Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

CS7CS4 Knowledge and Data Engineering

Individual Project – Flight Connectivity Knowledge Graph

Shane O'Donnell – 21364002

Contents

Introduction

The goal of this project was to create a flight connectivity knowledge graph that models airports, routes, and the possible landmarks to visit whilst in the city associated with the airport. This was done by mapping data from csv files to RDF, creating an ontology connect lifted data and apply cardinality constraints and OWL properties, then validating structures using SHACL and utilizing SPARQL to query the graph. As well as this the Landmark, LandmarkType, and Continent data was mapped manually to demonstrate the ability to do so. The tools used for this project included “rmlmapper-java” to convert rml uplifted data to rdf, CHOWLK to make a visual diagram of the ontology and convert to turtle, and SHACL Playground to familiarise myself with SHACL validation.

Method

Data Source and Pre-Processing

In order to acquire data for this knowledge graph, online databases were found which included data on the physical location of airports around the world and data including real flight routes operated by each airport. The first dataset used was an airport database provided by openflights, this dataset included data on over 10,000 airports which included each airports, ID, name, city, country, IATA code, ICAO code, location in latitude and longitude, altitude, timezone, Daylight saving time, Type (airport, port, station), and source. For the project scope the data which was of interest was airport ID, IATA, ICAO, Latitude, Longitude.

Another database utilised was a flight route database offered from Kaggle. This database included the following rows of data seen in figure 1. This database offered the data necessary to map the connection between airports to the database of airports outline above; this would be done in the RML mapping stage using “source airport”, “destination airport” and the “Airport ID”. Like the airport dataset, some of the data included was decided to not be relevant and was no included in the knowledge graph, such as equipment and codeshare.

The last data source was demonstrating manual mapping for Landmarks, Landmark types (Museum, Monument, Bridge etc..) and the list of continents and their connection to the mapped countries.

Airport ID	Unique OpenFlights identifier for this airport.
Name	Name of airport. May or may not contain the City name.
City	Main city served by airport. May be spelled differently from Name.
Country	Country or territory where airport is located. See Countries to cross-reference to ISO 3166-1 codes.
IATA	3-letter IATA code. Null if not assigned/unknown.
ICAO	4-letter ICAO code. Null if not assigned.
Latitude	Decimal degrees, usually to six significant digits. Negative is South, positive is North.
Longitude	Decimal degrees, usually to six significant digits. Negative is West, positive is East.
Altitude	In feet.
Timezone	
Type	Type of the airport. Value "airport" for air terminals, "station" for train stations, "port" for ferry terminals and "unknown" if not known. In airports.csv, only type=airport is included.
Source	Source of this data. "OurAirports" for data sourced from OurAirports, "Legacy" for old data not matched to OurAirports

Figure 1: Data included in Airports.csv

Airline	Airline 2-letter (IATA) or 3-letter (ICAO) code of the airline
Airline ID	Airline ID Unique OpenFlights identifier for airline (see Airline).
Source airport	Source airport 3-letter (IATA) or 4-letter (ICAO) code of the source airport.
Source airport ID	Source airport ID Unique OpenFlights identifier for source airport (see Airport)
Destination airport	Destination airport 3-letter (IATA) or 4-letter (ICAO) code of the destination airport.

Destination airport ID	Destination airport ID Unique OpenFlights identifier for destination airport (see Airport)
Codeshare	Codeshare "Y" if this flight is a codeshare (that is, not operated by Airline, but another carrier), empty otherwise.
Stops	Stops Number of stops on this flight ("0" for direct)

Figure 2: Data included in routes.csv

RML Mapping

CSV files

In order to lift data from the csv files rml mappring turtle files were created in order to create the initial connections and meaning between data points. Figure 2 outlines the classes and connections created between classes from the csv files data. For the csv files the rml mapping turtles files were ran in rmlmapper.jar [1] which converted the desired mapping outlined in the turtles file to rdf.

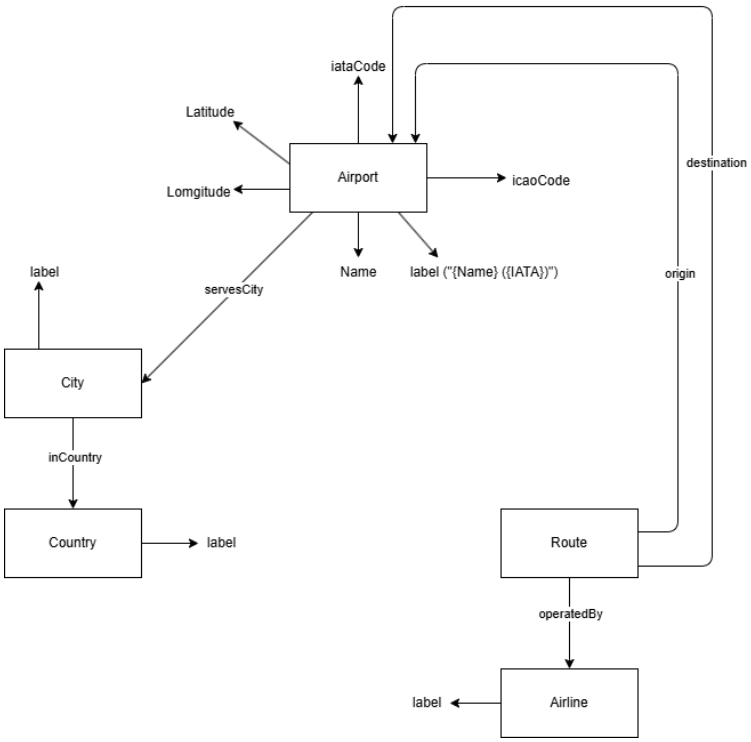

Figure 3: Connections created from uplifted data

Airports CSV

Due to processing times incurred when using the full dataset, in order to create a graph which demonstrates the desired functionality in the given timeframe the dataset was limited to airports which reside in Ireland, United Kingdom, Germany, Spain, France, and Portugal. This was done by filtering the CSV to only include the countries listed above.

From here it was possible to begin lifting the data and converting to rdf. Inside the rml lifting file the csv was set as the source and Unique URI's were made for the class Airport, Countries, and Cities. The airport ID was used to uniquely identify each airport (http://example.org/flight/airport/{Airport ID}) and from here the predicates, iataCode, icaoCode, airportName, latitude, longitude and label were initialised; from here the Airport ID would act as the subject, the predicates outlined above

acting as the predicates, and their objects would be take from their respected rows in the CSV files. An example of this can be seen in figure 3 which outlines the predicate declaration and object definition for the iataCode.

```
#IATA code (identifier for each airport)
rr:predicateObjectMap [
    rr:predicate ex:iataCode ;
    rr:objectMap [rml:reference "IATA"]
];
```

Figure 4: IATA Code mapping in rml mapping file

The countries triple map followed the same format by identifying the CSV file as the source and a subject map to define a URI yemplate for each country instance. Each country URI is then typed with ex:Country class and given a label.

