Intelligent Web Application with Semantic Technology

Magic Meal Maker

Group 33
Claire Kuhlkin, Ka-Ho Poon, Marina Santos and Sanne van den Berg

Contents:

Application Design & Reuse

Application Goal

The Magic Meal Maker application aims to make it easier for users to plan out their daily meals using ingredients they already have. It requires the user to input ingredients they have accessible and gives them different meal options based on the provided ingredients. Users also have the option to choose whether they want a specific type of meal diet. For example, a user is able to specify that they want a Vegan meal. The application will then inference over the available web data using the ingredients the user provided to pick out an appropriate meal. After finding appropriate meals given the user's data, the application will list the meals found.

Users

The application is intended for every person who cooks at home. It could be very useful to those who do not have much time to spend on meal planning because it is quick and easy to find meals using the application. Moreover, it is also useful for people that want to know an estimate of the number of calories a meal has, since this is included in the results. Overall, the application can be used by people in ranging age groups, from a university student that wants to find out what they can make with what is left in the fridge, to a family that wants to discover new recipes with the ingredients that they have available. Furthermore, the application includes meals and recipes that belong to various cultures around the world, hence, it can be used by natives from any culture or users that are willing to try something new.

Application Design

The web application is designed to be visually pleasing, easy to navigate and easily accessible. The user is presented with two fields they can fill in. In these fields, the user indicates what ingredients they have available. A text field will be present that gives suggestions while the user types in an ingredient, thus letting it auto-complete what the user is typing. Making it possible for the user to type in their ingredients makes the application more organized and more visually appealing than presenting the user with a drop-down menu with all the possible ingredients, which would result in an overwhelming amount of options. The dietary restriction option is presented as a drop-down menu, consisting of five different types of meals. The user can select an option from this list, and if they do not want a specific type of meal they can select no option. Lastly the user has the option to select whether they want a meal for breakfast, dinner or lunch, also presented through a drop-down menu.

A scenario could be that the user inputs the ingredients "meat" and "rice" in their respective ingredients field, then selects the type of meal "Meat", and finally selects "Dinner". As a result, the user will receive four meals they can choose from: Arroz Junto, Meatballs with rice, İskilip dolması and Bibimbap.

User Interaction

When the user opens the web application, they will be greeted with three text boxes, each with instructions on how to fill in the information needed to retrieve the meals they can prepare. The first box contains two mandatory input fields that both have the placeholder "Ingredient name" so that the user knows what information to fill in. After completing the inputs of the first text box, the user will go to the second box, where they can select whether they want their meal to be vegan, vegetarian, made with meat, dairy-free or pescetarian. These choices are presented through a drop-down menu. If no option is selected, the final results will thus not be restricted. Lastly, in the third box, the user selects what type of meal they want. A meal for breakfast, lunch or dinner, also presented through a drop-down menu. After selecting this the user clicks a button labelled "Show meals", and will receive the possible meals that they can prepare. Not only will these meals include the ingredients the user gave as input and the remaining ingredients that are needed to make the meal, the number of calories per measurement of each ingredient will also be shown. In order to put in new ingredients and load a new number of meals, the page has to be reloaded.

Ontology Reuse

To make the final ontology two ontologies were reused, both related to food. The first one is similar to the base ontology, which will be explained later in the report, and contains a number of classes related to food and ingredients, of which a few are the same as the base ontology. Besides these there are classes that were not included, resulting in a useful addition. It also has some instances that are not defined as meat sources in the external data sets that are used, allowing more different kinds of meals to be inferred. The second ontology that was used is an ontology based on recipes, also containing a number of classes related to food and ingredients. The classes about recipes were not useful for the ontology but the food classes did contain some new and useful instances.

External Sources of Data

For the web application, two external data sources were used to create the necessary instances for the final ontology. First, a nutritional dataset from Kaggle that contains information about common foods was used for creating some of the ingredients instances of the final ontology. This dataset was chosen because it has a lot of instances, and can be used to give the user some additional information about the ingredients such as measurements and calories. Secondly, Wikidata was used to query as many meals as possible and their ingredients. Using GraphDB these instances were added to the ontology. Since the ingredients that are given as input may not be all the ingredients needed to make a certain meal, Wikidata is used in order to present the remaining instances to the user.

