CHAIN DOCUMENTATION

TABLE OF CONTENTS

OBJECTIVES OF CHAIN 7

SETTING UP CHAIN 8

RUNNING FILES IN CHAIN 9

FILE STRUCTURE 10

TESTING RESULT DOCUMENTS 11

- BREAKDOWN OF TESTING DOCUMENTS 11

MATCH_STRUC.JAVA 13

- BASIC FUNCTIONALITY 13
- DESIGN 13
- PRIVATE VARIABLES/FIELDS 13
- METHODS 15
- ASSOCIATED TESTING DOCUMENTS 17
- RUNNING THIS FILE 17

CALL_SPSM.JAVA 18

- BASIC FUNCTIONALITY 18
- DESIGN 18
- PRIVATE VARIABLES/FIELDS 18
- METHODS 19
- ASSOCIATED TESTING DOCUMENTS 20
- RUNNING THIS FILE 20

BEST_MATCH_RESULTS.JAVA 21

- BASIC FUNCTIONALITY 21
- DESIGN 21
- PRIVATE VARIABLES/FIELDS 22
- METHODS 22
- ASSOCIATED TESTING DOCUMENTS 22
- RUNNING THIS FILE 23

REPAIR_SCHEMA.JAVA 24

- BASIC FUNCTIONALITY 24
- DESIGN 24
- PRIVATE VARIABLES/FIELDS 24
- METHODS 25
- ASSOCIATED TESTING DOCUMENTS 25
- RUNNING THIS FILE 25

NODE.JAVA 26

- BASIC FUNCTIONALITY 26
- DESIGN 26

PRIVATE VARIABLES/FIELDS 26
METHODS 26
ASSOCIATED TESTING DOCUMENTS 28
RUNNING THIS FILE 28

ONTOLOGY_STRUC.JAVA 29
BASIC FUNCTIONALITY 29
DESIGN 29
PRIVATE VARIABLES/FIELDS 29
METHODS 29
ASSOCIATED TESTING DOCUMENTS 30
RUNNING THIS FILE 30

CREATE_QUERY.JAVA 31
BASIC FUNCTIONALITY 31
DESIGN 31
PRIVATE VARIABLES/FIELDS 31
METHODS 32
ASSOCIATED TESTING DOCUMENTS 33
RUNNING THIS FILE 33

RUN_QUERY.JAVA 34
BASIC FUNCTIONALITY 34
DESIGN 34
PRIVATE VARIABLES/FIELDS 34
METHODS 35
ASSOCIATED TESTING DOCUMENTS 35
RUNNING THIS FILE 35

SCHEMA_FROM_QUERY.JAVA 36
BASIC FUNCTIONALITY 36
DESIGN 36
PRIVATE VARIABLES/FIELDS 36
METHODS 37
ASSOCIATED TESTING DOCUMENTS 37
RUNNING THIS FILE 38

RUN_CHAIN.JAVA 39
BASIC FUNCTIONALITY 39
DESIGN 39
PRIVATE VARIABLES/FIELDS 40
METHODS 41
ASSOCIATED TESTING DOCUMENTS 41
RUNNING THIS FILE 42

SPSM_TEST_CASES.JAVA 44
BASIC FUNCTIONALITY 44
EXPECTED RESULTS 44

TEST 1.1.1 44
TEST 1.2.1 44
TEST 1.2.2 44
TEST 1.2.3 45
TEST 1.2.4 45
TEST 1.3.1 45
TEST 1.3.2 45
TEST 1.3.3 45
TEST 1.4.1 45
TEST 1.4.2 46
TEST 1.4.3 46
TEST 1.4.4 46
TEST 1.4.5 46
TEST 1.5.1 46
TEST 1.5.2 46
TEST 1.5.3 46
TEST 1.5.4 46
TEST 1.5.5 47
TEST 1.5.6 47
TEST 1.5.7 47
TEST 1.5.8 47
TEST 1.6.1 47
TEST 1.6.2 47
TEST 1.6.3 47
TEST 1.6.4 47
TEST 1.6.5 48
TEST 1.6.6 48
TEST 1.6.7 48

MATCH_RESULTS_TEST_CASES.JAVA 49

BASIC FUNCTIONALITY 49

EXPECTED RESULTS 49

TEST 2.1 – FAIL WITH LIMIT 49
TEST 2.2 – MULTIPLE SUCCESSES MATCH 49
TEST 2.3 – MULTIPLE SUCCESSES MATCH 50
TEST 2.4 – SUCCESS WITH LARGE LIMIT 50
TEST 2.5 – SINGLE FAIL MATCH 50
TEST 2.6 – EMPTY MATCHES 50
TEST 2.7 – SUCCESS WITH LIMIT 50
TEST 2.8 – MULTIPLE FAIL MATCHES 50
TEST 2.9 – SINGLE SUCCESS MATCH 51

SPSM_FILTER_RESULTS_TEST_CASES.JAVA 52

BASIC FUNCTIONALITY 52

EXPECTED RESULTS 52

TEST 3.1 – SUCCESS SINGLE CALL 52

TEST 3.2 – FAIL WITH LIMIT 52
TEST 3.3 – SUCCESS MULTI CALL 53
TEST 3.4 – FAIL SINGLE CALL 53
TEST 3.5 – SUCCESS WITH LIMIT 53

REPAIR_SCHEMA_TEST_CASES.JAVA 54
BASIC FUNCTIONALITY 54
EXPECTED RESULTS 54
TEST 4.1.1 54
TEST 4.5.6 54
TEST 4.5.7 55

CREATE_QUERY_TEST_CASES.JAVA 56
BASIC FUNCTIONALITY 56
EXPECTED RESULTS 56
TEST 5.1.1 – SEPA QUERY 56
TEST 5.1.2 – SEPA QUERY 56
TEST 5.1.3 – SEPA QUERY 57
TEST 5.1.4 – SEPA QUERY 57
TEST 5.1.5 – SEPA QUERY 57
TEST 5.1.6 – SEPA QUERY 57
TEST 5.1.7 – SEPA QUERY 58
TEST 5.1.8 – SEPA QUERY 58
TEST 5.1.9 – SEPA QUERY 58
TEST 5.2.1 – DBPEDIA QUERY 58
TEST 5.2.2 – DBPEDIA QUERY 59
TEST 5.2.3 – DBPEDIA QUERY 59
TEST 5.2.4 – DBPEDIA QUERY 59
TEST 5.2.5 – DBPEDIA QUERY 59
TEST 5.2.6 – DBPEDIA QUERY 59
TEST 5.2.7 – DBPEDIA QUERY 60
TEST 5.2.8 – DBPEDIA QUERY 60
TEST 5.2.9 – DBPEDIA QUERY 60
TEST 5.2.10 – DBPEDIA QUERY 60
TEST 5.2.11 – DBPEDIA QUERY 60
TEST 5.2.12 – DBPEDIA QUERY 61
TEST 5.2.13 – DBPEDIA QUERY 61

RUN_QUERY_TEST_CASES.JAVA 62
BASIC FUNCTIONALITY 62
EXPECTED RESULTS 62
TEST 6.1.1 – SEPA QUERY 62
TEST 6.1.2 – SEPA QUERY 62
TEST 6.1.3 – SEPA QUERY 62
TEST 6.1.4 – SEPA QUERY 63
TEST 6.1.5 – SEPA QUERY 63