For the city class the same behaviour is mirrored by declaring the object as a unique URI, although this time the city URI is structured as "{City}_{Country}". This was done to avoid clashes to avoid overlapping city names in the world, an example of this in the world is Paris in France and Paris in Texas, United States. From here the predicate "label" connects the subject to the object "City_Country". As well as this the country class is connected to the city class by declaring the predicate "inCountry". This can be seen in figure 4, the "parentTripleMap" points to the countries triple map, whatever IRI that map creates in its "rr:subjectMap" for a given country row will be used as the object of this predicate. The "rr:joinCondition" declares the City class CSV row "Country" as the child and the column "Country" in the Country class CSV as the parent. Even though both maps read from the same file they are independent logical views and link the city to that countries IRI. As well at this the predicate cityName was created to create a unique connection for each city URI and creating an object which is the cities name.

```
# City -> Country via JOIN on the Country column
rr:predicateObjectMap [
    rr:predicate   ex:inCountry ;
    rr:objectMap [
        rr:parentTriplesMap <#Countries> ;
        rr:joinCondition [ rr:child "Country" ; rr:parent "Country" ]
    ]
] .
```

Figure 5: JoinCondition for city and country class

The same logic is applied to create the connection between cities and the airports which reside within them by creating the predicate "servesCity" in the airport Class and finding the corresponding city for each airport using the joinCondition statement.

Routes.csv Mapping

To begin mapping the data, the same filtering process was completed by only including routes between Ireland, United Kingdom, Spain, Portugal, France and the Netherlands. From here the Airline triple map was defined as well as the class "Airline". From here a unique URI was created for each airline as "rr:template http://example.org/flight/airline/{airline ID} ". From here the predicates, label

was created to point to the object “airline” which outlines the full name of the airline rather than just the Airline ID.

In order to connect the data in the two csv files, the airport class was defined in this mapping file as well and the csv for the airport data was set as the source. The connection between routes and airports was created by using the predicates, “origin” and “destination” which connected the “source airport id”, and “destination airport id” columns data in the routes CSV to the matching the “Airport ID” in the airport CSV.

A connection from the Routes class to the Airline class was also created using the predicate “operatedBy” which used the joinCondition to connect the airline ID for each flight route outlined in the csv file.

Finally the predicate “stops” was defined which outlines how many stops the route has from origin to destination.

```
#origin airport - join the airport id(airports.csv) to source_airport_id(flights.csv)
#from here the destination connections can be mapped through the
rr:predicateObjectMap [
    rr:predicate ex:origin ;
    rr:objectMap [
        rr:parentTriplesMap <#Airports> ;
        rr:joinCondition [rr:child "source airport id" ; rr:parent "Airport ID"]
    ]
];

# add the destination node to the airport ID
rr:predicateObjectMap [
    rr:predicate ex:destination ;
    rr:objectMap [
        rr:parentTriplesMap <#Airports> ;
        rr:joinCondition [rr:child "destination airport id" ; rr:parent "Airport ID"]
    ]
];
```

Figure 6: joinCondition's for origin and destination connection to corresponding airports

Generated RDF

From the rml mapping to both csv files and utilising rmlmapper.jar, two rdf files were generated for each csv file mapping. To examples of snippets from each rdf file can be seen below which demonstrates how a singular airport is mapped and the nodes connected to each airport ID URI. As well as this the routes mapping can be seen which defines the URI for each route and the connected nodes outlining the destination, origin, and how many stops are on this route.

```
<http://example.org/flight/airport/10723> a ex:Airport;
    ex:airportName "Elsenthal Grafe Airport";
    ex:icaoCode "EDNF";
    ex:latitude 48.822498;
    ex:longitude 13.3675;
    ex:servesCity <http://example.org/flight/city/Elsenthal_Germany> .

<http://example.org/flight/airport/10744> a ex:Airport;
    ex:airportName "Newtownards Airport";
    ex:icaoCode "EGAD";
    ex:latitude 54.5811004639;
    ex:longitude -5.69193983078;
    ex:servesCity <http://example.org/flight/city/Newtownards_United%20Kingdom> .
```

Figure 7: RDF data from Airport.csv mapping

```
<http://example.org/flight/route/600_492_4296> a ex:Route;
  ex:destination <http://example.org/flight/airport/492>;
  ex:operatedBy <http://example.org/flight/airline/4296>;
  ex:origin <http://example.org/flight/airport/600>;
  ex:stops 0 .

<http://example.org/flight/route/600_502_837> a ex:Route;
  ex:destination <http://example.org/flight/airport/502>;
  ex:operatedBy <http://example.org/flight/airline/837>;
  ex:origin <http://example.org/flight/airport/600>;
  ex:stops 0 .
```

Figure 8: RDF data from routes.csv mapping

Landmark Mapping

The landmark data was mapped manually to enrich the graph and make it possible to query the graph to return the connection between airports, cities and the landmarks present in these cities. Ex:Landmark represents individual points of cultural or structural significance while ex:LandmarkType categorises the landmarks according the nature of the landmark (ex. Museum, monument, bridge). The landmark types are outlines below in figure 6.

```
######## Landmark Types (once) ########
ex:Museum a ex:LandmarkType ; rdfs:label "Museum" .
ex:Monument a ex:LandmarkType ; rdfs:label "Monument" .
ex:Bridge a ex:LandmarkType ; rdfs:label "Bridge" .
ex:ReligiousSite a ex:LandmarkType ; rdfs:label "Religious Site" .
ex:Square a ex:LandmarkType ; rdfs:label "Square" .
ex:District a ex:LandmarkType ; rdfs:label "District" .
ex:Park a ex:LandmarkType ; rdfs:label "Park" .
ex:Architecture a ex:LandmarkType ; rdfs:label "Architecture" .
```

Figure 9: Definition of Types of Landmarks

Then for each country and city, the landmarks were connected to the city they reside in using the ex:hasLandmark property. From here the ex:hasLandmarkType property was used to connect each landmark to the category to describe it. This can be seen in figure 7 which is maps the landmarks and the type of landmarks for Dublin Ireland.

```
<http://example.org/flight/city/Dublin_Ireland> ex:hasLandmark
  ex:guinness_storehouse, ex:temple_bar, ex:molly_malone_statue .

ex:guinness_storehouse a ex:Landmark ; rdfs:label "Guinness Storehouse" ; ex:hasLandmarkType ex:Museum .
ex:temple_bar a ex:Landmark ; rdfs:label "Temple Bar" ; ex:hasLandmarkType ex:District .
ex:molly_malone_statue a ex:Landmark ; rdfs:label "Molly Malone Statue" ; ex:hasLandmarkType ex:Monument .
```

Figure 10: Mapping Cities to Landmarks and the type of Landmark each Landmark corresponds with

Continent Mapping

The connection between continents and countries was also manually mapped by linking each country to a singular continent. The continents were defined as seen in figure 8.

```
ex:northAmerica a ex:Continent .
ex:southAmerica a ex:Continent .
ex:europe a ex:Continent .
ex:africa a ex:Continent .
ex:asia a ex:Continent .
ex:oceania a ex:Continent .
ex:antarctica a ex:Continent .
```

Figure 11: Continent Definition

From here it was possible to connect each country present in the knowledge graph using the ex:inContinent property, this can be seen in figure 9.

```
<http://example.org/flight/country/France> ex:inContinent ex:europe .

<http://example.org/flight/country/Spain> ex:inContinent ex:europe .

<http://example.org/flight/country/United%20Kingdom> ex:inContinent ex:europe .

<http://example.org/flight/country/Ireland> ex:inContinent ex:europe .

<http://example.org/flight/country/Portugal> ex:inContinent ex:europe .
<http://example.org/flight/country/Germany> ex:inContinent ex:europe .
<http://example.org/flight/country/Netherlands> ex:inContinent ex:europe .
```

Figure 12: Connection between countries and continents

Ontology Design

The ontology contains 8 core classes which are outlined below. The ontology was modelled by creating a drawio diagram of the connection between classes to offer a visual reference for the design. Then, using the CHOWLK tool, the drawio file was converted into the TTL code needed for the ontology.