Domain Modelling & Data Integration

Domain & Scope

The domain of the application's ontology is food, and the application is to be a meal recommendation tool. Possible questions that can be answered with this application are "what can I make with these ingredients?" or "how many calories does this ingredient contain?". This ontology contains classes related to meal types, and properties that show what ingredients the meals have. The ontology has multiple purposes for the application. With the ontology, it will be possible to inference the meals that can be made with the ingredients that the user gives as input. Afterwards, the ingredients that are needed for the possible meal, but not given as input can be inferred with the ontology. As mentioned before, the ontology will contain a significant amount of information about food. Besides answering questions such as what ingredients do certain meals have, the number of calories for each ingredient can be inferred if possible.

If this ontology were to be published, people who are interested in weight loss or cooking can use this ontology. This is because there will be a large number of meals available, showing the number of calories and the ingredients to make them.

Methodology

For the construction of the ontology, a bottom-up approach was used. An ontology from Kaggle was reused, containing ingredients and meals that are useful for our web application. Some of the relevant terms that are used in our ontology are recipes, meals, and ingredients, these encompass the created or reused classes. For property names, examples like hasIngredient and isPartOfMeal were used. The taxonomic hierarchy used is based on meals, hence, "Food" is the top subclass, then "Ingredients" and "Meals". Other classes were created for the purpose of separating the meals and the ingredients for the user, for example, "Vegetarian" exists both under "Ingredients" and "Meals", in order to identify vegetarian ingredients and meals respectively. Moreover, the properties mentioned previously were defined with their respective domain and ranges, for example, the property "hasIngredient" has domain "Meal" and range "Ingredient". In order to define the classes, in this ontology it is clear that most of the classes have a global domain and range, as the classes are very descriptive of what they contain. Additionally, classes like "Ingredients" and "Meals" are disjoint with one another. Finally, instances were created for every class, a large majority of them were imported from data structures that we reused. At the end, we checked for anomalies and none were found and then imported the two ontologies.

Conceptualization

The domain of our ontology is food, and it was visualised using a number of classes and properties, see Figure 1. The model consists of the main class Food and all the subclasses that fall under that. In the case of our model those classes are Ingredients and Meal. The first one, as the name indicates, consists of all the individual ingredients that can be part of a meal. The second one consists of different types of meals.

Like the class Food, the subclass Ingredients also has a number of subclasses. These are Egg, Fish, Fruit, Grain, Meat, Nut, Dairy and Vegetable. They represent different categories that ingredients can be part of. For example, the class Fish could contain salmon and the class Meat could contain chicken. Furthermore, the subclass Meal also contains subclasses. These are Breakfast, Lunch, Dinner, Pasta, Salad, ContainsDairy, ContainsMeat, Pescaterian, Vegan and Vegetarian. They represent the different types of meals there are. So for instance, the class Pasta contains meals that have any type of pasta as an ingredient, the class ContainsDairy consists of meals that have dairy in them and the class Pescatarian contains meals that are made with some type of fish.

To relate individuals to other individuals or data, seven properties were created: hasIngredient, hasVariation, isPartOfMeal, hasCalories, hasCategory, hasMeasurement, rdfs:label and isPerishable. The property isPartOfMeal is the inverse of the property hasIngredient.

Ontology Description

The base of this ontology is described in the Conceptualization section, and it was extended with two existing ontologies. The first ontology is also an ontology on food and it has valuable addition. It contains a number of classes that were not included in the base ontology, for example the subclass HerbAndSpice and FatAndOil. There were also a number of subclasses that were the same as the classes we came up with in the base ontology like Fruit, Vegetable and Nut. These contained a lot of beneficial instances the initial ontology did not have, and the classes were aligned by adding a necessary and sufficient condition.
It also had some classes that could be defined as a subclass of an already defined subclass. The classes DryPasta and FreshPasta for example, are a subclass of Pasta.