TEST 6.1.6 – SEPA QUERY....63
TEST 6.1.7 – SEPA QUERY....63
TEST 6.1.8 – SEPA QUERY....63
TEST 6.1.9 – SEPA QUERY....63
TEST 6.1.10 – SEPA QUERY....63
TEST 6.2.1 – DBPEDIA QUERY....63
TEST 6.2.2 – DBPEDIA QUERY....63
TEST 6.2.3 – DBPEDIA QUERY....64
TEST 6.2.4 – DBPEDIA QUERY....64
TEST 6.2.5 – DBPEDIA QUERY....64
TEST 6.2.6 – DBPEDIA QUERY....64
TEST 6.2.7 – DBPEDIA QUERY....64
TEST 6.2.8 – DBPEDIA QUERY....64
TEST 6.2.9 – DBPEDIA QUERY....64
TEST 6.2.10 – DBPEDIA QUERY....64
TEST 6.2.11 – DBPEDIA QUERY....64
TEST 6.2.12 – DBPEDIA QUERY....64
TEST 6.2.13 – DBPEDIA QUERY....64
TEST 6.2.14 – DBPEDIA QUERY....65
TEST 6.2.15 – DBPEDIA QUERY....65

SCHEMA_FROM_QUERY_TEST_CASES.JAVA....66
BASIC FUNCTIONALITY....66
EXPECTED RESULTS....66
TEST 7.1.1....66
TEST 7.1.2....66
TEST 7.1.3....66
TEST 7.1.4....66
TEST 7.1.5....67
TEST 7.1.6....67
TEST 7.1.7....67
TEST 7.1.8....67
TEST 7.1.9....67
TEST 7.1.10....67
TEST 7.2.1....67
TEST 7.2.2....67
TEST 7.2.3....67
TEST 7.2.4....67
TEST 7.2.5....68
TEST 7.2.6....68
TEST 7.2.7....68
TEST 7.2.8....68
TEST 7.2.9....68
TEST 7.2.10....68
TEST 7.2.11....68

TEST 7.2.12 68
TEST 7.2.13 69
RUN_CHAIN_TEST_CASES.JAVA 70
BASIC FUNCTIONALITY 70
EXPECTED RESULTS 70
TEST 8.1 70
TEST 8.2 70
TEST 8.3 70
TEST 8.4 71
TEST 8.5 71
TEST 8.6 71

OBJECTIVES OF CHAIN

The CHAIn (Combining Heterogeneous Agencies' Information) system dynamically re-writes queries to databases when mismatches led to query failure.

In general, queries to databases only succeed if they are correctly writing according to the schema of that database, which is only possible if the schema of the database is known in advance. During an emergency response (and, indeed, many other situations), this is plausible in some situations, but emergencies are characterised by their unpredictability and such foresight is not always possible. If one wishes to broadcast a query to all relevant agencies, or to a new agency, or to a previous collaborator who has updated their data sources, for example, a query may fail even if there is relevant data in the data source.

CHAIn sits locally within an agency. When a query to the data sources of that agency fails, CHAIn will use various matching techniques to determine on-the-fly whether there is mismatch between the schema of the query and the schema of the database and, if so, whether there is anything in the database that matches, or approximately matches the query. If so, CHAIn rewrites the query according to the schema of the database and sends back an appropriate response (or responses), together with a score indicating how good the match is and information about the assumptions and approximations made in the match.

SETTING UP CHAIN

1. After downloading the CHAIn project, run the following four files that can be found in *spsm/s-match-source* called *0cloneSMatch, 2buildSMatch, 3createBinary, 4extractBinary.*

   Doing this should create the following folders in that directory: *s-match-parent, s-match-core, s-match-io, s-match-logic, s-match-nlp, s-match-nlp-annotation, s-match-nlp-opennlp, s-match-spsm, s-match-wordnet, s-match-examples, s-match-utils, s-match-spsm-prolog.*

2. Open Eclipse and select *File > New > Project > Java Project* and enter the name for the project. Uncheck the *Use Default Location* radio button and set the location to point to where this project was downloaded to and click *Next*.

3. Select the *Libraries* tab and then click on *Add External JARS*, navigate to where this project was downloaded and into *spsm/s-match/lib* and select all the files in this directory and select *Open*. The files in *jena/apache-jena-2.12.1/lib* should also be added in the same way.

4. Then click on the *Add Library* button and make sure that the *JRE System Library* & *Junit 4* libraries have been added to the project.

   Note: If you want to access the libraries imported into the project after set-up then you should right click on the project in the *Package Explorer* on the left-hand side of Eclipse and choose *Build Path/Configure Build Path…*

5. Finally, click on *Finish*. You should now have access to all the source and testing files for this project.

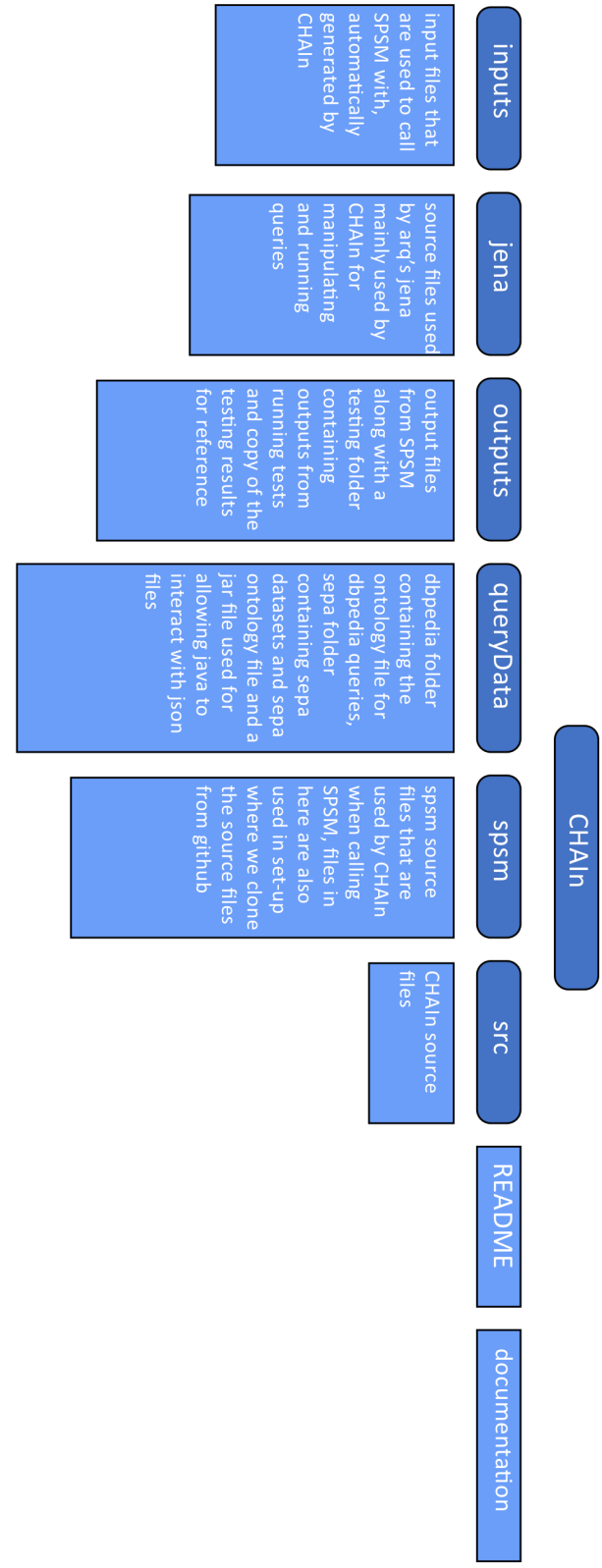
RUNNING FILES IN CHAIN

1. Select the file that you want to run by double clicking on the file in the *Package Explorer* panel on the left.

2. To run, click on the green start button on the top panel of Eclipse and there should be appropriate messages printed to the console to let you know what is happening.

Note: There are some files that will not do anything when you try to run them. You can check what should happen when you run a specific file by looking up that file in this document.

FILE STRUCTURE



TESTING RESULT DOCUMENTS

All tests will write their results to a text file that can be found in the *outputs/testing* folder. These files have been structured to give details about each individual test including what schemas CHAIn is being run with, what the expected result is and what the actual result is that is returned.

Note: When running the testing files for the first time you might need to select the option to run the file using *JUnit*.

