EXPLORATORY QUERIES

When faced with a new and unfamiliar dataset, it is helpful to start by getting a sense of the classes and properties being used to describe the data. Without this knowledge, writing queries is virtually impossible. The following "exploratory" queries are *simple*, *reusable*, and can quickly give you an idea *what a dataset is all about*.

Writing a "Get Classes" query

A query which asks for all the classes used in the dataset **(Figure 3)** is a great example of basic SPARQL syntax. You can write the query in any text editor, including a simple one like Notepad.

It starts out with the SELECT statement, in which we tell the query what we want it to return- a single variable, *"class"*.

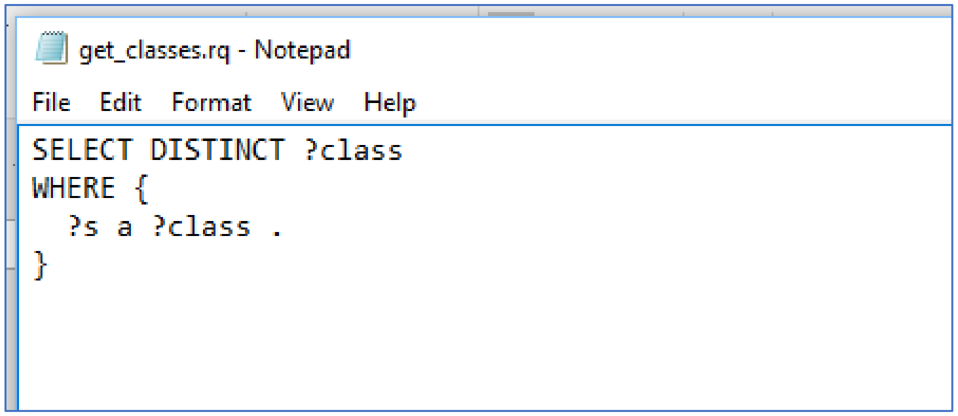
The keyword DISTINCT is added to make sure that each time a new class is found, you only get it listed in the results once. Otherwise, the results would include an entry for each occurrence of the class in the dataset- in this case, that could be millions.

In the next part of the query, the WHERE statement, you set additional conditions for which triples you want to return. The WHERE statement can contain one triple statement, or many. In this case, there is only one. The **subject** of the triple is a generic placeholder variable, *"?s"*.

The **predicate** is a special variable, *"a",* which is shorthand for *"rdf:type"*. This shorthand is often used because *"rdf:type"*, the predicate for declaring a class, occurs very frequently in queries.

The **object** of the triple is another placeholder variable. You can name it anything, including the commonly used *"?o",* but because you are searching for classes, *"?class"* is a more meaningful choice.

When you are done with the query, save it, giving it any name you want (e.g., "*get_classes*") and using the extension *".rq"*, which is standard for SPARQL queries.



```
SELECT DISTINCT ?class
WHERE {
  ?s a ?class .
}
```

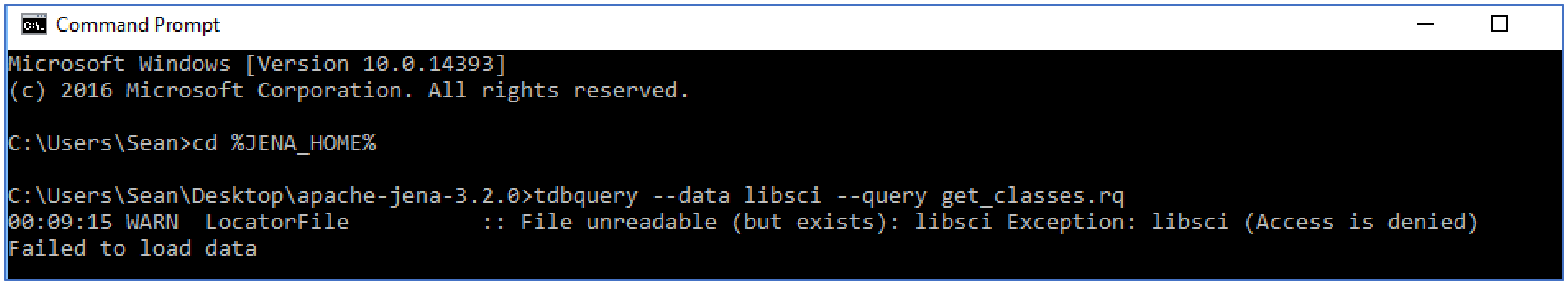
Figure 3- Simple SPARQL query saved as .rq file

To run this query, open a command prompt and make sure you are in the directory where you saved Apache Jena TDB and loaded the dataset. Then, simply type the following command:

```
tdbquery --loc libsci --query get_classes.rq
```

A few things to note about this command: The script starts with *"tdbquery"*, which tells Jena which utility to use for the task at hand- querying the data. The *"loc"* parameter points to the folder where you loaded are dataset, *"libsci"*. Finally, the *"query"* parameter always points to a file where you have saved a SPARQL query.