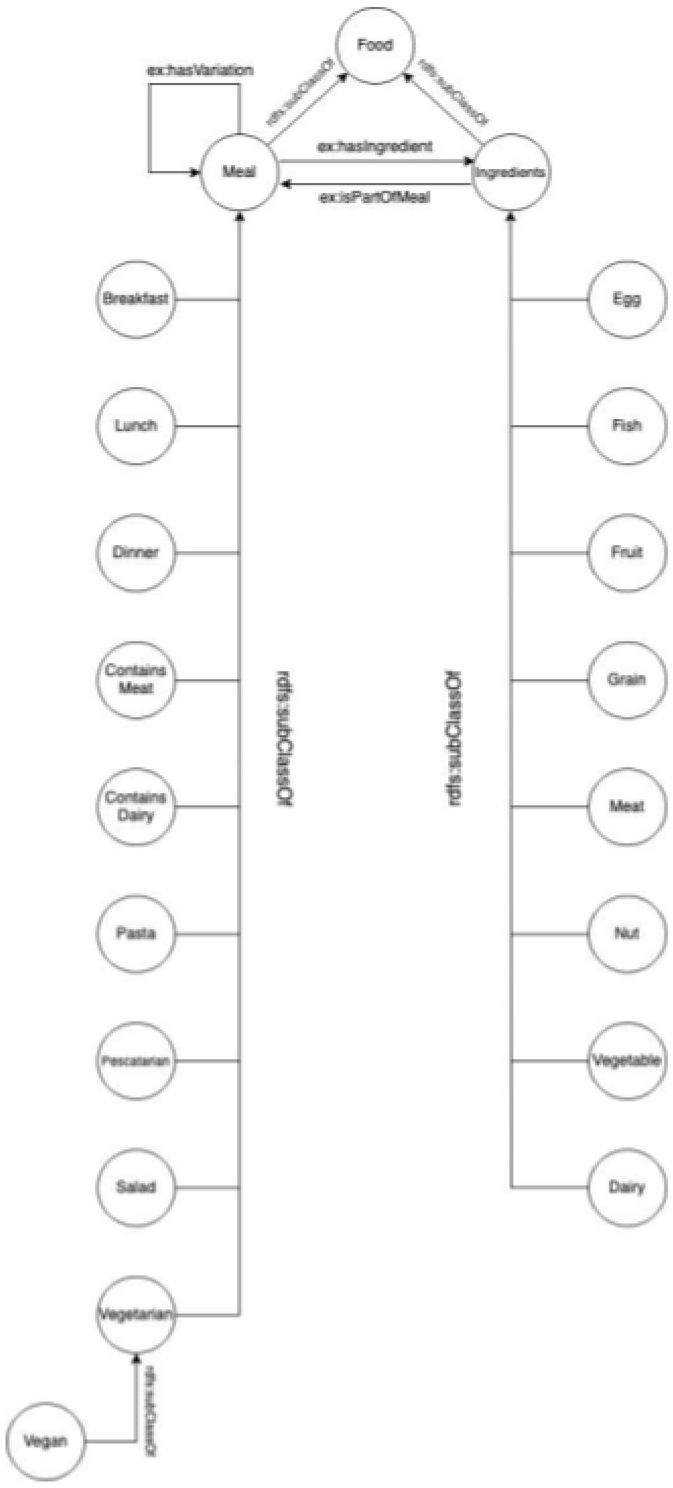


Figure 1: Depicting the relations between classes

The second ontology that was added was an ontology focused on recipes. The classes were not of much use to the ontology, but it contained a lot of useful instances. These can be used to produce significant inferences.

Furthermore, there were a couple class restrictions added in order to create more inferences. These were placed on the classes Dairy, FatAndOil, Fruit, Vegetable, Meat, ContainsMeat and Pescatarian. For ContainsMeat it is defined as a necessary and sufficient restriction, where "hasIngredient some Meat". This means that everything that is an instance of the class ContainsMeat, contains some type of meat. With this we can define whether a meal is vegetarian or not. The same concept was applied to the class Pescatarian, with the necessary and sufficient restriction "hasIngredient some Fish". Everything that is an instance of the class Pescatarian, contains some type of fish.

The rest of the restrictions were used to link instances from the Kaggle file to the existing classes. Each ingredient from this file has a label specifying in what category it belongs, for example milk hasCategory "Dairy products". It was then said that the class Dairy has a necessary and sufficient restriction where hasCategory value "Dairy products", making all the individuals with this category an instance of the class Dairy. The same is done for the remaining categories.

