Format:

1. number 14 size

1.1 number 12 size

1.1.1 number 12 size; underlined; there is space between the last line of 1.1.1 and 1.1.2

main body number 11 size

Contents:

...

How to convert CSV to RDF into Virtuoso?

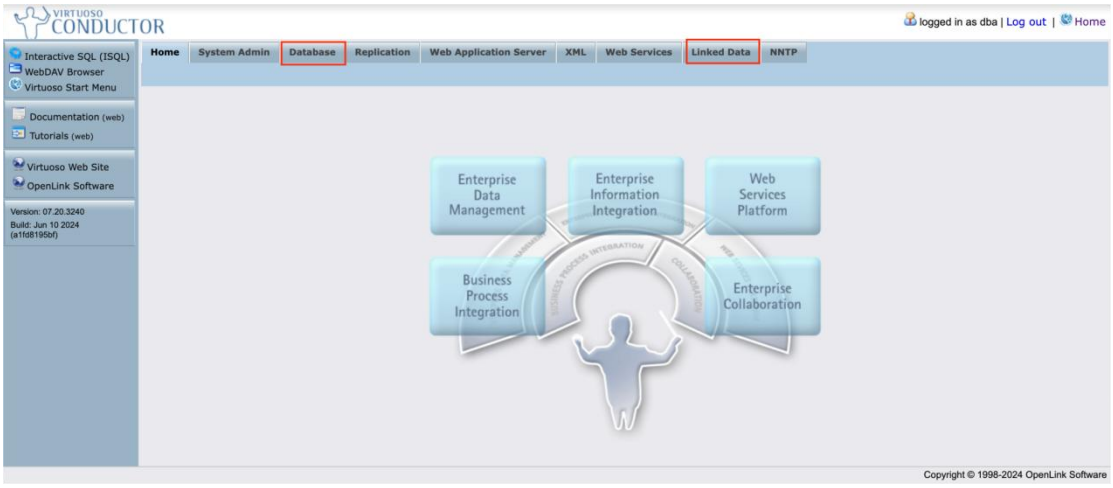
(Hi Ich, please know that I divided the former Method 1 into 2 methods, and Combined some following comparatively unimportant methods.)

0. Background and General Introduction

We adhere to the workflow of CSV to RDF because we currently need to perform reconciliation with Wikidata using OpenRefine to which CSV is more adaptable.
Assume the CSV we use has this structure: Each CSV represents one entity, the first column is the ID (or primary key); the other columns are attributes for the entity, which attributes are already reconciled as URIs.
Now we figure out at least 4 methods of converting “CSV to RDF (CSV2RDF) into Open Link Virtuoso”.
These methods as below actually interrelate with one another (rather than being isolated) , providing a comprehensive strategy.
Now let’s see these methods.

1. Virtuoso Conductor + SPARQL

The Virtuoso Conductor is the interface of Open Link Virtuoso. If you have a locally installed Virtuoso, its URL can be http://localhost:8890/conductor/.
And it looks like:



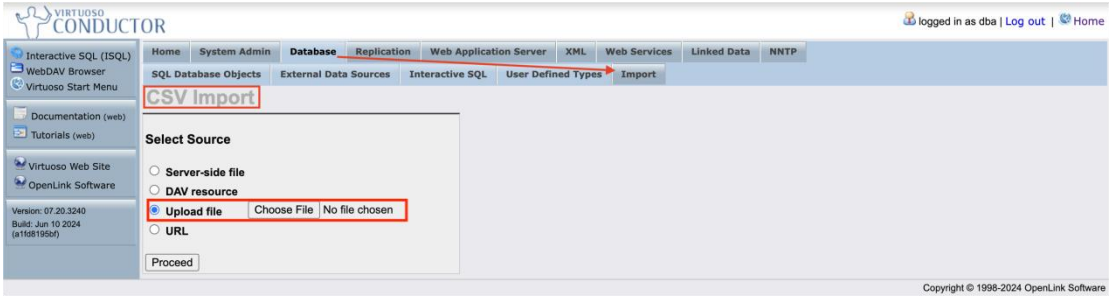
Notice the screenshot where we will manage mainly with two tabs. Database and Linked Data. Virtuoso is a representative of NOSQL database, which means “Not Only SQL”, namely a combination of Relational Data Base (RDB) and graph database (such as Linked Data). The Database tab is for RDB aspects.

