Overview

The project is split across 3 repositories

- obdabenchmark https://git.soton.ac.uk/gk1e17/obdabenchmark
- chasestepper https://git.soton.ac.uk/jde1g16/chasestepper
- obda-converter https://github.com/JamesErrington/obda-converter

This structure is very fluid and can be easily changed if needed - probably will need to be as the 'chasestepper' name was only meant to be a placeholder, and having stuff across 2 different git systems on 3 accounts probably isn't the best

obda-converter is a JavaScript project, written using TypeScript and bundled with NPM. obdabenchmark also contains a TypeScript segment, but is mostly bash scripts, data integration tools and scenario data. chasestepper is a Java project, bundled with Gradle - this could be altered to use maven or some other system if needed but is probably not recommended? It also contains the data generator files

ChaseStepper

To build

- `gradle chasestepperJar`
- `gradle dataJar`
- Can also use `gradle clean` to clear previous builds

Obdabenchmark

To download dependencies - `yarn` - this must be done on first downloading the code, or any time the 'node_modules' folder is deleted

To build

- `yarn build`
- Use `sudo npm link` to update the local binary command

Obda-converter

To download dependencies

- `yarn`

this must be done on first downloading the code, or any time the 'node_modules' folder is deleted

To build

- `yarn build`
- Use `sudo npm link` to update the local binary command

The `obdabenchmark llunatic <folder>` command is used to generate the llunatic xml files for a scenario

Scenario folder structure

```
name/
|-- data/               := CSV data files
| |-- a.csv
|-- dependencies/
| |-- ontology.owl      := OWL ontology file
| |-- st-tgds.txt       := ChaseBench rules
| |-- t-tgds.txt        := ChaseBench rules
|-- out/                := LLunatic output folders
| |-- size/
|   |-- Qx/
|     |-- Qx.csv
|-- queries
| |-- graal             := SPARQL queries
|   |-- Qx.rq
| |-- iqaros            := IQAROS queries
|   |-- Qx.txt
| |-- RDFox             := ChaseBench queries
|   |-- Qx/
|     |-- Qx.txt
|     |-- size.xml      := LLunatic XML
|     |-- size-bca.xml  := LLunatic XML for BCA
|-- schema/
| |-- s-schema.sql      := PostgreSQL sql
| |-- s-schema.txt      := ChaseBench schema
| |-- t-schema.sql      := PostgreSQL sql
| |-- t-schema.txt      := ChaseBench schema
|-- tests
| |-- size/
|   |-- Qx/
|     |-- tool.csv
|   |-- database.csv
|-- config.ini          := INI database config
```

The majority of this structure can be bootstrapped using the provided scripts. For a DL-Lite scenario, the minimum set up to be bootstrapped is the following:

```
name/
|-- data/
| |-- a.csv
|-- dependencies/
| |-- ontology.owl
|-- queries
| |-- Qx.rq
|-- config.ini
```

For a ChaseBench scenario:

```
name/
|-- data/
| |-- a.csv
|-- dependencies/
| |-- st-tgds.txt
| |-- t-tgds.txt
|-- queries
| |-- Qx.txt
|-- schema/
| |-- s-schema.txt
| |-- t-schema.txt
|-- config.ini
```

Scripts

Bootstrap DL Lite scenarios with `./scripts/bootstrap.sh <folder> dllite [data]` - add data if you also want to generate data Bootstrap ChaseBench scenarios with `./scripts/bootstrap.sh <folder> chasebench` The setup.sh script is used to automate the bootstrapping of multiple scenarios - edit it as you need. Within the bootstrap.sh script, you will need to change the SIZES list to match which data sizes you wish to generate.