Integrating External Datasets

In this ontology two external datasets were integrated. One is a file from Kaggle, containing a number of ingredients and their calories per indicated measurement. These ingredients are used as instances in our ontology. The original file also contained the columns grams, protein, fat, saturated fat, fiber and carbs but these were removed as they do not have any addition. The second external dataset that was used was Wikidata. From here as many meals as possible and their ingredients were extracted.

In order to insert these two sets of instances into the ontology, GraphDB was used. First the CSV file needed to be converted to RDF. This was done with RDF mapping in GraphDB. Here there were also a number of things stated which allowed us to map between the instances from the external dataset and the ontology. With GraphDB this was transformed to a SPARQL query and with this the data from Wikidata could be extracted. In order to map the data from Wikidata to the ontology, a query was made. With this query the ingredients were first filtered so that we would only have the ingredients from Wikidata that have the same label as the ingredients from Kaggle. After this the remaining ingredients from Wikidata were also inserted.

In order to map the categories from Kaggle to the classes of the ontology, we said that the class has a necessary and sufficient restriction on the property hasCategory. So for example, the class Dairy is equivalent to hasCategory value "Dairy products". This way a lot of inferences were inferred.

Description of Inferences

As you can see in Figure 2, located in the appendix, it was stated that the class Meat has the restriction hasCategory value "Meat, Poultry". The object refers to the instances from the Kaggle file that are of this category, and infers all the different types of meat. Since the class

ContainsMeat has the restriction hasIngredient some Meat, as can be seen in Figure 3, all the meals that have meat in them are inferred, including meals from Wikidata. These inferences are crucial to our ontology as our application gives the option for the user to decide what type of diet restriction they want. Therefore, we need to be able to inference which meals contain meat to be able to present appropriate results for the diet restrictions "Vegan" and "Vegetarian" to the user.
The same thing was done for the class Pescatarian, see Figure 4, stating it has the restriction hasIngredient some Fish, allowing the user to select the Pescatarian option from the dietary restriction field. More importantly, these inferences make it so that the ontology and, as a result, the application can be created efficiently and effortlessly, as there is no need to specify each instance, it is possible to do it in groups.

Description of SPARQL Queries

In order to have meaningful results over the integrated data and the ontology, multiple SPARQL queries were made. The main query takes the ingredients the user gave as input and returns the meals they can make. First the variable meal is defined, which is of type ex:Meal. It is related to three other variables with the predicate ex:hasIngredient. These variables all have their own rdfs:label. The user input is taken and used as the label for the first two variables. The third variable retrieves the rest of the needed ingredients to make the meal.

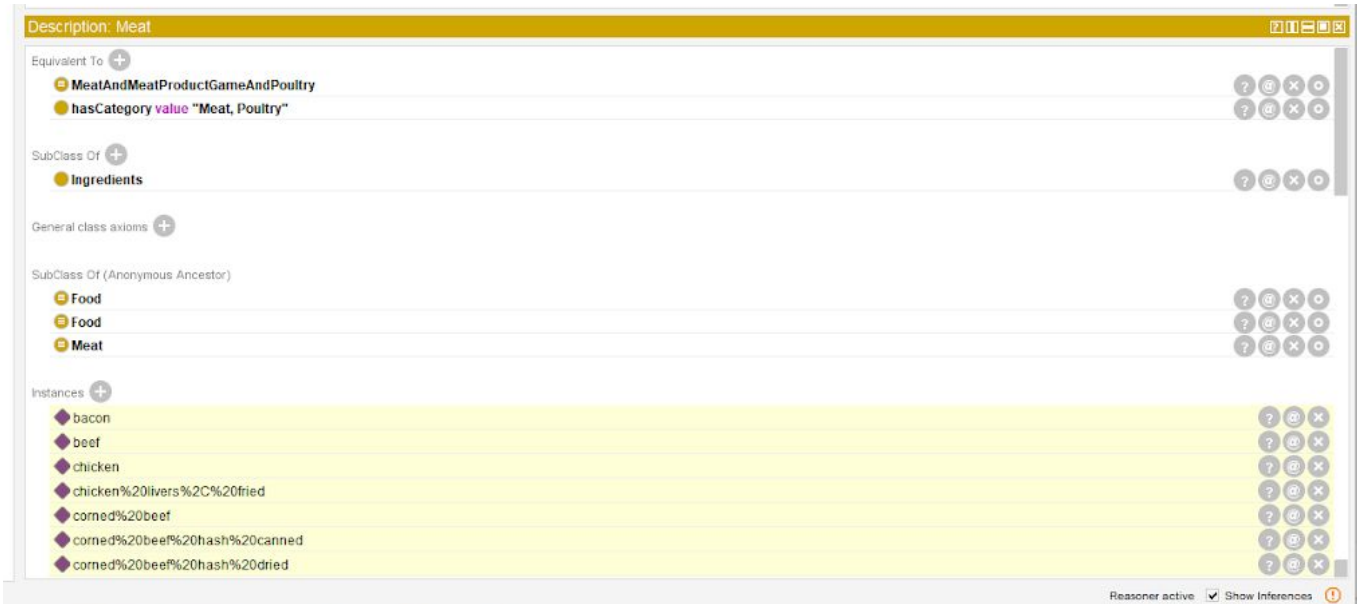
If the calories and measurement of an ingredient are specified in the Kaggle file, these will also be shown in the results. Since not every ingredient has these, an OPTIONAL statement is used.