Class
Airport
City
Country
Airline
Route
Landmark
Landmark Type
Continent

Figure 13: Classes in ontology

To visualise the ontology the drawio diagram can be seen below in figure 11. The diagram models the connection between classes as well as the owl object properties used to add further meaning to the graph, the object properties used include, “inverseOf”, “subPropertyOf”, “symmetricProperty”, “TransitiveProperty” and “propertyChainAxiom”.

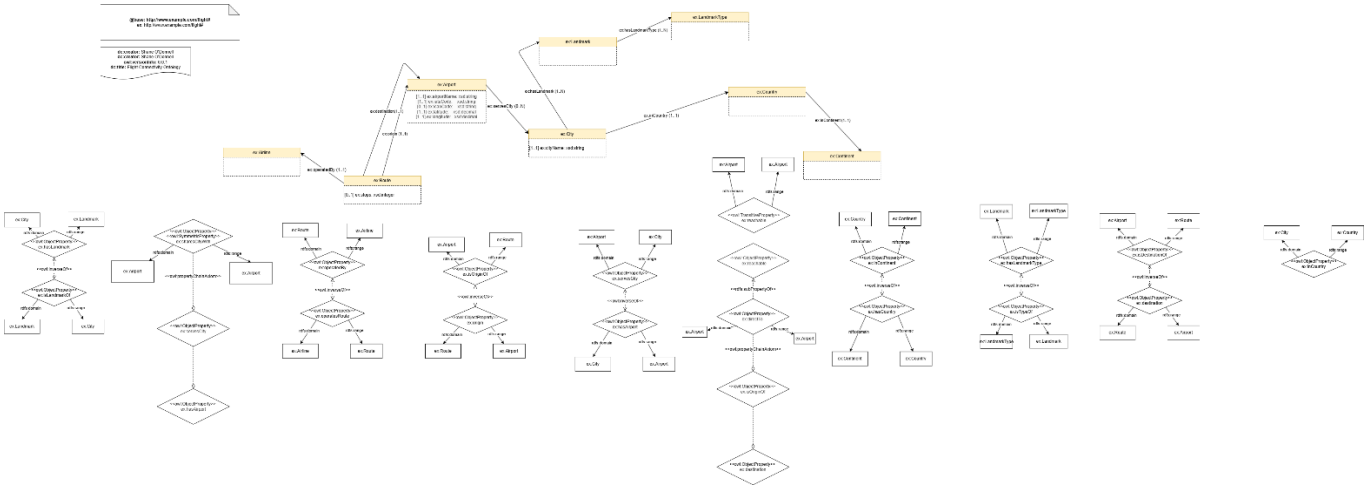

Figure 14: CHOWLK diagram

Once the CHOWLK diagram had been converted it was possible to check each data property was inferred correctly. The first property asserted was relating airports to the city they serve, this can be seen below outlining the property “servesCity”, which will help allow queries that involve a user looking to find flights which will bring them to a city and return the airport which resides in the desired city. As well as this, an inverse property was defined as “hasAirport”.

```
ex:servesCity a owl:ObjectProperty ;
    rdfs:domain ex:Airport ; rdfs:range ex:City ;
    rdfs:label "serves the city" .

ex:hasAirport a owl:ObjectProperty ;
  rdfs:domain ex:City ; rdfs:range ex:Airport ;
  rdfs:label "has the airport" ;
  owl:inverseOf ex:servesCity .
```

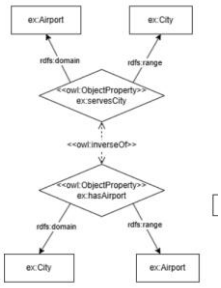

Figure 15: Chowlk Diagram of servesCity, inverseProperty and associated ttl code generated

To further add meaning to the data, the connection between cities and the countries they are in were made by the predicate “inCountry”. This can be seen below.

```
# make connection between cities and countries
ex:inCountry a owl:ObjectProperty ;
    rdfs:domain ex:City ; rdfs:range ex:Country .
```

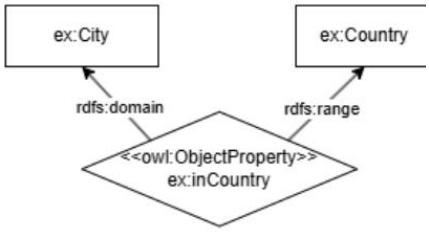

Figure 16: Chowlk Diagram of inCountry, and associated ttl code generated

The logic for adding routes and connecting the origin and destination to the airports is done by creating predicates by the name “origin” and “destination” assigned in the domain of the class Route and in the domain of the Airport class. This gives a directed edge at route level which can be later projected to an airport to airport level connectivity. An OWL inverse property was created for “origin” and “destination” to create the connections with the predicates “isOriginOf” and “isDestinationOf”.

```
ex:origin a owl:ObjectProperty ;
    rdfs:label "origin" ;
    rdfs:domain ex:Route ;
    rdfs:range ex:Airport ;
    owl:inverseOf ex:isOriginOf .

ex:destination a owl:ObjectProperty ;
    rdfs:label "destination" ;
    rdfs:domain ex:Route ;
    rdfs:range ex:Airport ;
    owl:inverseOf ex:isDestinationOf .
```

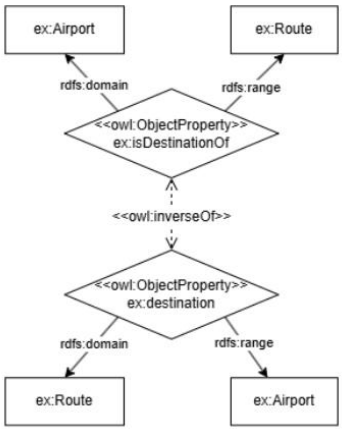

Figure 17: Chowlk Diagram of destination, inverseProperty and associated ttl code generated for origin and destination and their inverse

Ex:operatedBy connects each route and the ex:Airline that flies that route, its inverse was also included as “OperatesRoute” . This pair will allow common questions to be answered such as “who operates this route” and “which routes does Aer Lingus operate” in the form of SPARQL queries.

```
ex:operatesRoute a owl:ObjectProperty ;
    rdfs:label "operates route" ;
    rdfs:domain ex:Airline ;
    rdfs:range ex:Route ;
    owl:inverseOf ex:operatedBy .

ex:operatedBy a owl:ObjectProperty ;
    rdfs:label "operated by" ;
    rdfs:domain ex:Route ;
    rdfs:range ex:Airline .
```

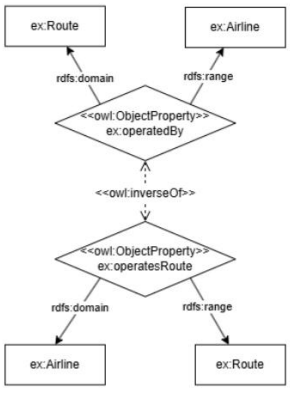

Figure 18: Chowlk Diagram of operatedBy, inverseProperty and associated ttl code generated

ex:sharesCityWith is a symmetric relationship between two different airports in the same city, the example for my thought process in this implementation is the fact Paris has 3 different airports. This allows grouping or filtering multi airport cities without having to repeatedly join through ex:servesCity. The property chain axiom inclusion allows for these connections to be inferred on the basis that if airport A servesCity C, and the city C hasAirport B then the reasoner can infer A ex:sharesCityWith B.

```
ex:sharesCityWith a owl:ObjectProperty,
        owl:SymmetricProperty ;
    rdfs:label "shares city with" ;
    rdfs:domain ex:Airport ;
    rdfs:range ex:Airport ;
    owl:propertyChainAxiom ( ex:servesCity ex:hasAirport )
```

