Task 1 : Summary Report

Creation of DDL code

As a first step, a relational PostgreSQL database is defined. The database is designed according to the ontology music, based on the classes and their object and data properties. The format used to define each table (based on a owl class) is the following:

```
CREATE TABLE Class (
  class_id SERIAL PRIMARY KEY,
  predicate_1 datatype,
  predicate_2 datatype,
  ……
  predicate_n datatype
);
```

For instance:

```
CREATE TABLE Composer (
  composer_id SERIAL PRIMARY KEY,
  firstName VARCHAR(100),
  lastName VARCHAR(100),
  name VARCHAR(100)
);
```

The use of SERIAL instead of INTEGER for the primary key definition is preferred, to allow an auto-increment of the key for each new added column.

Regarding the classes that are linked with a property (e.g Class1 predicate Class2), the implementation in the query is the following: after defining the primary key of the table (e.g Class1), we create a column e.g class2_id that stores INTEGER values and makes it a foreign key, linking it to the table of Class2. This is satisfied by using REFERECE Class2_table(class2_id). For instance:

```
CREATE TABLE Membership (
  membership_id SERIAL PRIMARY KEY,
  instrument_id INTEGER REFERENCES Instrument(instrument_id),
  orchestra_id INTEGER REFERENCES Orchestra(orchestra_id),
  musician_id INTEGER REFERENCES Musician(musician_id)

);
```

Regarding the use of Track primary key as a foreign key in the Composer table, it was decided not to follow the same approach as the other linked classes. Since we should allow for “many tracks to many composers” link, this is possible only if a new table is created, namely the *composer_track_link:*

```
CREATE TABLE composer_track_link (
  composer_id INTEGER REFERENCES Composer(composer_id),
  track_id INTEGER REFERENCES Track(track_id),
  PRIMARY KEY (composer_id, track_id)
);
```

Creation of DML code

In sequence, we want to populate the tables with data. It was decided to use synthetic data for this purpose, and populate the tables with arbitrary parameters that respect the semantics of the ontology. For example, the table Composer is populated by instances that feature real, arbitrary names (First name, Last name, Name). To generate names of persons, names of cities (data property “location”) or text (data property “description”), a python script was set up that uses the module Fake. This module generates random names of persons, cities or texts, that can be used to populate the table columns. Each of the generated instances is written as a query in the sql file that can be used to populate the table, after its creation. For example, for the table Composer, two exemplary DML queries are:

```
INSERT INTO Composer (firstName, lastName, name) VALUES ('Cynthia', 'Baker', 'Cynthia Baker');
INSERT INTO Composer (firstName, lastName, name) VALUES ('Laura', 'Valenzuela', 'Laura Valenzuela');
```

For the execution of the DDM and DDL queries, the GUI pgAdmin4 was used in which, the music-ontology database of this exercise was defined.

Creation of R2RML code

A ttl file is created including the R2RML code for mapping the data to music ontology. The format of the R2RML code is as follows: each mapping, called as “rr:TriplesMap” includes a defined data source “rr:LogicalTable” , a subject map “rr:SubjectMap” that specifies how to create the for each row in the database table and one or more rules to generate the predicate and object of the triples. The Triples map essentially maps the database table rows to RDF triples. As an example of the R2RML code implementation is the TriplesMap Membership, where the membership table is linked to orchestra, instrument and musician tables:

```
ex:MembershipMapping a rr:TriplesMap,
  rr:logicalTable [rr:tableName "Membership"];
  rr:subjectMap [
   rr:template "http://example.org/Membership/{membership_id}" ;
   rr:class ex:Membership ;
   ],


  rr:predicateObjectMap[
```

```
 rr:predicate ex:memberMusician;
 rr:objectMap [rr:template "http://example.org/Musician/{musician_id}"]
 ] ;

rr:predicateObjectMap[
 rr:predicate ex:memberOrchestra;
 rr:objectMap [rr:template "http://example.org/Orchestra/{orchestra_id}"]
 ] ;

rr:predicateObjectMap[
 rr:predicate ex:playsInstrument;
 rr:objectMap [rr:template "http://example.org/Instrument/{instrument_id}"]
 ] .
```