When the user selects a dietary restriction, a new line will be added to the query, depending on what the user selected. For example, if the user selects the option Vegetarian, two lines will be added, the first one stating that all meals of type ContainsMeat are filtered out and the second stating that all meals of type Pescatarian are filtered out. This will result in meals that do not have meat or fish in them, thus being vegetarian.

Evidence for SPARQL Queries

As shown in Figures 5 and 6 located in the appendix, if the ontology is asked to present all meals of rdf:type ex:ContainsMeat, which are all the meals that contain meat, with the reasoner turned off, there are no results, as all of our results are inferred. Nevertheless, if reasoning is turned on, more than 500 results are shown, due to the results coming solely from inferencing. This is due to importing the instances from external data, which had their own respective classes for meat, and as ex:ContainsMeat is used in the SPARQL query, which is our own class, it does not produce any results. To be able to obtain these results, we added class restrictions such as equivalent class and subclassOf, as mentioned previously. An example of the inferences produced is "roast beef" "beef" "3 oz." "245". This is showing that the meal "roast beef" has the ingredient "beef", the measurement is "3 oz." and the calories for "beef" is "245" kcal.

Appendix


Description: Meat

Equivalent To
MeatAndMeatProductGameAndPoultry
hasCategory value "Meat, Poultry"

SubClass Of
Ingredients

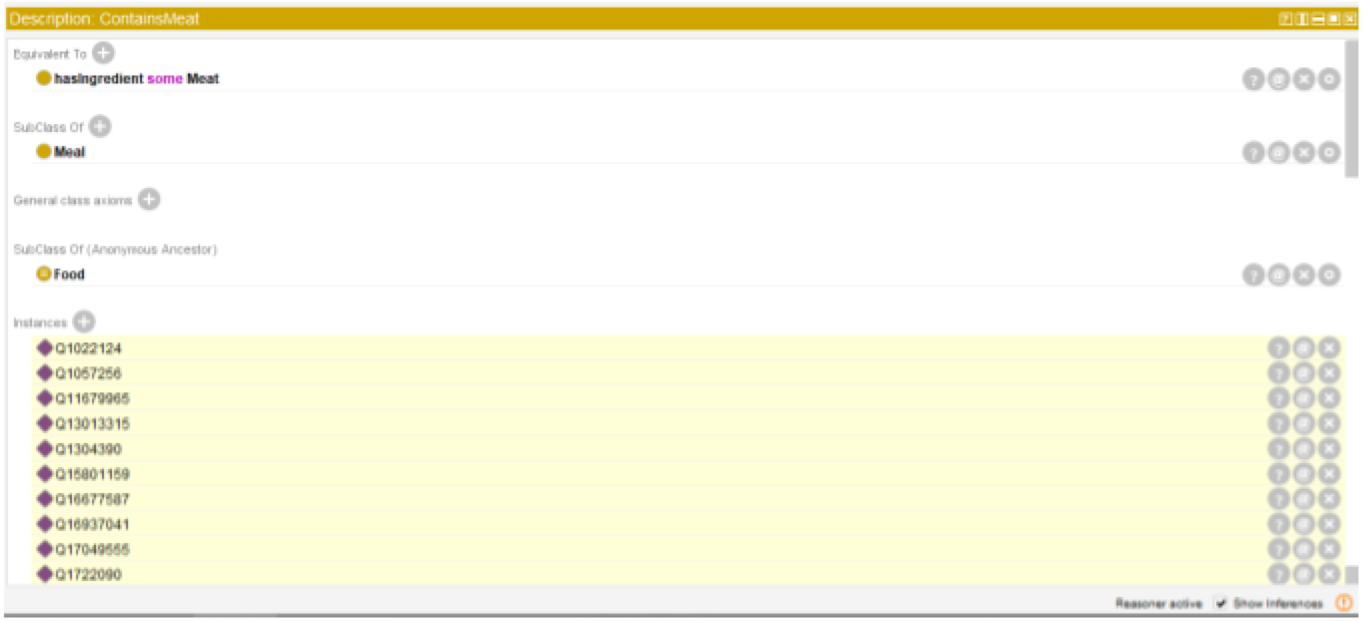
General class axioms

SubClass Of (Anonymous Ancestor)
Food
Food
Meat

Instances
bacon
beef
chicken
chicken%20livers%2C%20fried
corned%20beef
corned%20beef%20hash%20canned
corned%20beef%20hash%20dried

Reasoner active ✓ Show Inferences

Figure 2: showing the inferences of instances of the class Meat.


Description: ContainsMeat

Equivalent To
hasIngredient some Meat

SubClass Of
Meal

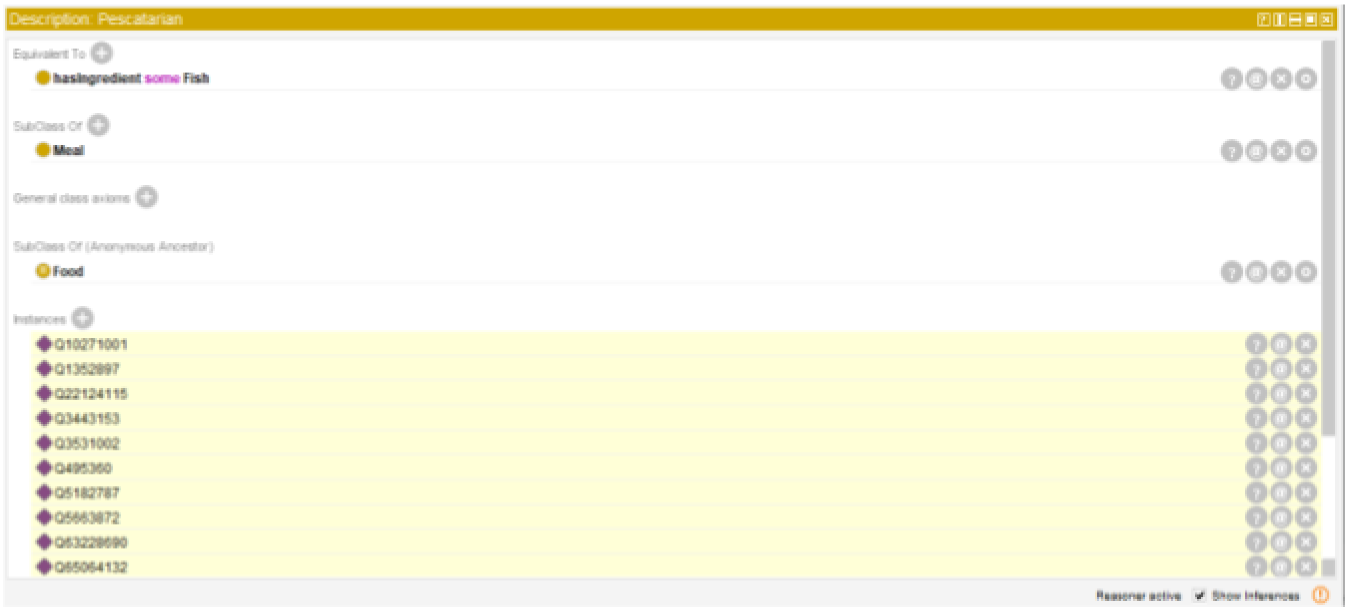
General class axioms

SubClass Of (Anonymous Ancestor)
Food

Instances
Q1022124
Q1057256
Q11679965
Q13013315
Q1304390
Q15801159
Q16677587
Q16937041
Q17049555
Q1722090