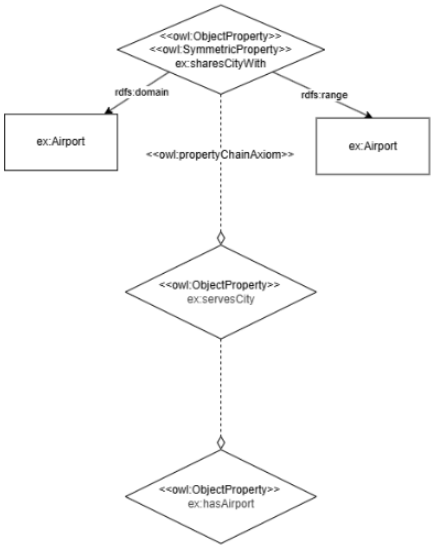

Figure 19: Chowlk Diagram of sharesCityWith, propertyChainAxiom and associated ttl code generated

The next property outlined is the "directTo" property which creates a connection between airports which operate a direct flight to each other. The ex:directTo property is inferred automatically using OWL 2 property chain axiom. This line instructs the reasoner that whenever airport A is the origin of a Route and that route has a destination B it can infer the connect that the connection "A ex:directTo B" is valid. In addition, since directTo is a sub property of "reachable", reachable can represent any connection (direct or layover) and "directTo" represents any immediate connection. This design supports both direct and transitive connections.

```
ex:directTo a owl:ObjectProperty ;
    rdfs:label "direct to" ;
    rdfs:domain ex:Airport ;
    rdfs:range ex:Airport ;
    rdfs:subPropertyOf ex:reachable ;
    owl:propertyChainAxiom ( ex:isOriginOf ex:destination ) .
```

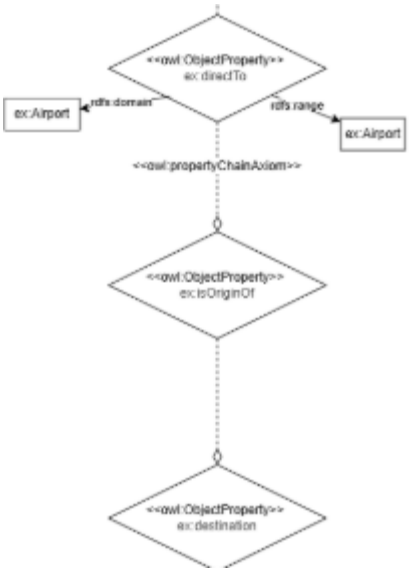

Figure 20: Chowlk Diagram of directTo, propertyChainAxiom and associated ttl code generated

An example of a transitive property outlined in the knowledge graph is with "reachable". The "directTo" property is a subproperty of reachable meaning a direct relation between these two properties is declared. Thus, if there is a flight from one airport "directTo" another, this destination is "reachable".

```
ex:reachable a owl:ObjectProperty,
        owl:TransitiveProperty ;
    rdfs:label "reachable" ;
    rdfs:domain ex:Airport ;
    rdfs:range ex:Airport .
```

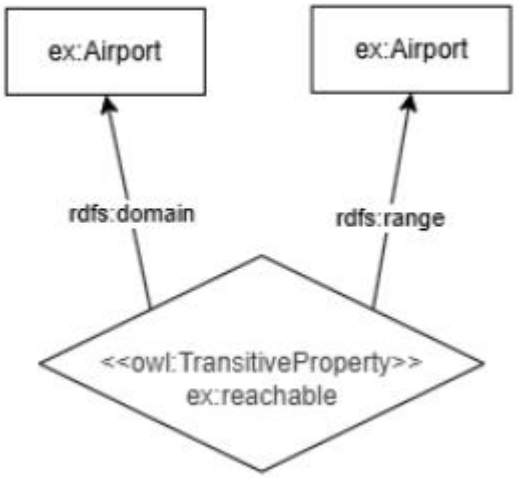

Figure 21: Chowlk Diagram of reachable, TransitiveProperty and associated ttl code generated

To create the connections for the landmark area of the graph the object property “hasLandmark” was created which connects cities and the landmarks that they have.

```
ex:isLandmarkOf a owl:ObjectProperty ;
    rdfs:label "is landmark of" ;
    rdfs:domain ex:Landmark ;
    rdfs:range ex:City ;
    owl:inverseOf ex:hasLandmark .

ex:hasLandmark a owl:ObjectProperty ;
    rdfs:label "has landmark" ;
    rdfs:domain ex:City ;
    rdfs:range ex:Landmark .
```

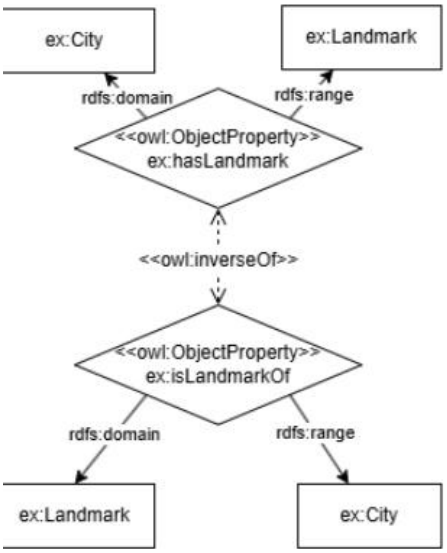

Figure 22: Chowlk Diagram of hasLandmark, inverseProperty and associated ttl code generated

The connection between the classes country and continent were connected using the Object Property “inContinent” and its inverse “hasCountry”.

```
ex:hasCountry a owl:ObjectProperty ;
    rdfs:label "has country" ;
    rdfs:domain ex:Continent ;
    rdfs:range ex:Country ;
    owl:inverseOf ex:inContinent .

ex:inContinent a owl:ObjectProperty ;
    rdfs:label "in continent" ;
    rdfs:domain ex:Country ;
    rdfs:range ex:Continent .
```

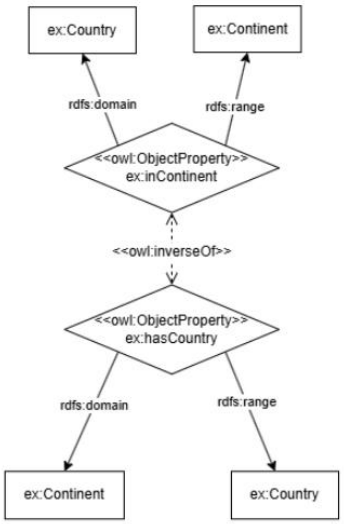

Figure 23: Chowlk Diagram of inContinent, inverseProperty and associated ttl code generated

The connection between a Landmark and the type of landmark was created my connecting the classes “landmark” and “landmarkType” using the object properties “hasLandmarkType” and its inverse “isTypeOf”.

```
ex:hasLandmarkType a owl:ObjectProperty ;
    rdfs:label "has landmark type" ;
    rdfs:domain ex:Landmark ;
    rdfs:range ex:LandmarkType .

ex:isTypeOf a owl:ObjectProperty ;
    rdfs:label "is type of" ;
    rdfs:domain ex:Landmark ;
    rdfs:range  ex:LandmarkType ;
    owl:inverseOf ex:hasLandmarkType .
```