BREAKDOWN OF TESTING DOCUMENTS

Task 1 Testing Documents

- *SPSM_Test_Cases.java* (which produces *Call_SPSM_Tests.txt)*
- *Match_Results_Test_Cases.java* (which produces *Filter_Limit_Tests.txt)*
- *SPSM_Filter_Results_Test_Cases.java* (which produces *Task_1_Tests.txt)*

These files can be run at once by running *Task1_Test_Suite.java*

Task 2 Testing Documents

- *Repair_Schema_Test_Cases.java* (which produces *Schema_Repair_Tests.txt)*
- *Create_Query_Test_Cases.java* (which produces *Create_Queries_Tests.txt)*
- *Run_Query_Test_Cases.java* (which produces *Run_Queries_Tests.txt*)

These files can be run at once by running *Task2_Test_Suite.java*

Task 4 Testing Documents

- *Schema_From_Query_Test_Cases.java* (which produces *Schema_From_Query_Tests.txt*)

Overall CHAIn Functionality

- *Run_CHAIn_Test_Cases.java* (which produces *Run_CHAIn_Test.txt*)

DESCRIPTION OF IMPLEMENTATION FILES

MATCH_STRUC.JAVA

BASIC FUNCTIONALITY

Data structure that has been created to store the results that are returned after calling SPSM. The structure allows for easy interaction to find out information about the match details with a certain target schema.

DESIGN

This java file represents a class that is responsible for creating an object that is used to store details about a schema/query to allow repairs to take place to successfully query a dataset.

PRIVATE VARIABLES/FIELDS

similarity

This is the variable that stores the similarity value between the source and this current target schema. (double)

dataset

This variable holds the name of the target schema that this structure is representing the results for. (String)

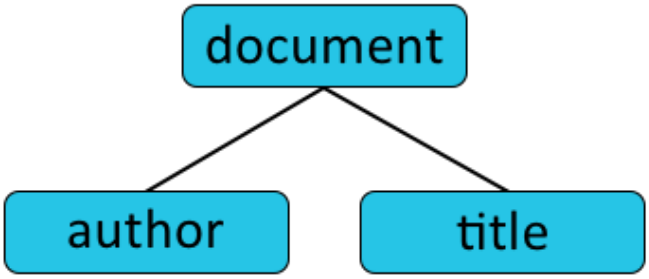
matches

List of arrays containing the match concepts between the source and this current target schema. (ArrayList of String arrays)

repairedSchemaTree

Tree structure that represents the repaired schema in a tree form. (Node)

For example: If the schema is *document(title,author)* then the tree will look like the following where *document* is the root node with two child nodes with the values *author* and *title*,



repairedSchema

This variable holds the repaired schema in a format that can easily be read. (String)

numMatches

Stores the total number matches which is calculated by the number of elements in the matches ArrayList. (int)

query

Stores the query that has been produced using the repaired schema (*String*)

METHODS

public Match_Struc (double sim, String targetSchema)

Constructor that is responsible for initialising the Match_Struc variable with the similarity value and the name of the target schema.

public void setSimilarity (double sim)

Sets the similarity of this match to be the value passed in as the parameter, *sim.*

public void setDatasetSchema (String targetSchema)

Sets the target schema to be the String that is passed in as the parameter, *targetSchema*.

public void addMatch (String[] match)

Responsible for adding another match concept to the list of matches stored in the ArrayList<String[]> called *matches.*

public int getNumMatches ()

Returns the total number of matches based on the number of elements inside *matches*.

public double getSimValue ()

Returns the similarity value for that target schema.

public String getDatasetSchema ()

Returns the name of the target schema.

public ArrayList<String[]> getMatches ()

Returns the ArrayList of arrays containing the match concept details.

public String[] getMatchAtIndex (int index)

Returns the specific match concept detail at the specified index passed as a parameter, *index*.

public void setRepairedSchemaTree (Node schemaTree)

Sets the repaired schema tree from the parameter, *schemaTree*.

public void setRepairedSchema (String stringSchema)

Sets the String version of the repaired schema from the parameter, *stringSchema*.

public String getRepairedSchema ()

Returns the String version of the repaired schema.

public Node getRepairedSchemaTree ()

Returns the repaired schema as a tree structure.

public void setQuery(String matchQuery)

Sets the query for this match structure.

public String getQuery()

Returns the query for this structure.

ASSOCIATED TESTING DOCUMENTS

SPSM_Test_Cases.java (which produces *Call_SPSM_Tests.txt*)

*Match_Results_Test_Cases.java (*which produces *Filter_Limit_Test.txt)*

SPSM_Filter_Results_Test_Cases.java (which produces *Task_1_Tests.txt)*

Task1_Test_Suite.java (which runs the testing files for task 1)

RUNNING THIS FILE

-

CALL_SPSM.JAVA

BASIC FUNCTIONALITY

This class is responsible for initially taking in both the target and source schemas before calling SPSM with these schemas. It will then store the results as an ArrayList of Match_Struc objects.

DESIGN

Inputs

- ArrayList of type Match_Struc objects where you want the results to be stored
- String version of source schema
- String version of target schema

Process

- Calls SPSM with the source and target schema, then picks out the returned results and stores as Match_Struc objects into the ArrayList that was passed in as an input

Outputs

- ArrayList of type Match_Struc objects containing the results from calling SPSM ***OR*** null if there has been a problem with calling SPSM

PRIVATE VARIABLES/FIELDS

targetList

Used temporarily for splitting the target schema String if there are more than one target schema within the String. This allows the application to then call SPSM for each individual target schema. (*String[]*)

METHODS

public ArrayList<Match_Struc> getSchemas (ArrayList<Match_Struc> results, String sourceSchema, String targetSchema)

Retrieve the schemas from the user, either through the console or as a parameter which gets written to their respective files. From there we can start to make a call to SPSM with the current target.

public ArrayList<Match_Struc> makeCallToSPSM (ArrayList<Match_Struc results, String currTarget)

Makes the call to SPSM using a bash file called *call-spsm.sh.* This should allow for SPSM to run and then write the results out as a serialised object to *serialised-results.ser*. It will then call the appropriate method to read this file and store the results in a Match_Struc object.

public ArrayList<Match_Struc> recordSerialisedResults (ArrayList<Match_Struc> results, String targetSchema)

Responsible for reading the serialised object from *serialised-results.ser* and calls the appropriate method that will then go through the different pieces of information stored within this object and stores what is required for CHAIn.

public ArrayList<Match_Struc> readObject (ArrayList<Match_Struc> results, String targetSchema, IContextMapping<INode> mapping)

Reads the information stored in the object read in and picks out the data that is required for use by CHAIn.

public String getNodePathString (INode node)

Responsible for taking in a *INode* object and turns it into something readable so that we can store the match concepts in a convenient way.

ASSOCIATED TESTING DOCUMENTS

SPSM_Test_Cases.java (which produces *Call_SPSM_Tests.txt*)

SPSM_Filter_Results_Test_Cases.java (which produces *Task_1_Tests.txt)*

Task1_Test_Suite.java (which runs the testing files for task 1)

RUNNING THIS FILE

Running this file will call SPSM with the source and target schemas that have been declared in the *main* method and will then print out the results that have been returned from SPSM by printing out the target schema, it's similarity to the source schema and the match concepts.

BEST_MATCH_RESULTS.JAVA

BASIC FUNCTIONALITY

Responsible for taking results after calling SPSM and then filtering to ensure each result is over the determined threshold, cut the number of results to the desired *n* and sort them so that the match with the highest similarity is at the top of the list.