Reasoner active ✓ Show Inferences

Figure 3: showing the inferences from Wikidata of class "Meat"



Description: Pescatarian

Equivalent To
hasIngredient some Fish

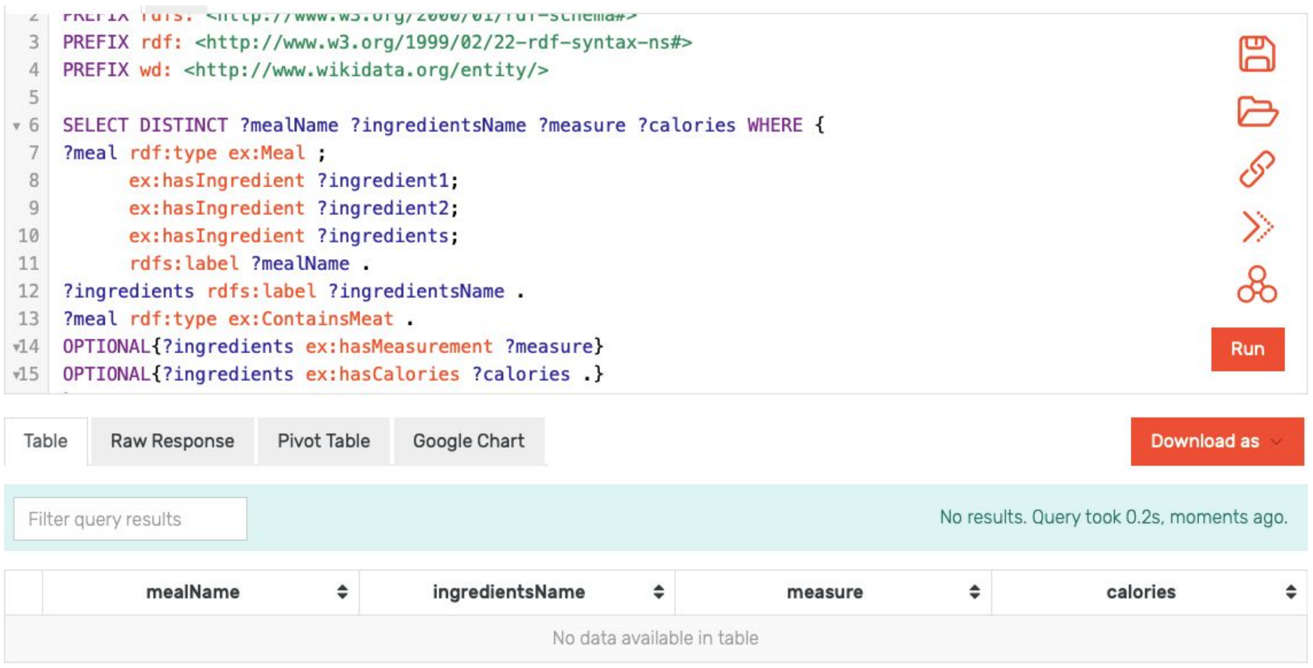
SubClass Of
Meal

General class axioms

SubClass Of (Anonymous Ancestor)
Food

Instances
Q10271001
Q1352897
Q22124115
Q3443153
Q3531002
Q495360
Q5182787
Q5663872
Q63228690
Q65064132

Figure 4: showing inferences of instances from Wikidata to the class "Fish"