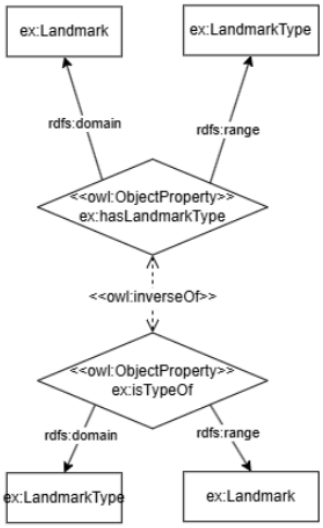

Figure 24: Chowlk Diagram of hasLandmarkType, inverseProperty and associated ttl code generated

The cardinality restrictions for each class were also defined within the ontology. For the country class it ensures that each country only belongs to one continent. For the landmark class it ensures that each Landmark is assigned a landmark type.

```
ex:Country a owl:Class ;
    rdfs:label "Country" ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:cardinality "1"^^xsd:nonNegativeInteger ;
            owl:onProperty ex:inContinent ] .

ex:Landmark a owl:Class ;
    rdfs:label "Landmark" ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:minCardinality "1"^^xsd:nonNegativeInteger ;
            owl:onProperty ex:hasLandmarkType ] .
```

Figure 25: Cardinality restrictions for Country and Landmark class

For the city class the cardinality restrictions ensure that each city has at least 1 landmark, each city has a object CityName, and each city belongs to exactly one country. For the route class it is ensured each route has a singular origin and destination and operated by only one airline.

```
ex:City a owl:Class ;
    rdfs:label "City" ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:minCardinality "1"^^xsd:nonNegativeInteger ;
            owl:onProperty ex:hasLandmark ],
        [ a owl:Restriction ;
            owl:onDataRange xsd:string ;
            owl:onProperty ex:cityName ;
            owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ] ,
        [ a owl:Restriction ;
            owl:onProperty ex:inCountry ;
            owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
            owl:onClass ex:Country ] .

ex:Route a owl:Class ;
    rdfs:label "Route" ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:cardinality "1"^^xsd:nonNegativeInteger ;
            owl:onProperty ex:origin ],
        [ a owl:Restriction ;
            owl:cardinality "1"^^xsd:nonNegativeInteger ;
            owl:onProperty ex:destination ],
        [ a owl:Restriction ;
            owl:cardinality "1"^^xsd:nonNegativeInteger ;
            owl:onProperty ex:operatedBy ],
        [ a owl:Restriction ;
            owl:maxQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
            owl:onDataRange xsd:integer ;
            owl:onProperty ex:stops ] .
```

Figure 26: Cardinality restrictions for City and Route class

For the airport class it is ensured that airports online have maximum one icao code, longitude, latitude, airport name and iata code.

```
ex:Airport a owl:Class ;
    rdfs:label "Airport" ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:maxQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
            owl:onDataRange xsd:string ;
            owl:onProperty ex:icaoCode ],
        [ a owl:Restriction ;
            owl:onDataRange xsd:decimal ;
            owl:onProperty ex:latitude ;
            owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ],
        [ a owl:Restriction ;
            owl:onDataRange xsd:decimal ;
            owl:onProperty ex:longitude ;
            owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ],
        [ a owl:Restriction ;
            owl:onDataRange xsd:string ;
            owl:onProperty ex:airportName ;
            owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ],
        [ a owl:Restriction ;
            owl:onDataRange xsd:string ;
            owl:onProperty ex:iataCode ;
            owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ] .
```

Figure 27: Cardinality restrictions for Airport class

Fnally, the data properties from the rml mapping were defined as owl:DatatypeProperty's. This defines the domain for each data property, attaching them to their corresponding class. As well as this the range defines what type of value can be stored within this data property. The ranges for these data properties are outlined as string's. decimals and integers.

```
###############
# Data Properties - based on data from csv and added properties
###############
ex:iataCode a owl:DatatypeProperty ; rdfs:domain ex:Airport ; rdfs:range xsd:string .
ex:icaoCode a owl:DatatypeProperty ; rdfs:domain ex:Airport ; rdfs:range xsd:string .
ex:airportName a owl:DatatypeProperty ; rdfs:domain ex:Airport ; rdfs:range xsd:string .
ex:latitude a owl:DatatypeProperty ; rdfs:domain ex:Airport ; rdfs:range xsd:decimal .
ex:longitude a owl:DatatypeProperty ; rdfs:domain ex:Airport ; rdfs:range xsd:decimal .
ex:stops a owl:DatatypeProperty ; rdfs:domain ex:Route   ; rdfs:range xsd:integer .
```

Figure 28: Data properties

SHACL Shapes and Validation

To validate the graph which was designed SHACL was used, the SHACL file was uploaded to graphDB and if no error was outlined during the upload the graph shapes passed every validation.

The SHACL file outlined the following validations, the first class targeted was the Airport class in which it was defined to ensure airports can only serve one city, airports can't have more than one IATA code, latitude values must fall between -90 and +90, and the airport must have a AirportName data property.

```
ex:AirportShape a sh:NodeShape ;
    sh:targetClass ex:Airport ;

    #- should only serve a singular city - and the object servesCity should be pointing to is of class City
    sh:property [
        sh:path ex:servesCity ;
        sh:minCount 1 ; sh:maxCount 1 ;
        sh:class ex:City
    ] ;

    # cant more than 1 IATA
    sh:property [
        sh:path ex:iataCode ;
        sh:maxCount 1 ;
        sh:datatype xsd:string ;
    ] ;
    #latitude must fall between 90 and -90
    sh:property [
        sh:path ex:latitude ;
        sh:datatype xsd:decimal ;
        sh:minInclusive -90 ; sh:maxInclusive 90 ;
        sh:message "Latitude must be between -90 and 90." ;
    ] ;

    #airport name cant be empty
    sh:property [
        sh:path ex:airportName ;
        sh:datatype xsd:string ;
        sh:minLength 1 ;
        sh:message "Airport name must be a non-empty string" ;
    ] .
```

Figure 29: SHACL shapes for Airport class

For the city class it was checked to ensure that a city can only belong to a singular country, and the object property “hasLandmark” can only appear to connect between the city class and the Landmark class.

```
ex:CityShape a sh:NodeShape ;
    sh:targetClass ex:City ;

    sh:property [
        sh:path ex:hasLandmark ;
        sh:class ex:Landmark ;
        sh:message "cities can only link to the class Landmark with ex:hasLandmark" ;
    ] ;

    #city can only belong to singular country
    sh:property [
        sh:path ex:inCountry ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:class ex:Country ;
    ] .
```

Figure 30: SHACL shapes for City class

It was very important to ensure the connection between routes and airports was checked to ensure the rml mapping was successful between the two data sources. It was checked that routes have exactly one origin and one destination and the amount of stops can’t be negative.

```
ex:RouteShape a sh:NodeShape ;
    sh:targetClass ex:Route ;

    #exactly 1 origin per route
    sh:property [
        sh:path ex:origin ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:class ex:Airport ;
    ] ;

    #exactly 1 destination per route
    sh:property [
        sh:path ex:destination ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:class ex:Airport ;
    ] ;

    sh:property [
        sh:path ex:stops ;
        sh:datatype xsd:integer ;
        sh:minInclusive 0 ;
        sh:message "stops cant be negative " ;
    ] .
```

Figure 31: SHACL shapes for Route class

The Landmark class was checked to ensure that each landmark has exactly one landmark type, to ensure that a landmark can't be both a museum and a bridge.

```
ex:LandmarkShape a sh:NodeShape ;
    sh:targetClass ex:Landmark ;
    sh:property [
        sh:path ex:hasLandmarkType ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
        sh:class ex:LandmarkType ;
        sh:message "A landmark must have exactly one ex:hasLandmarkType." ;
    ] .
```