The generate.sh script automates data generation - use `./scripts/generate.sh <folder> <size>` where 'folder' is the top level scenario name and 'size' is a data size defined in the config.ini in tools/datagenerator

build.sh drops a database if exists, then creates and imports - use `./scripts/build.sh <folder> <size>` where 'folder' is the top level scenario name and 'size' is a data size defined in the data folder of that scenario. The database information is taken from the scenario config.ini file

query.sh is the main heavy lifter of the scripts, and runs a test of a query on each tool, 6 times. It is invoked using `./scripts/query.sh <folder> <query number> <size>`, where 'folder' is the top level scenario name, query number is self explanatory and 'size' is a data size defined in the data folder of that scenario

Most of these commands are automated using the run.sh script. At the moment, the parameters of the test have to be changed in the script itself, but this could be changed to take command line arguments

Getting it to run

When first using the codebase, you need to download the Javascript dependencies and build the scripts. To do this:

```
cd obda-benchmark
yarn && yarn build && sudo npm link
```

Then, make sure you have the scenario folders set up as you need, following the layout defined above. You can use the setup.sh or bootstrap.sh scripts to automate the building of much of them.

Then, you need to edit the query.sh and run.sh scripts to define which tools you wish to run, over which scenarios and data sizes.

When the code has finished, the results are printed to set of CSV files found in the test/ folder of the scenarios.

Tools

Rapid

Rapid is taken from the link George sent to me, and I believe was slightly edited to not print as much superficial output, but nothing else was changed.

In this situation, it is invoked with the following command:

```
java -jar Rapid2.jar DU SHORT <OWL ontology file> <rule query file>
```

In the Rule Query file, numbers are used as variable names

IQAROS

IQAROS is taken from the link George sent to me, but this was found to be broken since it has been partially updated with names changed in one place but not the other. There is an unofficial github https://github.com/INL/iqaros that contains a working version, which on inspection has just removed the broken code, which was a fix I found to work on the original code.

To invoke, use the following command:

```
java -jar iqaros.jar <OWL ontology file> <rule query file>
```

In the Rule Query file, numbers are used as variable names

Graal

Rewriter

Graal uses only the query rewriting API from http://graphik-team.github.io/graal/doc/index as the full end-to-end could not be made to work. The examples were adapted to parse OWL ontologies and SPARQL queries and packaged into a Spring Jar as it was the only way found to get the dependencies to be included properly.

It is invoked with the following command:

```
java -jar obda-benchmark-graal-1.0-SNAPSHOT-spring-boot.jar <OWL ontology
file> <SPARQL query file>
```

Other progress

In terms of progress towards the end-to-end system, it was found that Graal does not properly support Postgres in the fact that it cannot handle upper case letters in names. In emails to the creators, the following responses are highlighted:

> the main problem comes from the case sensitiveness detection. However PostgreSQL is case sensitive, JDBC says me not (through DatabaseMetaData.supportsMixedCaseIdentifiers())) so wrong processing is done over predicate and table names.

> but it's not an option in Graal. I think we misunderstood the semantic of the "supportsMixedCaseIdentifiers" method, so it's surely a bug.

> I just wanted to inform you that we have planned a more in-depth analysis of this problem during the week of May 20. We will keep you informed.

If this bug is fixed, I do not think it would be much work to get Graal working, since all the other infrastructure (ontologies, queries, databases) is already set up.

RDFox

The RDFox jar is taken from the ChaseBench, and is invoked with the following command:

```
java -jar chaseRDFox-linux.jar -threads <number> -chase [standard | skolem]
-s-sch <source schema> -t-sch <target schema> -st-tgds <source to target
tgds> -src <data folder> -t-tgds <target tgds> -qdir <query folder>
```

I have added a run.sh script that simplifies running the code based on the folder structure defined above:

```
./run.sh <base scenario folder> <data size> <query number>
```

Queries have to use letters for variables

BCA

The base BCA code that generates the new st-tgds is invoked with the following command:

```
java -jar chasestepper-1.0.jar <source to target tgds> <target tgds> <rule
query file>
```

This is usually used in conjunction with RDFox to do the chasing, so I wrote a script to combine to two:

```
./run.sh <source schema> <target schema> <source to target tgds> <data
folder> <target tgds> <query folder> <query number>
```

ChaseFun

The ChaseFun code is from George's dropbox, and includes a jar, a start script, and some sample properties files. I have not got ChaseFun to work properly, but I have made some progress and can provide some help.

