Let's begin by exploring the data.

By extracting the **camera_specs.tar.gz**, we can see the following structure:

```
buy.net/

      4233.json
      4236.json
      ...
      ...
      ...
      6785.json

cammarkt.com/

...
...
...

www.wexphotographic.com/
```

We will write a JSON loader (UDF) that creates a table entry for each **.json** file in the folder-structure. For example, the **buy.net/4233.json** will correspond to a tabular entry with ID: **buy.net//4233**.

A point-of-interest here is that some of the JSON files contain an array of legitimate values for a particular key (e.g. *./www.camerafarm.com.au/705.json*). Although JSON-compliant as a format, our UDF function cannot emit an array-like value to the database. In order to load the entirety of the available information in the database, we choose to concatenate the elements of any array-like value into a single unified string. This seems reasonable at the time being, but we may reconsider down the road.

We will then utilize the CREATE TABLE FROM LOADER MonetDB command and we will populate a table called **specs** with all the JSON data.

Now, on with the **sigmod_medium_labelled_dataset.csv** data. This is our annotated data, in a CSV format. We will utilize the COPY INTO FROM MonetDB command to efficiently bulk-insert the CSV data into a new table called **spec_matchings**.

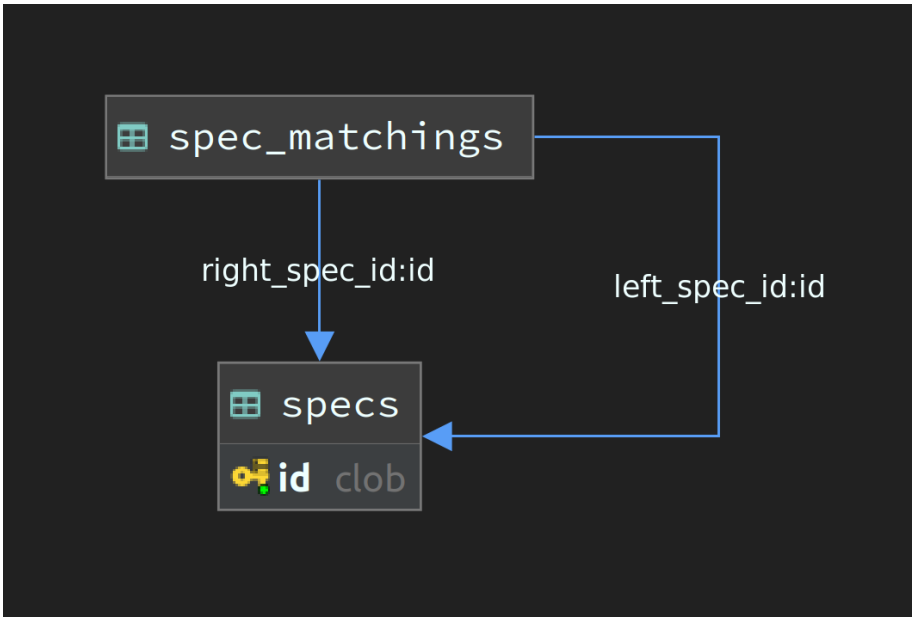
Having transferred our dataset to a MonetDB instance, we will additionally perform some ALTERations to our two tables in order to come up with a working, first schema:

a) enforce a primary key constraint on **specs.id** column.

b) enforce a foreign key constraint to the **specs_matchings.left_spec_id** (referencing **specs.id**)

c) enforce a foreign key constraint to the **specs_matchings.right_spec_id** (referencing **specs.id**)

The resulting schema can be seen in the following UML diagram:



Now that we have a working database schema, we can proceed with some data exploration to build an understanging towards the entity resolution task.

We have **29787** products (i.e. their specs), and an astonishing **4662** attributes for each product. This unusually high number of columns stems from the fact that the JSON files were essentially unstructured information, with little overlap, even between products of the same retailer.

We are also equipped with **46665** product pairs (out of the pool of all possible product pairs that can be formulated from our dataset), for which we know the ground truth i.e. if they are the same or not.

Table 1: select label, count(*) as count from spec_matchings group by label;

label	count
1	3582
0	43083

As we've expected - given the fact that the label corresponds to a product pair - the problem is imbalanced. We have, comparatively, very few products that are "matching".

We can also exploit the ANALYZE command along with the **sys.statistics** table and derive the support (i.e. the ratio of non-null values) for all columns of **specs**. Below are the 10 columns with the highest support (the ID is excluded):

Table 2: top 10 specs columns, based on support (rounded to 2 decimal digits)

column	support
<page title>	1
brand	0.53
model	0.5
megapixels	0.46
type	0.46
screen size	0.41
optical zoom	0.39
mpn	0.35
condition	0.33
upc	0.26

By taking a closer look at these 10 columns (aka product features) we can suspect that they contain significant discriminative ability and will most probably be of vital importance to our machine learning component.
It is also highly informative to get the number of columns with support below a given threshold:

Table 3: no. of columns with support below x%

# columns	**support threshold**
4638	10%
4600	5%
4502	2%
4383	1%
3527	0.1%
62	0.01%

This is in line with our initial impression of the data being very much unstructured: for example, 4383 out of the total 4662 columns have less than 1% support!