Figure 32: SHACL shapes for Landmark class

Finally the country class was checked to ensure that countries were not share among different continents.

```
ex:CountryShape a sh:NodeShape ;
    sh:targetClass ex:Country ;
    sh:property [
        sh:path ex:inContinent ;
        sh:minCount 1 ; sh:maxCount 1 ;
        sh:class ex:Continent ;
        sh:message "Each country must be in exactly one continent" ;
    ] .
```

Figure 33: SHACL shape for Country class

Once the SHACL file was uploaded to graphDB, the shapes were validated and if the uploaded was successful, all shapes were validated successfully and passed. On the first upload the

SHACL file highlighted an error in which Landmark Types (Museum, Bridge etc.) were being defined to belong to the class LandmarkType and the class Landmark which is not the desired functionality. This problem was diagnosed to be an ontology error with the definition of the domain and range of hasLandmarkType. Once this was amended, all SHACL validations passed.

```
_:e0fa2a1050f346e2beb1188c05845b2d53409 a sh:ValidationResult;
 sh:focusNode <http://example.org/flight#Museum>;
 rsx:shapesGraph rdf4j:SHACLShapeGraph;
 sh:resultPath <http://example.org/flight#hasLandmarkType>;
 sh:sourceConstraintComponent sh:MinCountConstraintComponent;
 sh:resultSeverity sh:Violation;
 sh:resultMessage "A landmark must have exactly one ex:hasLandmarkType.";
 sh:sourceShape _:823198be-0017-4e74-950c-55b8621b4bdc-10 .
```

Figure 34: SHACL validation violation report from GraphDB

SPARQL Queries

Query 1

To test the capabilities of the graph a series of queries were run to demonstrate the functionality. The first query tested was returning the full list of airports in Ireland, this query is outlines below as well as its result. The query begins my finding all the airports and the cities they serve, from here only cities in the country of Ireland would be accepted, then on optional statement is used in which if the graph has the data for the airport, return both the airport and its full name.

```
PREFIX ex:<http://example.org/flight#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
SELECT ?airport ?name
WHERE {
  ?airport a ex:Airport ;
           ex:servesCity ?city .
  ?city ex:inCountry <http://example.org/flight/country/Ireland> .
  OPTIONAL { ?airport ex:airportName ?name }
}
ORDER BY COALESCE(?name, STR(?airport))
```

Figure 35: Query Returning airports in Ireland

The graph was able to find a great number of Irish airports as seen below in figure 29. The graph successfully returned the airport ID and the name associated with each ID number.

	airport	name
1	http://example.org/flight/airport/8698	"Bantry Aerodrome"
2	http://example.org/flight/airport/602	"Casement Air Base"
3	http://example.org/flight/airport/6422	"Connemara Regional Airport"
4	http://example.org/flight/airport/596	"Cork Airport"
5	http://example.org/flight/airport/5577	"Donegal Airport"
6	http://example.org/flight/airport/599	"Dublin Airport"
7	http://example.org/flight/airport/597	"Galway Airport"
8	http://example.org/flight/airport/7030	"Inisheer Aerodrome"
9	http://example.org/flight/airport/7468	"Inishmaan Aerodrome"
10	http://example.org/flight/airport/6421	"Inishmore Aerodrome"
11	http://example.org/flight/airport/600	"Ireland West Knock Airport"
12	http://example.org/flight/airport/601	"Kerry Airport"
13	http://example.org/flight/airport/8683	"Newcastle Aerodrome"
14	http://example.org/flight/airport/603	"Shannon Airport"
15	http://example.org/flight/airport/604	"Sligo Airport"
16	http://example.org/flight/airport/605	"Waterford Airport"
17	http://example.org/flight/airport/5578	"Weston Airport"

Figure 36: Query return result

Query 2

Continuing on, the next query desired to return all the direct flights someone could take from Dublin Airport specifically. This query desired to return the destination airports ID, the name of the airport and the country the airport is in. The possible destinations are found using the "DirectTo" predicate and the Airport ID for Dublin airport as the subject. From here the optional statement was used again to return the airports name if the name is present in the graph. The "servesCity" predicate was used to find the city that the destination airport resides in and then the country that city is connected to through the predicate "inCountry" was returned as well.

```
PREFIX ex:<http://example.org/flight#>
SELECT DISTINCT ?dest ?name ?country
WHERE {
  <http://example.org/flight/airport/599> ex:directTo ?dest .
  OPTIONAL { ?dest ex:airportName ?name }
  ?dest ex:servesCity ?city .
  ?city ex:inCountry ?country .
}
```

Figure 37: Query Returning destinations with direct flight from Dublin Airport

The query's result displays that 28 destinations are on offer by Dublin Airport serving the countries of Germany and the United Kingdom.

	dest	name	country
1	http://example.org/flight/airport/337	"Berlin-Schönefeld Airport"	http://example.org/flight/country/Germany
2	http://example.org/flight/airport/340	"Frankfurt am Main Airport"	http://example.org/flight/country/Germany
3	http://example.org/flight/airport/342	"Hamburg Airport"	http://example.org/flight/country/Germany
4	http://example.org/flight/airport/344	"Cologne Bonn Airport"	http://example.org/flight/country/Germany
5	http://example.org/flight/airport/345	"Düsseldorf Airport"	http://example.org/flight/country/Germany
6	http://example.org/flight/airport/346	"Munich Airport"	http://example.org/flight/country/Germany
7	http://example.org/flight/airport/350	"Stuttgart Airport"	http://example.org/flight/country/Germany
8	http://example.org/flight/airport/352	"Hannover Airport"	http://example.org/flight/country/Germany
9	http://example.org/flight/airport/353	"Bremen Airport"	http://example.org/flight/country/Germany
10	http://example.org/flight/airport/355	"Frankfurt-Hahn Airport"	http://example.org/flight/country/Germany
11	http://example.org/flight/airport/469	"Birmingham International Airport"	http://example.org/flight/country/United%20Kingdom
12	http://example.org/flight/airport/478	"Manchester Airport"	http://example.org/flight/country/United%20Kingdom
13	http://example.org/flight/airport/488	"Cardiff International Airport"	http://example.org/flight/country/United%20Kingdom
14	http://example.org/flight/airport/490	"Bristol Airport"	http://example.org/flight/country/United%20Kingdom
15	http://example.org/flight/airport/491	"Liverpool John Lennon Airport"	http://example.org/flight/country/United%20Kingdom
16	http://example.org/flight/airport/492	"London Luton Airport"	http://example.org/flight/country/United%20Kingdom
17	http://example.org/flight/airport/494	"Bournemouth Airport"	http://example.org/flight/country/United%20Kingdom
18	http://example.org/flight/airport/495	"Southampton Airport"	http://example.org/flight/country/United%20Kingdom
19	http://example.org/flight/airport/502	"London Gatwick Airport"	http://example.org/flight/country/United%20Kingdom
20	http://example.org/flight/airport/503	"London City Airport"	http://example.org/flight/country/United%20Kingdom
21	http://example.org/flight/airport/507	"London Heathrow Airport"	http://example.org/flight/country/United%20Kingdom
22	http://example.org/flight/airport/508	"Southend Airport"	http://example.org/flight/country/United%20Kingdom
23	http://example.org/flight/airport/514	"Blackpool International Airport"	http://example.org/flight/country/United%20Kingdom
24	http://example.org/flight/airport/517	"Leeds Bradford Airport"	http://example.org/flight/country/United%20Kingdom
25	http://example.org/flight/airport/521	"Newcastle Airport"	http://example.org/flight/country/United%20Kingdom
26	http://example.org/flight/airport/523	"East Midlands Airport"	http://example.org/flight/country/United%20Kingdom
27	http://example.org/flight/airport/532	"Aberdeen Dyce Airport"	http://example.org/flight/country/United%20Kingdom
28	http://example.org/flight/airport/534	"Glasgow International Airport"	http://example.org/flight/country/United%20Kingdom

Figure 38: Query return result

Query 3

Similarly, to test the transitive property “reachable” the same query was ran but instead of returning the flights direct from Dublin, the airports that are reachable by layovers was returned using the ex:reachable property. As seen from the responses, Spanish airports are not present with the “DirectTo” query but is present in the “reachable” query. This is because reachable is transitive meaning if Airport A is connected to Airport B and Airport B is connected to Airport C, technically Airport C is reachable from Airport A with a layover.