DESIGN

Inputs

- ArrayList of type Match_Struc containing a list of matches returned from calling SPSM
- The threshold that you want the similarity value of the results to have as a minimum (where passing in 0 will not filter/limit the results)
- The maximum number of results you want returned (where passing in 0 will not filter/limit the results)

Process

- After taking in the list of results after calling SPSM, we remove any elements in the list that are not equal to or greater than the threshold value and then sort the list so that elements are in descending order based on their similarity value. Finally, the list is restricted in size based on the maximum number of elements that were requested

Outputs

- ArrayList of Match_Struc objects that have been filtered based on the threshold value and limit that has been passed in

PRIVATE VARIABLES/FIELDS

-

METHODS

public ArrayList<Match_Struc> getThresholdAndFilter (ArrayList<Match_Struc> results, Double threshVal, int limNum)

Start off by taking in the results, threshold value and the number of results wanted but if these values haven't been passed in through parameters, then ask the user to enter them through the console. It will then strip off any matches that are lower than the threshold value that has been entered.

public ArrayList<Match_Struc> sortResultingMatches (ArrayList<Match_Struc> filteredRes, int limitNo)

This method will sort the resulting matches after they have been filtered to ensure that only those greater than or equal to the threshold value are left. These matches are sorted so that the matches with the highest similarity value is at the top of the list.

public void displayResults (ArrayList<Match_Struc> res)

This method is used to print out the results after they have been filtered and sorted.

ASSOCIATED TESTING DOCUMENTS

Match_Results_Test_Cases.java (which produces *Filter_Limit_Tests.txt)*

SPSM_Filter_Results_Test_Cases.java (which produces *Task_1_Tests.txt)*

Task1_Test_Suite.java (which runs the testing files for task 1)

RUNNING THIS FILE

Running this file will use the Match_Struc variables in the list of results and filter the list to ensure that anything in the list if over a specified threshold and that there are n number of entries if specified by the user. The *main* method will then print out each element in the list after filtering and sorting the list.

REPAIR_SCHEMA.JAVA

BASIC FUNCTIONALITY

Responsible for taking in an ArrayList of Match_Struc objects and going through each one and creating a repaired schema based on the relation between the original source and target schemas.

DESIGN

Inputs

- ArrayList of Match_Struc objects

Process

- For each of the elements in the list passed in, create a repaired schema based on the details from comparing the source and target schemas after calling SPSM

Outputs

- ArrayList of Match_Struc objects where each element now has a repaired schema attribute/field

PRIVATE VARIABLES/FIELDS

-

METHODS

public ArrayList<Match_Struc> prepare (ArrayList<Match_Struc> matchRes)

Takes in the ArrayList of objects and starts processing them in turn to create a repaired schema.

public Match_Struc repairSchema(Match_Struc matchStructure)

Creates a tree structure of the schema and adds that as a field to the Match_Struc object based on the relation between the source and target schemas. It also adds a String version of the repaired schema as a field to the Match_Struc object.

public Node modifyRepairedTree(Node root, String concept)

Using the relation concept data, the tree is modified to ensure that there are only links to parts of the schema that we think there is a match with. This tree is then returned.

ASSOCIATED TESTING DOCUMENTS

Repair_Schema_Test_Cases.java (which produces *Schema_Repair_Tests.txt*)

Create_Query_Test_Cases.java (which produces *Create_Queries_Tests.txt*)

RUNNING THIS FILE

Running this file will go take in the source and target schemas, call SPSM and then create the repaired schema based on the match details between these two original schemas. The results will be printed to the console.

NODE.JAVA

BASIC FUNCTIONALITY

Structure that is being used to store the repaired schema tree that consists of a node that has a value and a list of children nodes.

DESIGN

This java file represents a class that is responsible for creating a tree object that is used to store the breakdown of a schema.

PRIVATE VARIABLES/FIELDS

value

Holds the value of the node such as the predicate or parameter name (*String*)

children

The list of nodes that are the children of the current node (*List<Node>*)

METHODS

public Node(String val)

The constructor that creates the node structure and sets the value of that node.

public void addChild(Node child)

Adds the node, *child*, to the list of children nodes.

public String getValue()

Returns the value of the current node as a String.

public List<Node> getChildren()

Returns the list of children nodes.

public ArrayList<String> getChildrenValues()

Returns a list containing the values of all the children nodes.

public Boolean hasChildren()

Returns whether or not the node has any children nodes.

public int getNumChildren()

Returns the total number of children nodes.

public String printTree()

Prints out the tree as a repaired schema.

public String printChildren(List<Node> childrenList)

Prints out the children in the tree to create the repaired schema as a *String*.

public void addToTree(String[] paramParts)

Modifies the tree to add a list of parameters correctly into the current tree.

ASSOCIATED TESTING DOCUMENTS

Repair_Schema_Test_Cases.java (which produces *Schema_Repair_Tests.txt)*

RUNNING THIS FILE

-

ONTOLOGY_STRUC.JAVA

BASIC FUNCTIONALITY

Structure that is used to store the ontology structure being used to create the queries.

DESIGN

This java file represents a class that is responsible for storing the details of an ontology file to allow for other methods to access whilst creating queries.

PRIVATE VARIABLES/FIELDS

name

The name of the ontology structure (*String*)

link

The web link of the ontology structure (*String*)

properties

The list of key words or properties associated with the structure (*String[]*)

METHODS

public Ontology_Struc(String ontName, String ontLink, String[] ontProperties)

Constructor for creating an ontology structure.

public void setName(String ontName)

Sets the name of the structure.

public String getName()

Returns the name of the structure.

public void setLink(String ontLink)

Sets the link for the structure.

public String getLink()

Returns the link of the structure.

public void setProperties(String[] ontProperties)

Sets the properties or key words associated with the structure.

public String[] getProperties()

Returns the list of key words or properties associated with the structure.

ASSOCIATED TESTING DOCUMENTS

Create_Query_Test_Cases.java (which produces *Create_Queries_Tests.txt*)

RUNNING THIS FILE

-

CREATE_QUERY.JAVA

BASIC FUNCTIONALITY

Responsible for using the repaired schema to create either a sepa or dbpedia query, this query is then added as a field onto the Match_Struc object structure.

DESIGN

Inputs

- ArrayList of Match_Struc objects with elements containing repaired schemas that you want to create queries for
- The query type as a string (either sepa or dbpedia)
- The directory of the dataset if local as a string otherwise null
- The maximum number of results you want returned

Process

- For each of the elements in the list of Match_Struc objects, we create either a sepa or dbpedia query based on the repaired schema that is stored as a field of a Match_Struc element. This query is then stored as an attribute/field

Outputs

- ArrayList of Match_Struc with elements now containing a query as a string

PRIVATE VARIABLES/FIELDS

-

METHODS

public ArrayList<Match_Struc> createQueryPrep(ArrayList<Match_Struc> matchRes, String queryType, String additionalInfo)

Sets up so that the methods can create a query for each of the repaired schemas in the list of Match_Struc objects.

public String createSepaQuery(Match_Struc matchDetails, String datafileDir, int noResults)

Creates structure for a local sepa query.

public String createDbpediaQuery(Match_Struc matchDetails, int noResults)

Creates structure for a remote dbpedia query.

public ArrayList<Ontology_Struc> setupPrefixes(JSONArray jsonArr, ArrayList<Ontology_Struc> ontologies)

Reads ontology file and processes it into Ontology_Struc objects.

public String writePrefix(ArrayList<Ontology_Struc> ontologies)

Writes prefixes for query through reading the Ontology_Struc objects.

public String dataMatching(List<Node> children, ArrayList<Ontology_Struc> ontologies)

Finds what ontology file a specific key word or property is from or relates to.

ASSOCIATED TESTING DOCUMENTS

Create_Query_Test_Cases.txt (which produces *Create_Queries_Tests.txt*)

RUNNING THIS FILE

Running this file will setup a sepa or dbpedia query based on the repaired schema produced from calling SPSM on the target and source schemas. These details are specified in the *main* method. It also prints out the repaired schema and the query string created.

RUN_QUERY.JAVA

BASIC FUNCTIONALITY

Responsible for running either a sepa or dbpedia query.