NOTE: Apache Jena ARQ users will find this process familiar. However, take care to use the *"loc"* parameter when using *"tdbquery"* rather than the *"data"* parameter used with the *"sparql"* utility in ARQ. Mixing these two parameters up will result in an error (**Figure 4**):



```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Sean>cd %JENA_HOME%

C:\Users\Sean\Desktop\apache-jena-3.2.0>tdbquery --data libsci --query get_classes.rq
00:09:15 WARN  LocatorFile          :: File unreadable (but exists): libsci Exception: libsci (Access is denied)
Failed to load data
```

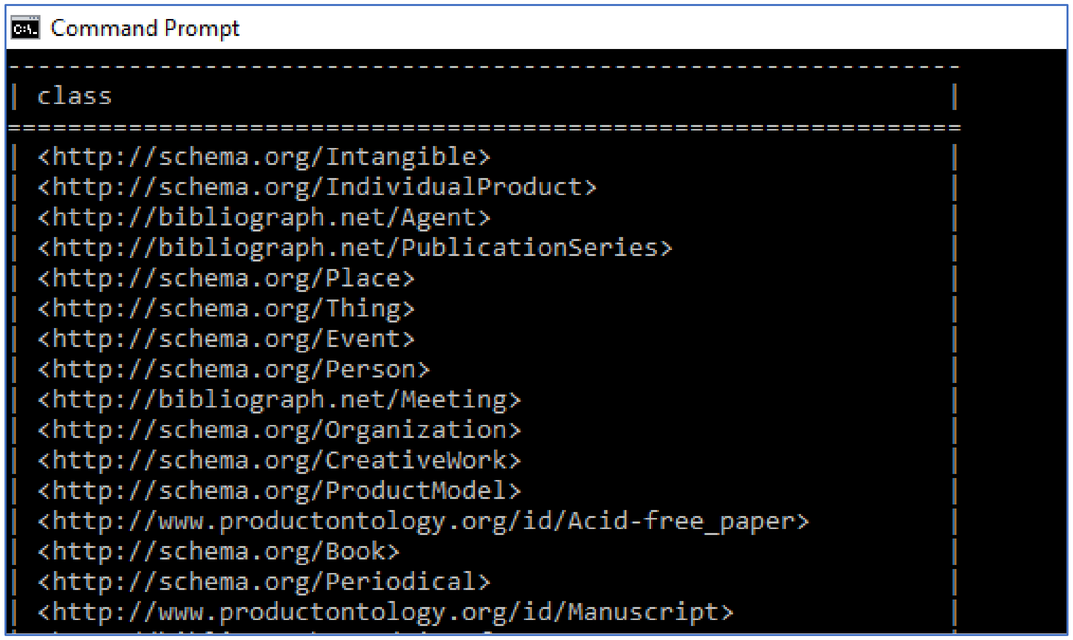
Figure 4- Error message when querying dataset loaded in TDB

Alternatively, you could have typed the query directly into the command line. However, this could get messy when writing long and complex queries. Plus, you wouldn't have a record of your query in case you wanted to use it again (on this or any other dataset). Many SPARQL queries only need to be modified slightly to be used on new datasets.

Running this query (and many others in this tutorial) will likely take at least 90 seconds, and possibly a bit longer. Remember, the dataset is massive- over 25 million triples! If you want to know exactly how long each query takes, you can add the *"time"* parameter to your command line, like so:

```
tdbquery --loc libsci --time --query get_classes.rq
```

When the query finishes, you should see a table that looks something like the one below **(Figure 5)**.


Command Prompt

```
----------------------------------------------------------------
| class                                                        |
================================================================
| <http://schema.org/Intangible>                               |
| <http://schema.org/IndividualProduct>                        |
| <http://bibliograph.net/Agent>                               |
| <http://bibliograph.net/PublicationSeries>                   |
| <http://schema.org/Place>                                    |
| <http://schema.org/Thing>                                    |
| <http://schema.org/Event>                                    |
| <http://schema.org/Person>                                   |
| <http://bibliograph.net/Meeting>                             |
| <http://schema.org/Organization>                             |
| <http://schema.org/CreativeWork>                             |
| <http://schema.org/ProductModel>                             |
| <http://www.productontology.org/id/Acid-free_paper>          |
| <http://schema.org/Book>                                     |
| <http://schema.org/Periodical>                               |
| <http://www.productontology.org/id/Manuscript>               |
```

Figure 5- Results table (truncated)

Obviously, this table is temporary and will disappear as soon as the command prompt is closed. To save the results to a *.csv file* instead, you just need to add a few parameters to the command line script used to run the query:

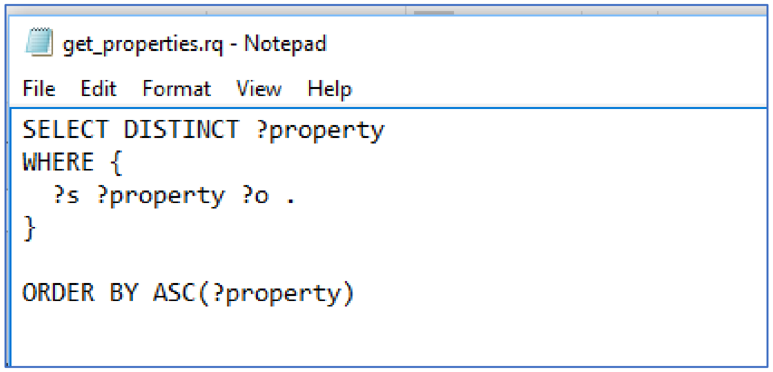
```
tdbquery --loc libsci --time --results CSV --query get_classes.rq > classes.csv
```

As you can see, this example added the *"results"* parameter with the value *"CSV"*. Also, the *">"* operator is used to **redirect** the results of the query to the file specified (e.g., *"classes.csv"*). *Now the results are saved somewhere you can refer back to when writing more complex queries.*

Writing a "Get Properties" query

Knowing all the classes used in the dataset is a good start, because they will be used as **objects** in some of the triple patterns that form future queries. Equally important are the **properties** that will be used as the **predicates** to link resources both to literal values and to other resources- the whole point of **Linked Data**.

Retrieving a list of all the properties used in the dataset can be accomplished with another simple query **(Figure 6)**. This query follows the same logic and syntax as the previous query. It asks for any *"?property"* connecting any **subject** *(?s)* and any **object** *(?o)*. Placeholder variables are used and DISTINCT is employed to avoid repetitious results – without DISTINCT, this query would return the property for *every triple in the dataset*. The only difference is the addition of the ORDER BY keyword to receive the results in alphabetical order *according to the property variable* (in this case, this happens to be the only variable asked to see results for anyway). Also, the *"ASC"* function has been applied so that the results start from A, rather than Z.



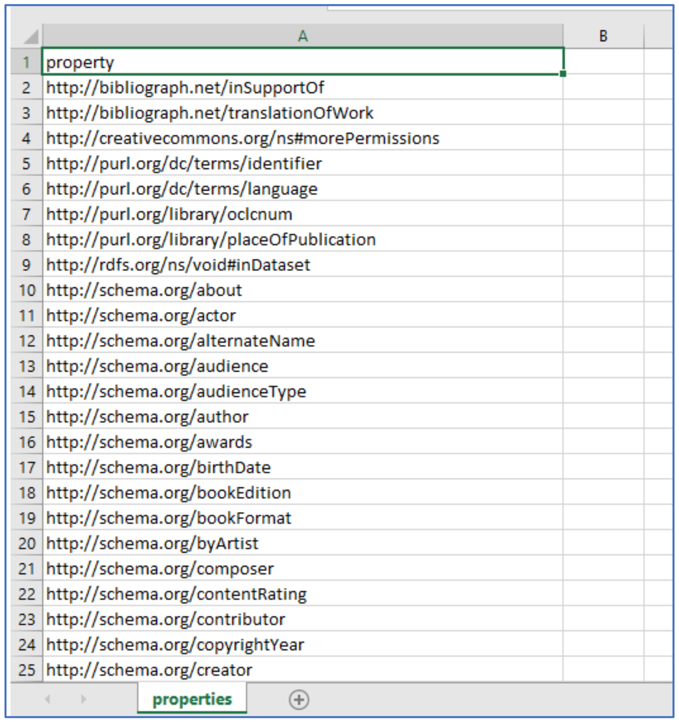
```
SELECT DISTINCT ?property
WHERE {
  ?s ?property ?o .
}

ORDER BY ASC(?property)
```

Figure 6- SPARQL query for retrieving all properties in dataset

Just like the previous query, this query is executed from the command line, changing only the value for the *query file* and the *output file* where you would like the results redirected:

```
tdbquery --loc libsci --time --results CSV --query get_properties.rq > properties.csv
```

As you will see in the following exercises, saving the lists of classes and properties to refer back to later when writing more specific queries can be very convenient (**Figure 7**).



	A	B
1	property	
2	http://bibliograph.net/inSupportOf	
3	http://bibliograph.net/translationOfWork	
4	http://creativecommons.org/ns#morePermissions	
5	http://purl.org/dc/terms/identifier	
6	http://purl.org/dc/terms/language	
7	http://purl.org/library/oclcnum	
8	http://purl.org/library/placeOfPublication	
9	http://rdfs.org/ns/void#inDataset	
10	http://schema.org/about	
11	http://schema.org/actor	
12	http://schema.org/alternateName	
13	http://schema.org/audience	
14	http://schema.org/audienceType	
15	http://schema.org/author	
16	http://schema.org/awards	
17	http://schema.org/birthDate	
18	http://schema.org/bookEdition	
19	http://schema.org/bookFormat	
20	http://schema.org/byArtist	
21	http://schema.org/composer	
22	http://schema.org/contentRating	
23	http://schema.org/contributor	
24	http://schema.org/copyrightYear	
25	http://schema.org/creator	

Figure 7: Properties saved as table in CSV file