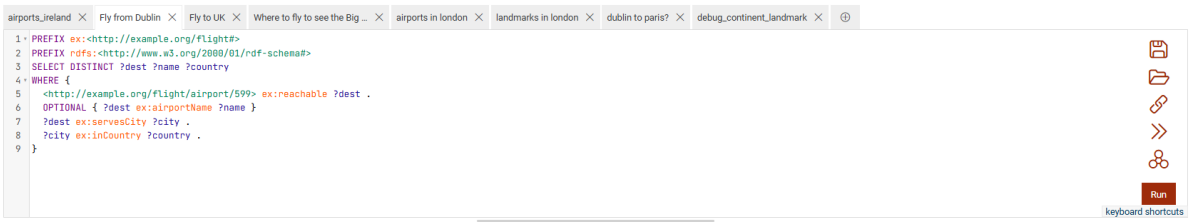

Figure 39: Query returning Airport destinations reachable from Dublin with layover

	dest	name	country
1	http://example.org/flight/airport/1051	"Fuerteventura Airport"	http://example.org/flight/country/Spain
2	http://example.org/flight/airport/1052	"Hierro Airport"	http://example.org/flight/country/Spain
3	http://example.org/flight/airport/1053	"La Palma Airport"	http://example.org/flight/country/Spain
4	http://example.org/flight/airport/1054	"Gran Canaria Airport"	http://example.org/flight/country/Spain
5	http://example.org/flight/airport/1055	"Lanzarote Airport"	http://example.org/flight/country/Spain
6	http://example.org/flight/airport/1056	"Tenerife South Airport"	http://example.org/flight/country/Spain
7	http://example.org/flight/airport/1057	"Tenerife Norte Airport"	http://example.org/flight/country/Spain
8	http://example.org/flight/airport/1058	"Melilla Airport"	http://example.org/flight/country/Spain
9	http://example.org/flight/airport/1212	"Alicante International Airport"	http://example.org/flight/country/Spain
10	http://example.org/flight/airport/1213	"Almería International Airport"	http://example.org/flight/country/Spain
11	http://example.org/flight/airport/1214	"Asturias Airport"	http://example.org/flight/country/Spain
12	http://example.org/flight/airport/1216	"Bilbao Airport"	http://example.org/flight/country/Spain
13	http://example.org/flight/airport/1218	"Barcelona International Airport"	http://example.org/flight/country/Spain
14	http://example.org/flight/airport/1220	"A Coruña Airport"	http://example.org/flight/country/Spain
15	http://example.org/flight/airport/1222	"Girona Airport"	http://example.org/flight/country/Spain
16	http://example.org/flight/airport/1223	"Federico García Lorca Airport"	http://example.org/flight/country/Spain
17	http://example.org/flight/airport/1225	"Ibiza Airport"	http://example.org/flight/country/Spain
18	http://example.org/flight/airport/1226	"Jerez Airport"	http://example.org/flight/country/Spain
19	http://example.org/flight/airport/1227	"San Javier Airport"	http://example.org/flight/country/Spain
20	http://example.org/flight/airport/1229	"Adolfo Suárez Madrid–Barajas Airport"	http://example.org/flight/country/Spain
21	http://example.org/flight/airport/1230	"Málaga Airport"	http://example.org/flight/country/Spain
22	http://example.org/flight/airport/1231	"Menorca Airport"	http://example.org/flight/country/Spain
23	http://example.org/flight/airport/1236	"Reus Air Base"	http://example.org/flight/country/Spain
24	http://example.org/flight/airport/1243	"Santiago de Compostela Airport"	http://example.org/flight/country/Spain
25	http://example.org/flight/airport/1246	"Valencia Airport"	http://example.org/flight/country/Spain
26	http://example.org/flight/airport/1251	"Santander Airport"	http://example.org/flight/country/Spain

Figure 40: Query returned results

Query 4

Part of the implementation was adding connections between famous landmarks present in the world and the city to visit to see them. An example of this implementation being functional is seen using a query which return the Airport Destinations to fly to in order to see the Big Ben. This query returns the city, airport and airport's name associated with the chosen landmark. The query begins by returning the city which "hasLandmark" Big Ben. From here the airports associated with the city are returned as well the full name of the airport.

```
PREFIX ex:<http://example.org/flight#>
SELECT DISTINCT ?city ?airport ?name
WHERE {
    ?city ex:hasLandmark ex:big_ben .
    ?city ex:hasAirport ?airport .
    ?airport ex:airportName ?name .

}
```

Figure 41: Query returning Airports to fly to if you want to visit Big Ben

From the returned airports it can be seen that the city of London has multiple airports, this city can be used as an example to test the "sharesCityWith" predicate.

	city	airport	name
1	http://example.org/flight/city/London_United%20Kingdom	http://example.org/flight/airport/492	"London Luton Airport"
2	http://example.org/flight/city/London_United%20Kingdom	http://example.org/flight/airport/502	"London Gatwick Airport"
3	http://example.org/flight/city/London_United%20Kingdom	http://example.org/flight/airport/503	"London City Airport"
4	http://example.org/flight/city/London_United%20Kingdom	http://example.org/flight/airport/507	"London Heathrow Airport"
5	http://example.org/flight/city/London_United%20Kingdom	http://example.org/flight/airport/548	"London Stansted Airport"
6	http://example.org/flight/city/London_United%20Kingdom	http://example.org/flight/airport/7722	"London Heliport"

Figure 42: Query return

Query 5

For this example the user is aware of Heathrow Airport but wants to know all the airports which are also in London. For this the sharesCityWith property is used with Heathrow's airport ID as the subject.

```
PREFIX ex:<http://example.org/flight#>
SELECT DISTINCT ?airport ?name
WHERE {
    <http://example.org/flight/airport/507> ex:sharesCityWith ?airport .
    ?airport ex:airportName ?name .


}
```

Figure 43: Query returning airports that share a city with Heathrow Airport

	airport	name
1	http://example.org/flight/airport/492	"London Luton Airport"
2	http://example.org/flight/airport/502	"London Gatwick Airport"
3	http://example.org/flight/airport/503	"London City Airport"
4	http://example.org/flight/airport/507	"London Heathrow Airport"
5	http://example.org/flight/airport/548	"London Stansted Airport"
6	http://example.org/flight/airport/7722	"London Heliport"

Figure 44: Query return

Query 6

The next query returns all the landmarks in London so that a user can hypothetically plan an itinerary based around the sights in the city.

```
PREFIX ex:<http://example.org/flight#>
SELECT DISTINCT ?landmark
WHERE {
    <http://example.org/flight/city/London_United%20Kingdom> ex:hasLandmark ?landmark


}
```

Run keyboard shortcuts

Table | Raw response | Pivot Table | Google Chart | Download as

Filter query results | Compact view | Hide row numbers | Showing results from 0 to 3 of 3. Query took 0.1s, yesterday at 15:44.

