Graphs and Types

There are two basic ways of having a schema or type system for a graph database.

- define a schema system as part of the tool.
- define the schema in the graphs.

This doc is about the second method.

With that method, the graphs are still just graphs --- there's no external schema to help interpret them --- there are extra triples that describe the schema, and possibly some tools that treat these bits of the schema specially.

RDFS (RDF Schema) and similar

This is the approach taken by RDFS, Freebase and schema.org. (The latter two use versions of the former)

Schema in RDFS

RDFS allows triples like

```
A rdf:type rdfs:class
```

For `A` is a type in our schema.

```
B rdfs:subClassOf A
C rdfs:subClassOf A
```

For `B` and `C` are subclasses-of/specialisations-of/inherit-from `A`

Similarly with edges/properties, one can write

```
p rdf:type rdfs:Property
```

For `p` is a edge/property

```
p rdfs:range A
p rdfs:domain B
```

For the domain of `p` is always an instance of `A`, and similarly for range.

```
q rdfs:subPropertyOf p
```

For all triples `a q b`, must also be in `p`; i.e. `a p b`.

(Note it's common then to provide human readable labels for the classes such as `A rdfs:label "A"`)

Data in RDFS

So that's the schema, and then one can write

```
a rdf:type A
```

For `a` is an instance of the `rdfs:class` called `A`

or `b rdf:type B`

For `b` is an instance of `B`, but by the sub class inference must also be an instance of `A`.

The regular data just stays the same. If we've specified an edge `p`, then we still just have a triple `a p b` for `a` has edge `p` to node `b`. So all the regular data stays the same, it's the meta data that's RDFS.

What's it mean in Dgraph

Using RDFS doesn't have to change lots of the mechanics of Dgraph (though for inference it does). Any graph that didn't want this sort of schema wouldn't have to have it (maybe, but if Dgraph went this way, then why not always store something akin to the current schema information in rdfs triples).

For graphs that do have RDFS, schema queries could be just regular queries in GraphQL+-. Maybe something like

```
b_superclasses( ... (rdfs:label "B")) {
  rdfs:subClassOf {
    rdfs:label
  }
}
```

Of course this could still be in a schema query language that is rewritten to queries against the schema.

It should also give the ability to query based on type - for example give me all the people at location l, Vs give me all the cars at location l, etc.

BUT Things get harder once the implied type inference gets thrown in (from memory SPARQL doesn't require any RDFS reasoning at query time, and from what I know, RDF stores support bits of RDFS but not all on the fly at query time - I'd have to look more deeply at the state of the art for that one. Stardog has the most complete reasoning system, but they are aiming at OWL, which is a step up from RDFS, so a more

complete reasoning system in exchange for smaller graphs).

Some parts of the type inference can be accounted for because the schema would be a small graph and schema changes are infrequent, so should be able to compute all the type inferences on the schema and store them. The question is, given all the type inferences done on the schema, can we do the inference on the data on the fly --- AlegroGraph does something like this.

The inferences on the data can also be precomputed, but that blows out the graph and it's an issue when facts are modified or deleted because then you have to know what's inferred from each triple etc.

Another option is support RDFS stye schema but with out `rdfs:subClassOf` or `rdfs:subPropertyOf`, or allow them, but do no reasoning.

Triples, lots of Triples

In general RDFS adds lots of triples to a graph store. The schema itself is small, but every node (typically) gets a type and so an edge of `a rdfs:type A` is added for every node of known type that is added to the graph. Then there might be a big table for edge `rdfs:type`. Plus other bookkeeping.

However, because Dgraph has an edge centric store, I think this could be stored implicitly. For example, if we knew

```
A rdf:type rdfs:class
B rdfs:subClassOf A
C rdfs:subClassOf A

p rdf:type rdfs:property
```

```
p rdfs:range A
```

If the `p` edges were all stored together, then queries for things of type `C` that have a `p` edge requires looking up the type table to determine the things in the `p` posting list that are of type `C`, rather than types `A` or `B`.

But if the `p` data is sharded by type then the query just looks in the `p–C` shard. That way would mean not explicitly storing the type data and saving the search at query time.

Back the other way works fine too, cause a query for all the `A` data, just looks in all the shards. This might help too with the reasoning if inference was required.