DESIGN

Inputs

- Match_Struc object containing the query as a string that you want to run
- The type of query as a string (either sepa or dbpedia)
- The directory of the dataset as a string if local otherwise null

Process

- The query stored as a field on the Match_Struc object that has been passed in gets extracted and then the query gets setup and run with the details of this dependent on whether it is a sepa or dbpedia query

Outputs

- Results from running the query or error messages if the query was unable to run are printed to the console

PRIVATE VARIABLES/FIELDS

-

METHODS

public boolean runQuery(Match_Struc current, String queryType, String datasetDir)

Responsible for selecting the correct method for running either a sepa or dbpedia query based on the *queryType* parameter that has been passed in.

public boolean runSepaQuery(String query, String datasetToUseDir, Match_Struc currMatchStruc)

Responsible for running the sepa query that has been passed into this method.

public boolean runDbpediaQuery(String query, Match_Struc currMatchStruc)

Responsible for running the dbpedia query that has been passed into this method.

ASSOCIATED TESTING DOCUMENTS

Run_Query_Test_Cases (which produces *Run_Queries_Test.txt*)

RUNNING THIS FILE

Running this file will run either a sepa or dbpedia query depending on the setup in the *main method.* It also prints out the query that it is trying to run followed by the results if the query has run successfully with data returned.

SCHEMA_FROM_QUERY.JAVA

BASIC FUNCTIONALITY

Responsible for taking in either a sepa or dbpedia query that is valid and creating a CHAIn schema for that specific query.

DESIGN

Inputs

- Query as a string
- The type of the query as a string (either sepa or dbpedia)

Process

- Parts of the query are extracted to create a schema represented as a tree which is then stored in a Match_Struc object

Outputs

- Match_Struc containing the details of the schema created from the query that was originally passed in

PRIVATE VARIABLES/FIELDS

-

METHODS

public Match_Struc getSchemaFromQuery(String query, String queryType)

Initial method that gets passed in the query as a string and the type of query it is (either sepa or dbpedia). This information is then used to determine whether we need to parse a sepa or dbpedia query and calls methods accordingly to get this information. Also responsible for returning the result as a Match_Struc object with the schema stored as the repaired schema string and also represented as a tree.

public Match_Struc schemaFromSepaQuery(String query, Match_Struc res)

Responsible for taking in the sepa query and using parts of this query in order to create a schema specific for it. This schema is then added to the *res* parameter as a string and represented as a tree before returning it.

public Match_Struc schemeFromDbpediaQuery(String query, Match_Struc res)

Responsible for taking in the dbpedia and using parts of this query in order to create a schema specific for it. This schema is then added to the *res* parameter as a string and represented as a tree before returning it.

public Node createTreeFromSchemaString(String schemaStr)

Takes in the schema represented as a string and parses this string to create the tree representation of the schema which then gets returned.

ASSOCIATED TESTING DOCUMENTS

Schema_From_Query_Test_Cases.java (which produces *Schema_From_Query_Tests.txt*)

RUNNING THIS FILE

Running this file will create a schema for either a sepa or dbpedia query depending on the setup in the *main method.* It then prints out the schema that has been created.

RUN_CHAIN.JAVA

BASIC FUNCTIONALITY

Responsible for running the overall CHAIn functionality when passed in a query, target schema, the number of results you want returned from the query, the similarity value threshold and the maximum number of queries you want produced.

DESIGN

Inputs

- The query as a string
- The type of query as a string (either sepa or dbpedia)
- The target schema(s) as a string with each schema being separated by a ';'
- The maximum number of results you want printed after running the new queries
- The threshold of the similarity value that you want matches from SPSM to meet the condition of (where passing in 0 will not limit/filter the results)
- The maximum number of matches from SPSM that you want to create queries from (where passing in 0 will not limit/filter the results)
- A PrintWriter that you want the results to be printed out to ***OR*** null

Process

- Start off by trying to run the initial query that has been passed in and if this query does not successfully run with no results returned then it will call SPSM with the schema from this query along with the target schemas to create new queries from this information that will then be run

Outputs

- There are no data structures returned from this, however, there will be messages printed to the console and if you choose to pass in a PrintWriter then a file will be

produced with commentary of what happened during runtime. This file can be found in *outputs/testing/Run_CHAIn_Tests.txt*.

PRIVATE VARIABLES/FIELDS

getSchema

The instance of the class Schema_From_Query.java that extracts a schema from a query (*Schema_From_Query*)

spsm

The instance of the class Call_SPSM.java that calls SPSM with two schemas and works out the similarity between the two (*Call_SPSM*)

filterRes

The instance of the class Best_Match_Results.java that filters the results after calling SPSM (*Best_Match_Results*)

repairSchema

The instance of the class Repair_Schema.java that uses the match concept information returned from SPSM in order to create a schema (*Repair_Schema*)

createQuery

The instance of the class Create_Query.java that creates either a sepa or dbpedia query depending on the schema that has been created (*Create_Query*)

runQuery

The instance of the class Run_Query.java that runs either a sepa or dbpedia query (*Run_Query*)

METHODS

public void startCHAIn (String query, String queryType, String targetSchemas, int queryLim, double simThresholdVal, int resLimit, PrintWriter fOut)

This method starts off by trying to run the query that has been passed in and if it runs successfully then returns. If we have an invalid query that cannot be run then we also return. Otherwise we start to repair and once that has been completed we try to run the new queries that have been produced.

public ArrayList<Match_Struc> startRepair(Match_Struc current, String targetSchemas, String queryType, String dataset, int queryLim, double simThresholdVal, int resLimit)

Starts off by calling SPSM with the schema extracted from the initial query along with the target schemas that have been passed in by the user. If SPSM returns no results then we return null otherwise we filter the results, create a repaired schema and then create the associated query for that schema. This information is stored as a list of Match_Struc objects which is then returned as an ArrayList.

public Boolean runRepairedQueries(Match_Struc res, String queryType, String dataset)

This method tries to run the query that is attached to the Match_Struc object that has been passed in where if the query does not successfully run and return results then we return false otherwise return true.

ASSOCIATED TESTING DOCUMENTS

Run_CHAIn_Test_Cases.java (which produces *Run_CHAIn_Tests.txt*)

RUNNING THIS FILE

Running this file will carry out the overall CHAIn functionality that has been described whilst printing messages to the console. If the user chooses to pass in a PrintWriter then extra detail about what CHAIn has done will be printed out to the file that can be found in *outputs/testing/Run_CHAIn_Tests.txt*.

DESCRIPTION OF TESTING FILES

SPSM_TEST_CASES.JAVA

BASIC FUNCTIONALITY

Responsible for testing the output results from *Call_SPSM.java* which takes in source and target schemas and then calls SPSM before reading in results as a serialised object.

After running this test, the results will have been documented in *outputs/testing/Call_SPSM_Test.txt* where you will be able to see the test number, the target and source schemas that SPSM has been called with, the expected result and the actual result. This test should be hand checked.

Note: These tests will be named with a prefix of 1. For example, test 1.1.1, refers to test 1.1 of SPSM_Test_Cases.java.

EXPECTED RESULTS

TEST 1.1.1

This test calls SPSM with empty source and target schemas, therefore, the size of the results returned is 0.

TEST 1.2.1

This test calls SPSM with a source schema but no target schema, therefore, the size of the results returned is 0.

TEST 1.2.2

This test calls SPSM with a target schema but no source schema, therefore, the size of the results returned is 0.

TEST 1.2.3

This test calls SPSM with the same source and target schema, *author(name)*. Therefore, the similarity value returned should be 1.0 with 2 individual matches.

TEST 1.2.4

This test calls SPSM with a single source schema and three target schemas, where only two of these target schemas have a level of similarity with the source schema. Where *author(name ,document)* has a similarity of 1.0 and 2 individual matches and *reviewAuthor(firstname, lastname ,review)* has a similarity of 0.75 and 2 individual matches.

TEST 1.3.1

This test calls SPSM with a single source schema and two target schemas, however, there are no matches between the source and target schemas. Therefore, the size of the results returned is 0.

TEST 1.3.2

This test calls SPSM with a single source schema and two target schemas where only one of these target schemas matches with the source. So only one of the targets has match data. This target is *author(name)* that has a similarity of 1.0 and 2 individual matches.