	landmark
1	ex:tower_of_london
2	ex:big_ben
3	ex:tower_bridge

Figure 45: Query and returned points for finding every landmark in London

Query 7

This query returns all the monuments present in the graph, the city they reside in and what airport to fly to if the user want to visit the monument.

```
SELECT DISTINCT
  ?lmName
  ?city     ?cityName
  ?airport  ?airportName
WHERE {
  ?landmark a ex:Landmark ;
            ex:hasLandmarkType ex:Monument .
  ?landmark rdfs:label ?lmName .
  ?city ex:hasLandmark ?landmark .
  ?city ex:cityName ?cityName .
  ?airport a ex:Airport ;
           ex:servesCity ?city .
  ?airport ex:airportName ?airportName .
}
```

Press Alt+Enter to autocomplete

Run keyboard shortcuts

The returned values from this query list all the monuments, their corresponding cities and airports. There are some reoccurring monuments due to some cities sharing airports.

	lmName	city	cityName	airport
1	"Eiffel Tower"	http://example.org/flight/city/Paris_France	"Paris"	http://example.org/flight/airport/1380	"Paris-Le Bourget Airport"
2	"Eiffel Tower"	http://example.org/flight/city/Paris_France	"Paris"	http://example.org/flight/airport/1382	"Charles de Gaulle International Airport"
3	"Eiffel Tower"	http://example.org/flight/city/Paris_France	"Paris"	http://example.org/flight/airport/1386	"Paris-Orly Airport"
4	"Molly Malone Statue"	http://example.org/flight/city/Dublin_Ireland	"Dublin"	http://example.org/flight/airport/599	"Dublin Airport"
5	"Tower of London"	http://example.org/flight/city/London_United%20Kingdom	"London"	http://example.org/flight/airport/492	"London Luton Airport"
6	"Tower of London"	http://example.org/flight/city/London_United%20Kingdom	"London"	http://example.org/flight/airport/502	"London Gatwick Airport"
7	"Tower of London"	http://example.org/flight/city/London_United%20Kingdom	"London"	http://example.org/flight/airport/503	"London City Airport"
8	"Tower of London"	http://example.org/flight/city/London_United%20Kingdom	"London"	http://example.org/flight/airport/507	"London Heathrow Airport"
9	"Tower of London"	http://example.org/flight/city/London_United%20Kingdom	"London"	http://example.org/flight/airport/548	"London Stansted Airport"
10	"Tower of London"	http://example.org/flight/city/London_United%20Kingdom	"London"	http://example.org/flight/airport/7722	"London Heliport"
11	"Big Ben"	http://example.org/flight/city/London_United%20Kingdom	"London"	http://example.org/flight/airport/492	"London Luton Airport"
12	"Big Ben"	http://example.org/flight/city/London_United%20Kingdom	"London"	http://example.org/flight/airport/502	"London Gatwick Airport"
13	"Big Ben"	http://example.org/flight/city/London_United%20Kingdom	"London"	http://example.org/flight/airport/503	"London City Airport"
14	"Big Ben"	http://example.org/flight/city/London_United%20Kingdom	"London"	http://example.org/flight/airport/507	"London Heathrow Airport"
15	"Big Ben"	http://example.org/flight/city/London_United%20Kingdom	"London"	http://example.org/flight/airport/548	"London Stansted Airport"
16	"Big Ben"	http://example.org/flight/city/London_United%20Kingdom	"London"	http://example.org/flight/airport/7722	"London Heliport"
17	"Edinburgh Castle"	http://example.org/flight/city/Edinburgh_United%20Kingdom	"Edinburgh"	http://example.org/flight/airport/535	"Edinburgh Airport"
18	"Brandenburg Gate"	http://example.org/flight/city/Berlin_Germany	"Berlin"	http://example.org/flight/airport/337	"Berlin-Schönefeld Airport"
19	"Brandenburg Gate"	http://example.org/flight/city/Berlin_Germany	"Berlin"	http://example.org/flight/airport/343	"Berlin-Tempelhof International Airport"
20	"Brandenburg Gate"	http://example.org/flight/city/Berlin_Germany	"Berlin"	http://example.org/flight/airport/351	"Berlin-Tegel Airport"
21	"Belém Tower"	http://example.org/flight/city/Lisbon_Portugal	"Lisbon"	http://example.org/flight/airport/1638	"Humberto Delgado Airport (Lisbon Portela Airport)"

Query 8

This query uses the ASK function to check if it is possible for a user to travel from Dublin to London and then travel from London to Paris afterwards.

```
PREFIX ex:  <http://example.org/flight#>

ASK {
  # hop 1: Dublin -> London
  <http://example.org/flight/airport/599> ex:directTo ?lon .
  ?lon ex:servesCity <http://example.org/flight/city/London_United%20Kingdom> .

  # hop 2: London -> Paris
  ?lon ex:directTo ?paris .
  ?paris ex:servesCity <http://example.org/flight/city/Paris_France> .
}
```

Press Alt+Enter to autocomplete

Run

keyboard shortcuts

Query took 0.1s, today at 15:11.

YES

Personal Reflection

This project offered great experience into how to construct a knowledge graph from scratch, including experience designing an ontology using CHOWLK, RML mapping from CSV files and converting this data to RDF, manually inputting RDF data through turtle, validating using SHACL and querying the graph using SPARQL. From this project I was able to successfully find relevant data from 2 different sources and connect the data using rml mapping using overlapping airport ID number as the common key to join and relate entities. This process highlighted the importance of consistent identifiers and ontology design to ensure interoperability between the two datasets.

By manually entering RDF data for continents, landmarks and landmark types, this furthered my ability to manually insert RDF data and solidified this ability which was first practice earlier in the module when a personal graph “about me” was created. This experience highlighted that creating a

graph by manually inserting rdf data would be extremely time consuming and the use of existing datasets speeds the process up a lot.

The creation of the ontology through CHOWLK helped me visualise the desired connections in the graph and allowed me to gain experience in using the tool. The most difficult aspect of this step was learning the "syntax" of the tool and how to create the desired connections for "owl;InverseOf" and owl:PropertyChainAxiom". This took many tries and analysis of the examples in the documentation provided from the GitHub repo.

Using SHACL was straight forward after practicing using the SHACL Playground website [2] which was found through a YouTube tutorial and was helpful to ensure the cardinality constraints defined the ontology were being asserted.

Once the data had been uploaded to GraphDB, querying the graph was very enjoyable as it allowed me to test all the different possible queries the graph could support.

Overall the desired implementation was designed although the shortcomings involved the fact that the size of the original graph (mapping every route and airport in the CSV) caused the import of data to graphDB to be stuck on "importing" which made me downsize to include only a few countries. As well as this at the beginning of the project I wanted to included data to find the travel time for each flight and the distance travelled but I could not find a dataset to support this so instead I implemented the landmark and continent functionality.

Deliverable	Amount Achieved	Note
Classes	8	
Subclasses	0	
Object Properties	18	Includes at least 1 transitive 1 symmetric and 1 inverseOf
Data Properties	7	
Explicit Statements	19,030	
Inferred Statements	76,197	
Instance triples	3869	

References

[1] - https://github.com/RMLio/rmlmapper-java

[2] - https://shacl.org/playground/

[3] - https://github.com/oeg-upm/Chowlk

[4] – https://openflights.org/data

[5] - https://www.kaggle.com/datasets/open-flights/flight-route-database?resource=download

Appendix

Throughout this report the files used have been outlined. The rdf triple file was shown as a small snippet due to the length of the file. If this is an issue the full file can be provided on request.