction)
* [Disclaimers](https://www.wikidata.org/wiki/Special:MyLanguage/Project:General_disclaimer)
* [Mobile view](https://m.wikidata.org/w/index.php?title=Wikidata:SPARQL_tutorial&mobileaction=toggle_view_mobile)
* [Data access](https://www.wikidata.org/wiki/Special:MyLanguage/Wikidata:Data_access)
* [Developers](https://developer.wikimedia.org/)
* [Statistics](https://stats.wikimedia.org/#/www.wikidata.org)
* [Cookie statement](https://foundation.wikimedia.org/wiki/Cookie_statement)
* [Wikimedia Foundation](https://wikimediafoundation.org/)
* [Powered by MediaWiki](https://www.mediawiki.org/)