TEST 1.3.3

This test calls SPSM with a single source schema and four target schemas where only three of these target schemas matches with the source. Therefore, we have three matches in our list of results. Where *author(name)* has a similarity of 1.0 and 2 individual matches, *paperWriter(firstname, surname, paper)* has a similarity of 0.5 and 2 individual matches and *reviewAuthor(firstname, lastname, review)* has a similarity of 0.75 and 2 individual matches.

TEST 1.4.1

This test calls SPSM with a single source and target schema that has a similarity value of 1.0 and 1 individual match.

TEST 1.4.2

This test calls SPSM with a single source and target schema that do not have any similarity, therefore, the size of the results returned is 0.

TEST 1.4.3

This test calls SPSM with a single source and target schema that do not have any similarity, therefore, the size of the results returned is 0.

TEST 1.4.4

This test calls SPSM with a single source and target schema that have a similarity value of 0.5 and 2 individual matches.

TEST 1.4.5

This test calls SPSM with a single source and target schema, however, this is an example that causes SPSM to crash. Therefore, returning no data to CHAIn so we have null results.

TEST 1.5.1

This test calls SPSM with a single source and target schema where it was expected that there would be a similarity of 1.0 returned, however, a similarity of 0.5 was returned with the expected 5 individual matches.

TEST 1.5.2

This test calls SPSM with a single source and target schema that returns a similarity of 0.6 and 5 individual matches.

TEST 1.5.3

This test calls SPSM with a single source and target schema that do not have a similarity, therefore, the size of the results is 0.

TEST 1.5.4

This test calls SPSM with a single source and target schema, however, this is an example that causes SPSM to crash. Therefore, returning no data to CHAIn so we have null results.

TEST 1.5.5

This test calls SPSM with a single source and target schema, however, this is an example that causes SPSM to crash. Therefore, returning no data to CHAIn so we have null results.

TEST 1.5.6

This test calls SPSM with a single source and target schema where the similarity between the two is 0.625 and 12 individual matches.

TEST 1.5.7

This test calls SPSM with a single source and target schema, however, this is an example that causes SPSM to crash. Therefore, returning no data to CHAIn so we have null results.

TEST 1.5.8

This test calls SPSM with a single source and target schema where there are no similarities between the two, therefore, the size of the results returned is 0.

TEST 1.6.1

This test calls SPSM with a single source and target schema where there is a similarity value of 0.5 and 2 individual matches.

TEST 1.6.2

This test calls SPSM with a single source and target schema where there is a similarity value of 0.5 and 2 individual matches.

TEST 1.6.3

This test calls SPSM with a single source and target schema where there is a similarity value of 0.5 and 2 individual matches.

TEST 1.6.4

This test calls SPSM with a single source and target schema where there is a similarity value of 0.5 and 2 individual matches.

TEST 1.6.5

This test calls SPSM with a single source and target schema where there is a similarity value of 0.5 and 2 individual matches.

TEST 1.6.6

This test calls SPSM with a single source and target schema where there is a similarity value of 0.5 and 2 individual matches.

TEST 1.6.7

This test calls SPSM with a single source and target schema, however, this is an example that causes SPSM to crash. Therefore, returning no data to CHAIn so we have null results.

MATCH_RESULTS_TEST_CASES.JAVA

BASIC FUNCTIONALITY

Responsible for testing *Best_Match_Results.java* file to make sure that when we pass in the results that we have received from SPSM, we get returned only results that have similarity values over the threshold value that are sorted and that we only have n number of results as requested.

After running this test, the results will have been documented in *outputs/testing/Filter_Limit_Test.txt* where you will be able to see the test number, what the threshold value and limit values are that *Best_Match_Results.java* got called with, the expected result and the actual result. This test should be hand checked.

Note: These tests will be named with a prefix of 2. For example, test 2.1, refers to test 1 of Match_Results_Test_Cases.java.

EXPECTED RESULTS

TEST 2.1 – FAIL WITH LIMIT

This test calls *Best_Match_Results.java* with a threshold value of 1.0 and a limit of 5, therefore, the size of the results is 0 because there are no entries matching this criteria.

TEST 2.2 – MULTIPLE SUCCESSES MATCH

This test calls *Best_Match_Results.java* with a threshold value of 0.6 and a limit of 0 meaning that there is no restriction on the number of results returned. This causes the size of the results to be 2 afterwards meaning that there are 2 entries with a threshold greater than or equal to 0.6.

TEST 2.3 – MULTIPLE SUCCESSES MATCH

This test calls *Best_Match_Results.java* with a threshold value of 0.2 and a limit of 0 meaning that there is no restriction on the number of results returned. This causes the size of the results to be 4 afterwards meaning that there are 4 entries with a threshold greater than or equal to 0.2.

TEST 2.4 – SUCCESS WITH LARGE LIMIT

This test calls *Best_Match_Results.java* with a threshold value of 0.2 and a limit of 5 which causes the size of the results to be 4 afterwards meaning that there are 4 entries with a threshold values greater than or equal to 0.2.

TEST 2.5 – SINGLE FAIL MATCH

This test calls *Best_Match_Results.java* with a threshold value of 0.5 and a limit of 0 meaning that there is no restriction on the number of results returned. This causes the size of the results to be 0 afterwards meaning that there are 0 entries with a threshold greater than or equal to 0.5.

TEST 2.6 – EMPTY MATCHES

This test calls *Best_Match_Results.java* with a threshold value of 0.2 and a limit of 0 meaning that there is no restriction on the number of results returned. This causes the size of the results to be 0 afterwards meaning that there are 0 entries with a threshold greater than or equal to 0.5.

TEST 2.7 – SUCCESS WITH LIMIT

This test calls *Best_Match_Results.java* with a threshold value of 0.2 and a limit of 3 which causes the size of the results to be 3 afterwards meaning that there are at least 3 entries with a threshold value greater than or equal to 0.2.

TEST 2.8 – MULTIPLE FAIL MATCHES

This test calls *Best_Match_Results.java* with a threshold value of 1.0 and a limit of 0 meaning that there is no restriction on the number of results returned. This causes the size of the results to be 0 afterwards meaning that there are 0 entries with a threshold greater than or equal to 1.0.

TEST 2.9 – SINGLE SUCCESS MATCH

This test calls *Best_Match_Results.java* with a threshold value of 0.1 and a limit of 0 meaning that there is no restriction on the number of results returned. This causes the size of the results to be 1 afterwards meaning that there was 1 entry with a threshold value greater than or equal to 0.1.

SPSM_FILTER_RESULTS_TEST_CASES.JAVA

BASIC FUNCTIONALITY

Responsible for testing *Call_SPSM.java* and *Best_Match_Results.java* to make sure that after calling SPSM and filtering the results based on the restrictions entered, we receive the appropriate results afterwards.

After running this test, the results will have been documented in *outputs/testing/Task_1_Tests.txt* where you will be able to see the test number, the source and target schemas that SPSM was called with, what the threshold value and limit values are that *Best_Match_Results.java* got called with, the expected result and the actual result. This test should be hand checked.

Note: These tests will be named with a prefix of 3. For example, test 3.1, refers to test 1 of SPSM_Filter_Results_Test_Cases.java.

EXPECTED RESULTS

TEST 3.1 – SUCCESS SINGLE CALL

This test calls SPSM with a single source and target schema that are the same and filters the results with a threshold value of 0.6 and a limit of 0 (meaning that there are no restrictions on the number of results returned). This causes the size of the results to be 1 meaning that there is only 1 result returned from SPSM that matches the filtering criteria.

TEST 3.2 – FAIL WITH LIMIT

This test calls SPSM with a single source and target schema and filters the results with a threshold of 0.0 and a limit of 2. This causes the size of the results to be 0 meaning that there are no results returned from SPSM that matches the filtering criteria.