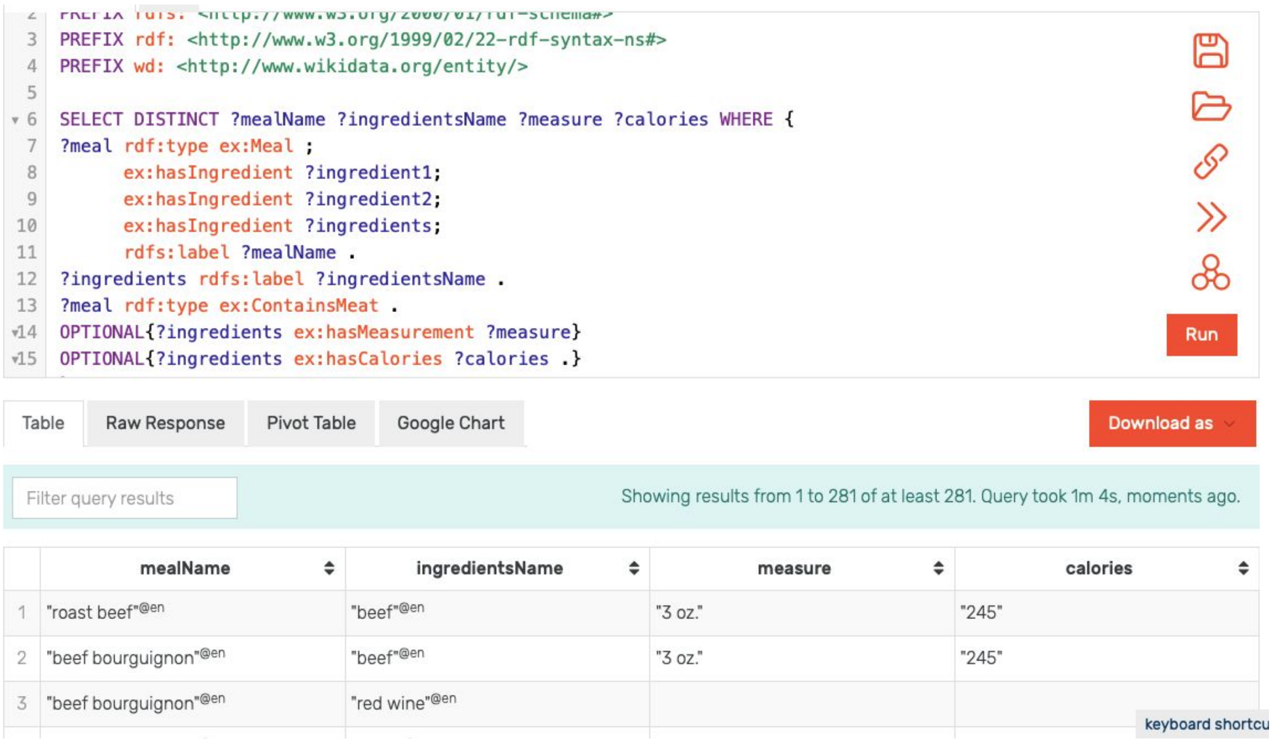
```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX wd: <http://www.wikidata.org/entity/>

SELECT DISTINCT ?mealName ?ingredientsName ?measure ?calories WHERE {
?meal rdf:type ex:Meal ;
      ex:hasIngredient ?ingredient1;
      ex:hasIngredient ?ingredient2;
      ex:hasIngredient ?ingredients;
      rdfs:label ?mealName .
?ingredients rdfs:label ?ingredientsName .
?meal rdf:type ex:ContainsMeat .
OPTIONAL{?ingredients ex:hasMeasurement ?measure}
OPTIONAL{?ingredients ex:hasCalories ?calories .}
```

No results. Query took 0.2s, moments ago.

	mealName	ingredientsName	measure	calories
	No data available in table			

Figure 5: showing that with reasoning off, there are no results



```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX wd: <http://www.wikidata.org/entity/>

SELECT DISTINCT ?mealName ?ingredientsName ?measure ?calories WHERE {
?meal rdf:type ex:Meal ;
      ex:hasIngredient ?ingredient1;
      ex:hasIngredient ?ingredient2;
      ex:hasIngredient ?ingredients;
      rdfs:label ?mealName .
?ingredients rdfs:label ?ingredientsName .
?meal rdf:type ex:ContainsMeat .
OPTIONAL{?ingredients ex:hasMeasurement ?measure}
OPTIONAL{?ingredients ex:hasCalories ?calories .}
```

Showing results from 1 to 281 of at least 281. Query took 1m 4s, moments ago.

	mealName	ingredientsName	measure	calories
1	"roast beef"@en	"beef"@en	"3 oz."	"245"
2	"beef bourguignon"@en	"beef"@en	"3 oz."	"245"
3	"beef bourguignon"@en	"red wine"@en		

Figure 6: showing that with reasoning, there are 597 results.