Firstly, the properties file. Although there is both a 'userdb' and 'pwddb' field, the code internally only reads the 'userdb' property and assigns the it to the username and password - the result of this being your Postgres role must have the same username and password in order for ChaseFun to run.

From decompiling the jar, I can provide the following API

```
userdb : username for rdmbs
pwddb : COMPLETELY USELESS
dbUrlDriverSource : jdbc url of source database
dbUrlDriverTarget : jdbc url of target database
SourceSchema : name of source database (default : source)
TargetSchema : name of target database (default : target)
scenarioPath : base folder of scenario files
dataPath : data folder
solutionPath : folder to output solution to
recreateDb : remakes database (i think) (default : true)
doMaterialize : run chase (default : true)
threads : number of threads to use (default : all)
scenarioType : type of abstraction layer to use (default : pods)
```

scenarioType : pods (or leaving it blank) is what is needed here - otherwise it uses XML

At the moment, I can parse the scenario and import the data, but it then throws an error connecting (somehow?)

```
total tuples source: 5250000
Start: chase and export solution
Connection to the DB is impossible!! due to:java.sql.SQLException: Error
preloading the connection pool
Try to restore the pool!!
Connection to the DB is definitly impossible!! due
to:java.sql.SQLException: Error preloading the connection pool
```

I have tried using only lower case in the database names, no luck. Googling this error gives suggestions of setting the pool number to 0, but that doesn't seem to be an option we have control over.

KNOWN ISSUE

In ChaseFun, you can only have one file named 's-schema' or 't-schema', regardless of file extension. This means you will have to rename the SQL files, or add a ~ to the start of the file name.

Ontop

Ah, Ontop - this has been a thorn in my side for months. I have been using the CLI version, since this we need a way of invoking it automatically and the standard Ontop program is a plugin for Protege which is a GUI program. Download from here https://sourceforge.net/projects/ontop4obda/files/ and documentation here https://ontop.inf.unibz.it/documentation/.

There is also a seperate download for the tree-witness rewriter that is used internally http://www.dcs.bbk.ac.uk/~roman/tw-rewriting/ and that can be used as such:

```
java -jar tw-rewriting.jar <OWL ontology file> <SPARQL query file>
```

However, this rewriter produces Datalog which is not currently supported since I could not find a parser / converter and did not have time to write one. With a Datalog -> SQL converter, it would be trivial to add the Tree Witness Rewriter akin to Rapid or IQAROS.

For the end-to-end, the download comes with a script with various commands - one of these is the bootstrap command, which creates the mapping files needed for Ontop:

```
./ontop bootstrap -m <mapping file name> -l <jdbc url> -p <database
password> -u <database username> -d <jdbc driver name> -t <ontology file
name> -b <base uri>
```

The other command is to query

```
./ontop query -d <jdbc driver name> -l <jdbc url> -m <mapping file> -o
<output folder> -p <database password> -q <SPARQL query file> -t <ontology
file> -u <database username>
```

The errors seem to be in the mapping file - from what I can ascertain by inspection, they seem to be mislabelling the relation names with the column names, which makes the ontology invalid - but this is just a guess. I've also tried the lower case Postgres fix, no help.

Llunatic

While Llunatic I think has been discounted from this due to it only using one thread and therefore being very slow compared to RDFox, I can still provide some pointers for its use.

Download from https://github.com/donatellosantoro/Llunatic as this version definitely includes the fix that I had to submit to them. Build it with the ant command they define

```
ant gfp
```

and then use the runExp.sh script in the lunaticEngine folder. I have written a Javascript command that autogenerates the XML files needed to define a scenario in Llunatic:

```
obdabenchmark llunatic <folder>
```

If you provide a valid scenario folder of the structure defined above, this command should produce an XML file for each query for each data size you have data for.

I think there were still issues with its use for some of the scenarios, but we definitely got it to work for LUBM.