TEST 3.3 – SUCCESS MULTI CALL

This test calls SPSM with a single source and 4 target schemas and then filters the results with a threshold value of 0.6 and a limit of 0 (meaning that there are no restrictions on the number of results returned). This causes the size of the results to be 2 meaning that there are 2 results that have been returned from SPSM that matches the filtering criteria.

TEST 3.4 – FAIL SINGLE CALL

This test calls SPSM with a single source and target schema and then filters the results with a threshold value of 0.0 and a limit of 0 (meaning that there are no restrictions on the number of results returned). This causes the size of the results to be 0 meaning that there are 0 results that have been returned from SPSM that matches the filtering criteria.

TEST 3.5 – SUCCESS WITH LIMIT

This test calls SPSM with a single source and 4 target schemas and then filters the results with a threshold value of 0.0 and a limit of 2. This causes the size of the results to be 2 meaning that there are 2 results that have been returned from SPSM that matches the filtering criteria.

REPAIR_SCHEMA_TEST_CASES.JAVA

BASIC FUNCTIONALITY

Responsible for testing the element of CHAIn that after calling SPSM and filtering the results will try to create a repaired schema based on the match data between the source and target schemas.

After running this test, the results will have been documented in *outputs/testing/Schema_Repair_Tests.txt* where you will be able to see the test number, the source and target schemas that SPSM was called with, the expected result and the actual result. This test should be hand checked.

Note: These tests will be named with a prefix of 4. For example, test 4.1.1, refers to test 1.1 of Repair_Schema_Test_Cases.java.

EXPECTED RESULTS

TEST 4.1.1

This test calls SPSM with a single source and target schema with the expected repaired schema to be *car(colour, brand)*. However, this is an example that causes SPSM to crash causing no data to be returned to CHAIn so we have null results.

TEST 4.5.6

This test calls SPSM with a single source and target schema with the expected repaired schema to be *conference(paper(title, document(date(day, month, year), writer(name(first, second))))).* The actual repaired schema that gets generated does match this, however, it should be noted that the parameters are in a slightly different order but still have the same meaning.

TEST 4.5.7

This test calls SPSM with a signle source and target schema with the expected repaired schema to be *conference(paper(title, document(writer(name(first, second))))).* However, this is an example that causes SPSM to crash causing no data to be returned to CHAIn so we have null results.

CREATE_QUERY_TEST_CASES.JAVA

BASIC FUNCTIONALITY

Responsible for testing *Create_Query.java* to ensure that after calling SPSM and repairing the target schema we can create the appropriate sepa or dbpedia query.

After running this test, the results will have been documented in *outputs/testing/Create_Queries_Tests.txt* where you will be able to see the test number, the schema that the query is being created based on, the expected result and the actual result.

Note: These tests will be named with a prefix of 5. For example, test 5.1.1, refers to test 1.1 of Create_Query_Test_Cases.java.

EXPECTED RESULTS

TEST 5.1.1 – SEPA QUERY

This test creates a sepa query based on the schema,

waterBodyPressures(dataSource, identifiedDate, affectsGroundwater, waterBodyId)

This creates the appropriate query based on this schema although it should be noted that the parameters are in a slightly different order in the query that is created in comparison to the expected query.

TEST 5.1.2 – SEPA QUERY

This test creates a sepa query based on the schema,

water(timePeriod, geo, measure, resource)

This creates the appropriate query based on this schema although it should be noted that the parameters are in a slightly different order in the query that is created in comparison to the expected query.

TEST 5.1.3 – SEPA QUERY

This test creates a sepa query based on the schema,

waterBodyMeasures(timePeriod, geo, measure, resource)

This creates the appropriate query based on this schema although it should be noted that the parameters are in a slightly different order in the query that is created in comparison to the expected query.

TEST 5.1.4 – SEPA QUERY

This test creates a sepa query based on the schema,

waterBodyPressures(identifiedDate, waterBodyId, assessmentCategory, source)

This creates the appropriate query based on this schema although it should be noted that the parameters are in a slightly different order in the query that is created in comparison to the expected query.

TEST 5.1.5 – SEPA QUERY

This test creates a sepa query based on the schema,

waterBodyMeasures(waterBodyId, secondaryMeasure, dataSource)

This creates the appropriate query based on this schema although it should be noted that the parameters are in a slightly different order in the query that is created in comparison to the expected query.

TEST 5.1.6 – SEPA QUERY

This test creates a sepa query based on the schema,

surfaceWaterBodies(riverName, associatedGroundwaterId)

This creates the appropriate query based on this schema although it should be noted that the parameters are in a slightly different order in the query that is created in comparison to the expected query.

TEST 5.1.7 – SEPA QUERY

This test creates a sepa query based on the schema,

bathingWaters(catchment, localAuthority, lat, long)

This creates the appropriate query based on this schema although it should be noted that the parameters are in a slightly different order in the query that is created in comparison to the expected query.

TEST 5.1.8 – SEPA QUERY

This test creates a sepa query based on the schema,

surfaceWaterBodies(subBasinDistrict, riverName, altitudeTypology, associatedGroundwaterId)

This creates the appropriate query based on this schema although it should be noted that the parameters are in a slightly different order in the query that is created in comparison to the expected query.

TEST 5.1.9 – SEPA QUERY

This test creates a sepa query based on the schema,

bathingWaters(bathingWaterId)

This creates the appropriate query based on this schema although it should be noted that the parameters are in a slightly different order in the query that is created in comparison to the expected query.

TEST 5.2.1 – DBPEDIA QUERY

This test creates a dbpedia query based on the schema,

City(country, populationTotal)

This creates the appropriate query based on this schema although it should be noted that the parameters are in a slightly different order in the query that is created in comparison to the expected query.

TEST 5.2.2 – DBPEDIA QUERY

This test creates a dbpedia query based on the schema,

Country

This creates the appropriate query based on this schema.

TEST 5.2.3 – DBPEDIA QUERY

This test creates a dbpedia query based on the schema,

Astronaut(nationality)

This creates the appropriate query based on this schema.

TEST 5.2.4 – DBPEDIA QUERY

This test creates a dbpedia query based on the schema,

Mountain(elevation)

This creates the appropriate query based on this schema.

TEST 5.2.5 – DBPEDIA QUERY

This test creates a dbpedia query based on the schema,

Person(occupation, birthPlace)

This creates the appropriate query based on this schema.

TEST 5.2.6 – DBPEDIA QUERY

This test creates a dbpedia query based on the schema,

Person(occupation, instrument)

This creates the appropriate query based on this schema although it should be noted that the parameters are in a slightly different order in the query that is created in comparison to the expected query.

TEST 5.2.7 – DBPEDIA QUERY

This test creates a dbpedia query based on the schema,

Cave(location)

This creates the appropriate query based on this schema.

TEST 5.2.8 – DBPEDIA QUERY

This test creates a dbpedia query based on the schema,

FormulaOneRacer(races)

This creates the appropriate query based on this schema.

TEST 5.2.9 – DBPEDIA QUERY

This test creates a dbpedia query based on the schema,

River(length)

This creates the appropriate query based on this schema.

TEST 5.2.10 – DBPEDIA QUERY

This test creates a dbpedia query based on the schema,

Royalty(parent)

This creates the appropriate query based on this schema although it should be noted that the parameters are in a slightly different order in the query that is created in comparison to the expected query.

TEST 5.2.11 – DBPEDIA QUERY

This test creates a dbpedia query based on the schema,

river(length)

This creates the appropriate query based on this schema which is an empty string because we cannot make a query from this schema.

TEST 5.2.12 – DBPEDIA QUERY

This test creates a dbpedia query based on the schema,

Stream(length)

This creates the appropriate query based on this schema which is an empty string because we cannot make a query from this schema.

TEST 5.2.13 – DBPEDIA QUERY

This test creates a dbpedia query based on the schema,

River(Mountain(elevation))

This creates the appropriate query based on this schema.

