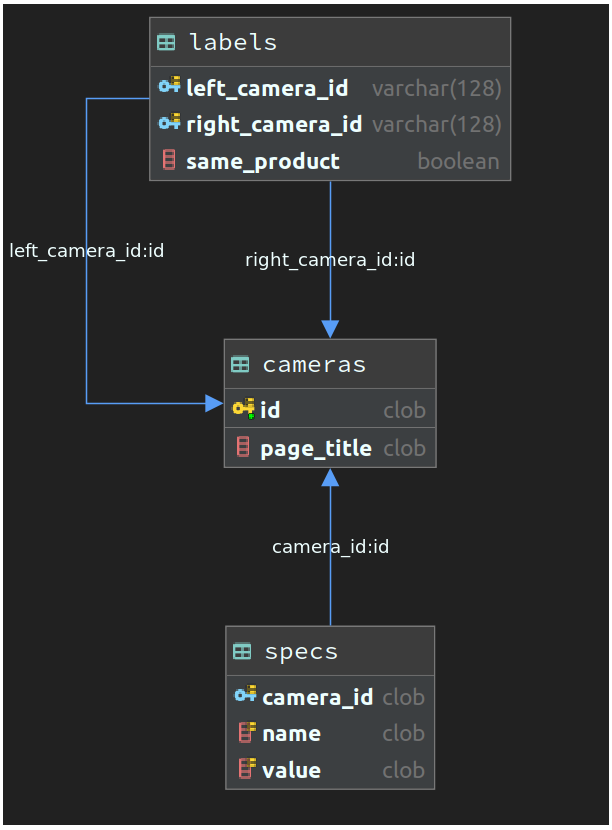
Let's begin by exploring the data.

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/
```

Given that each camera entity, apart from the page title information, can have any number of features (specs) available and that feature-overlap and consistency between any two products is far from the norm, we will utilize the following schema:



We will write 2 pythonic loaders (UDFs) that will create and populate the tables **cameras** (holding the primary identifier and the page title) and **specs** (holding a reference to the camera, the different camera specs with their corresponding values.

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*). In that case, our specs loader function will deduplicate the array of values before attempting to generate all the specs-values mappings.

For these two tables, we will utilize the MonetDB command CREATE TABLE FROM LOADER.

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 **labels**.

Having transferred our dataset to a MonetDB instance, we will additionally perform some ALTERations to our tables in order to add ensure referential integrity and thus come up with a working chema:

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

b) enforce a foreign key constraint to the **specs.camera_id** (referencing **cameras.id**)

c) enforce a unique constraint on **specs** table, for the columns (**camera_id**, **name**, **value**)

d) enforce a foreign key constraint to the **labels.left_camera_id** (referencing **cameras.id**)

e) enforce a foreign key constraint to the **labels.right_camera_id** (referencing **cameras.id**)

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. cameras), and an astonishing **4661**(4660 + the page title) different specs across these products. This unusually high number of specs 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 same_product, count(*) as count from labels group by same_product;

same_product?	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".

It is illuminating to calculate the support for each spec (i.e. the ratio of non-null values for this spec). We will include <page title> in the analysis, as the most prominent camera spec, even though we chose to include it in the cameras table and not in the specs table:

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

spec	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 specs 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 specs with support below a given threshold:

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

# specs	**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 specs have less than 1% support!