RUN_QUERY_TEST_CASES.JAVA

BASIC FUNCTIONALITY

Responsible for testing *Run_Query.java* to ensure that after creating a query, we can run it without any errors.

After running this test, the results will have been documented in *outputs/testing/Run_Queries_Tests.txt* where you will be able to see the test number, the query that is being run and whether the query has run successfully. If the query has not run successfully then there will be an error message printed.

Note: These tests will be named with a prefix of 6. For example, test 6.1.1, refers to test 1.1 of Run_Query_Test_Cases.java.

EXPECTED RESULTS

TEST 6.1.1 – SEPA QUERY

This test successfully runs a sepa query that queries the *waterBodyPressures* data file.

TEST 6.1.2 – SEPA QUERY

This test does not successfully run the query. This is because the application fails to find the dataset being defined in the query, *water.n3*, which is not in the sepa data files so this is expected.

TEST 6.1.3 – SEPA QUERY

This test successfully runs a sepa query that queries the *waterBodyMeasures* data file.

TEST 6.1.4 – SEPA QUERY

This test successfully runs a sepa query that queries the *waterBodyPressures* data file.

TEST 6.1.5 – SEPA QUERY

This test successfully runs a sepa query that queries the *waterBodyMeasures* data file.

TEST 6.1.6 – SEPA QUERY

This test successfully runs a sepa query that queries the *surfaceWaterBodies* data file.

TEST 6.1.7 – SEPA QUERY

This test successfully runs a sepa query that queries the *bathingWaters* data file.

TEST 6.1.8 – SEPA QUERY

This test successfully runs a sepa query that queries the *surfaceWaterBodies* data file.

TEST 6.1.9 – SEPA QUERY

This test successfully runs a sepa query that queries the *bathingWaters* data file.

TEST 6.1.10 – SEPA QUERY

This test does not successfully run the query. This is because the application fails to find the dataset being defined in the query, *waterBodyTemperatures.n3*, which is not in the sepa data files so this is expected.

TEST 6.2.1 – DBPEDIA QUERY

This test successfully runs a remote dbpedia query.

TEST 6.2.2 – DBPEDIA QUERY

This test successfully runs a remote dbpedia query.

TEST 6.2.3 – DBPEDIA QUERY

This test successfully runs a remote dbpedia query.

TEST 6.2.4 – DBPEDIA QUERY

This test successfully runs a remote dbpedia query.

TEST 6.2.5 – DBPEDIA QUERY

This test successfully runs a remote dbpedia query.

TEST 6.2.6 – DBPEDIA QUERY

This test successfully runs a remote dbpedia query.

TEST 6.2.7 – DBPEDIA QUERY

This test successfully runs a remote dbpedia query.

TEST 6.2.8 – DBPEDIA QUERY

This test successfully runs a remote dbpedia query.

TEST 6.2.9 – DBPEDIA QUERY

This test successfully runs a remote dbpedia query.

TEST 6.2.10 – DBPEDIA QUERY

This test successfully runs a remote dbpedia query.

TEST 6.2.11 – DBPEDIA QUERY

This test does not successfully run the query. This is because the query is an empty string, therefore, an error occurs as soon as the query gets parsed.

TEST 6.2.12 – DBPEDIA QUERY

This test does not successfully run the query. This is because the query is an empty string, therefore, an error occurs as soon as the query gets parsed.

TEST 6.2.13 – DBPEDIA QUERY

This test successfully runs a remote dbpedia query.

TEST 6.2.14 – DBPEDIA QUERY

This test does not successfully run the query. This is because the query is an empty string, therefore, an error occurs as soon as the query gets parsed.

TEST 6.2.15 – DBPEDIA QUERY

This test successfully runs a remote dbpedia query.

SCHEMA_FROM_QUERY_TEST_CASES.JAVA

BASIC FUNCTIONALITY

Responsible for testing *Schema_From_Query.java* to ensure that we can create a valid schema from either a sepa or dbpedia query.

After running this test, the results will have been documented in *outputs/testing/Schema_From_Query_Test.txt* where you will be able to see the test number, the query that is getting a schema extracted from, the expected schema and the actual schema extracted from the query.

Note: These tests will be named with a prefix of 7. For example, test 7.1.1, refers to test 1.1 of Schema_From_Query_Test_Cases.java.

EXPECTED RESULTS

TEST 7.1.1

This test creates a schema from a valid sepa query and returns the correct schema.

TEST 7.1.2

This test creates a schema from a valid sepa query and returns the correct schema.

TEST 7.1.3

This test creates a schema from a valid sepa query and returns the correct schema.

TEST 7.1.4

This test creates a schema from a valid sepa query and returns the correct schema.

TEST 7.1.5

This test creates a schema from a valid sepa query and returns the correct schema.

TEST 7.1.6

This test creates a schema from a valid sepa query and returns the correct schema.

TEST 7.1.7

This test creates a schema from a valid sepa query and returns the correct schema.

TEST 7.1.8

This test creates a schema from a valid sepa query and returns the correct schema.

TEST 7.1.9

This test creates a schema from a valid sepa query and returns the correct schema.

TEST 7.1.10

This test attempts to create a schema from an invalid sepa query (one of the prefixes are missing) and therefore returns null as expected.

TEST 7.2.1

This test creates a schema from a valid dbpedia query and returns the correct schema.

TEST 7.2.2

This test creates a schema from a valid dbpedia query and returns the correct schema.

TEST 7.2.3

This test creates a schema from a valid dbpedia query and returns the correct schema.

TEST 7.2.4

This test creates a schema from a valid dbpedia query and returns the correct schema.

TEST 7.2.5

This test creates a schema from a valid dbpedia query and returns the correct schema.

TEST 7.2.6

This test creates a schema from a valid dbpedia query and returns the correct schema.

TEST 7.2.7

This test creates a schema from a valid dbpedia query and returns the correct schema.

TEST 7.2.8

This test creates a schema from a valid dbpedia query and returns the correct schema.

TEST 7.2.9

This test creates a schema from a valid dbpedia query and returns the correct schema.

TEST 7.2.10

This test creates a schema from a valid dbpedia query and returns the correct schema.

TEST 7.2.11

This test creates a schema from a valid dbpedia query and returns the correct schema. Note that the query passed in is an empty string, therefore, the schema should be an empty string.

TEST 7.2.12

This test creates a schema from a valid dbpedia query and returns the correct schema.

TEST 7.2.13

This test attempts to create a schema from an invalid dbpedia query (one of the prefixes are missing) and therefore returns null as expected.

RUN_CHAIN_TEST_CASES.JAVA

BASIC FUNCTIONALITY

Responsible for testing *Run_CHAIn.java* to ensure that we can run CHAIn given several different scenarios such as an invalid query or several target schemas to match against the schema extracted from the original query.

After running this test, the results will have been documented in *outputs/testing/Run_CHAIn_Tests.txt* where you will be able to see the test number, the original query that CHAIn has been called with including whether it successfully runs and the steps that CHAIn takes depending on this including any new queries produced and whether they are successful.

Note: These tests will be named with a prefix of 8. For example, test 8.1, refers to test 1 of Run_CHAIn_Test_Cases.java.

EXPECTED RESULTS

TEST 8.1

This test takes in a basic sepa query that doesn't return any results, which after passing into CHAIn results in a new query being created that runs successfully.

TEST 8.2

This test takes in a sepa query; however, this query is invalid meaning that CHAIn terminates immediately after trying to run it.

TEST 8.3

This test takes in a basic sepa query that returns no results, however, after attempting to repair and create new queries we get zero matches from SPSM meaning that no new queries are created.

TEST 8.4

This test takes in a basic dbpedia query that initially returns no results, after running CHAIn with this query we now have a new query that does run successfully and returns results.

TEST 8.5

This test takes in a basic dbpedia query that already runs successfully, meaning that CHAIn terminates immediately after running this initial query.

TEST 8.6

This test takes in a dbpedia query that returns no results, after running CHAIn with this query we now have two new queries where only one of these queries returns results.