We demonstrate step by step:

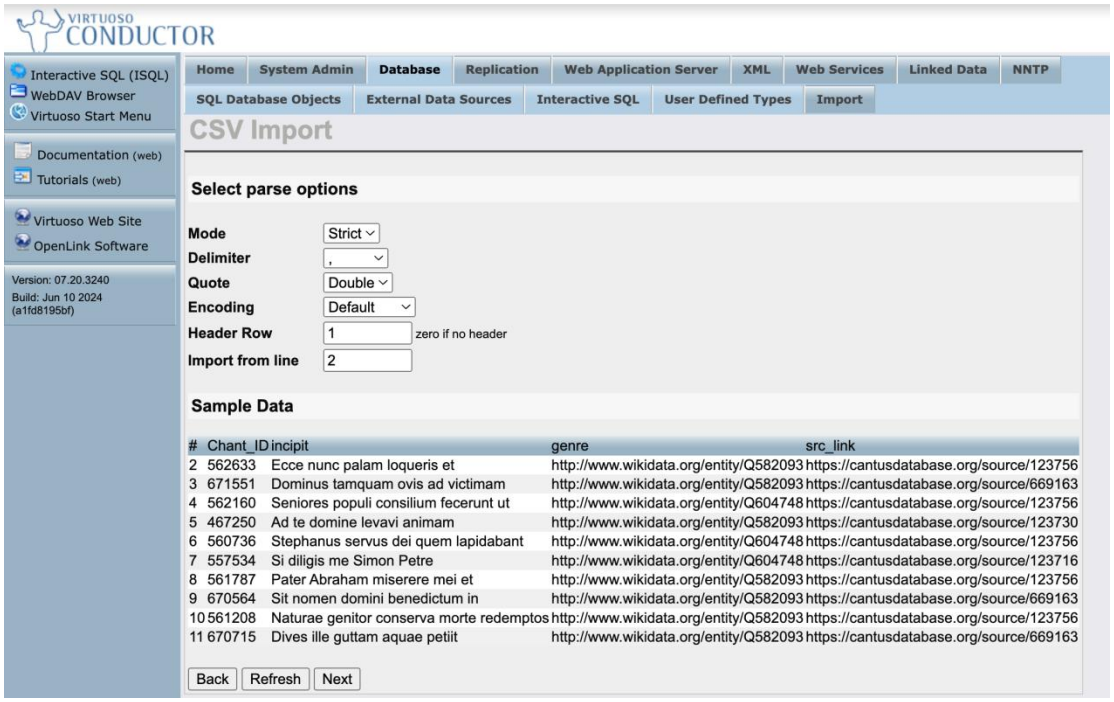
1.1 Demonstration

1.1.1 Upload CSV

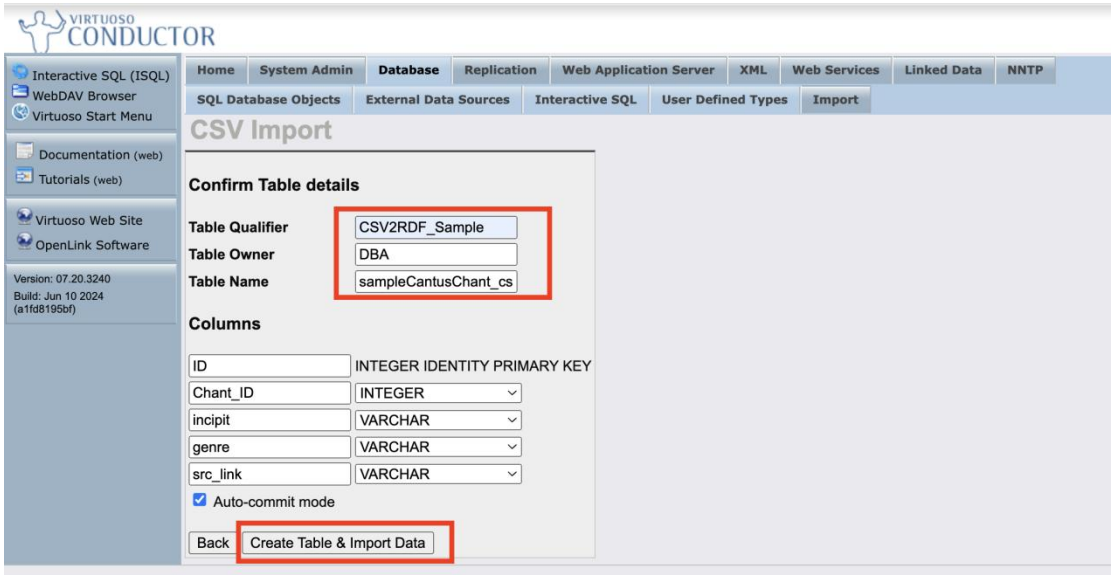
The basic interface for uploading a CSV to Virtuoso is navigated as follows: Conductor > Database > Import > Upload File > Choose File. See:



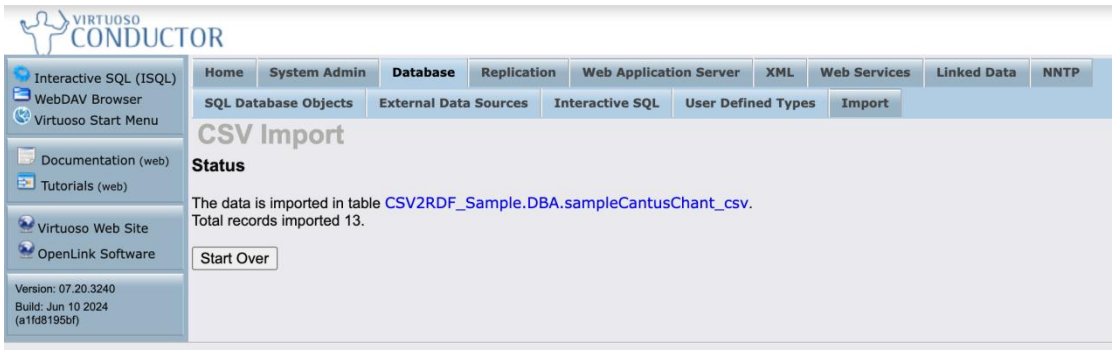
Given a sample CSV file named “sampleCantusChant.csv”, you may upload it and proceed, seeing interfaces one by one, such as:



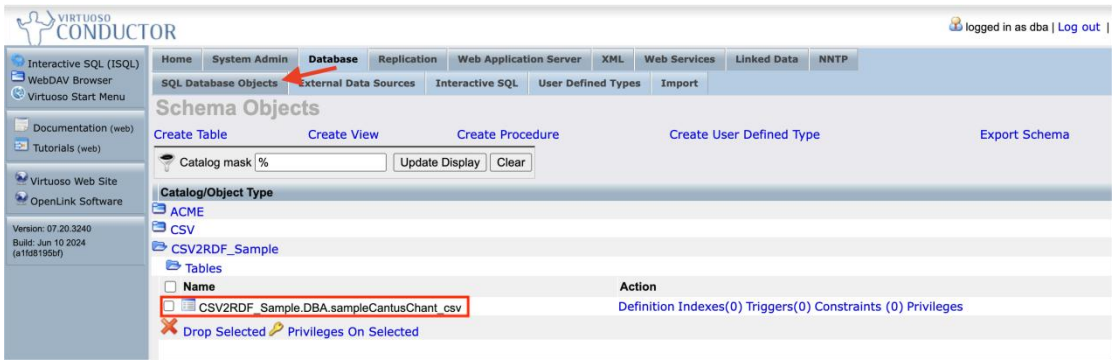
--please notice the property values of genre, src_link are already reconciled beforehand.



--before clicking on “Create Table & Import Data”, please designate a Table Qualifier/Owner/Name. Then proceed to upload the CSV:



After import, you can check the data from tab Database > SQL Database Objects:



1.1.2 CSV2RDF via the Linked Data tab

Then please navigate to Linked Data > Views:

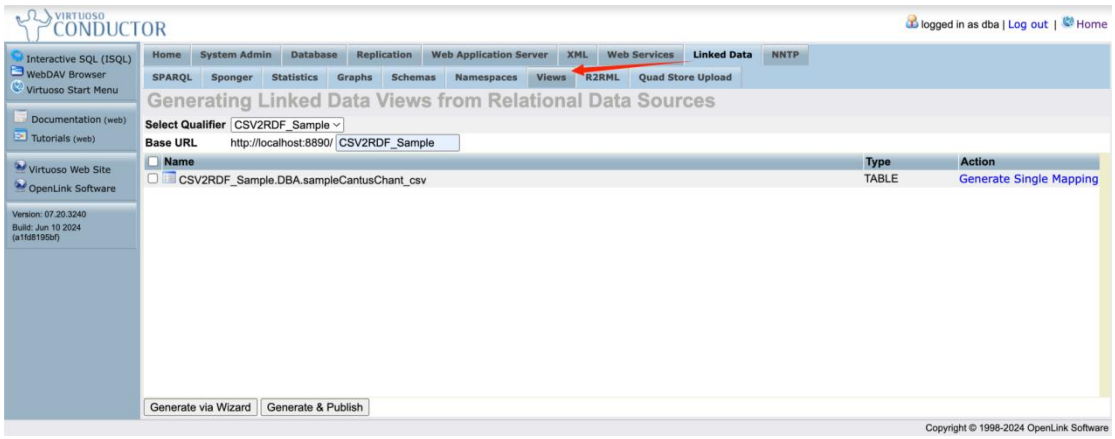
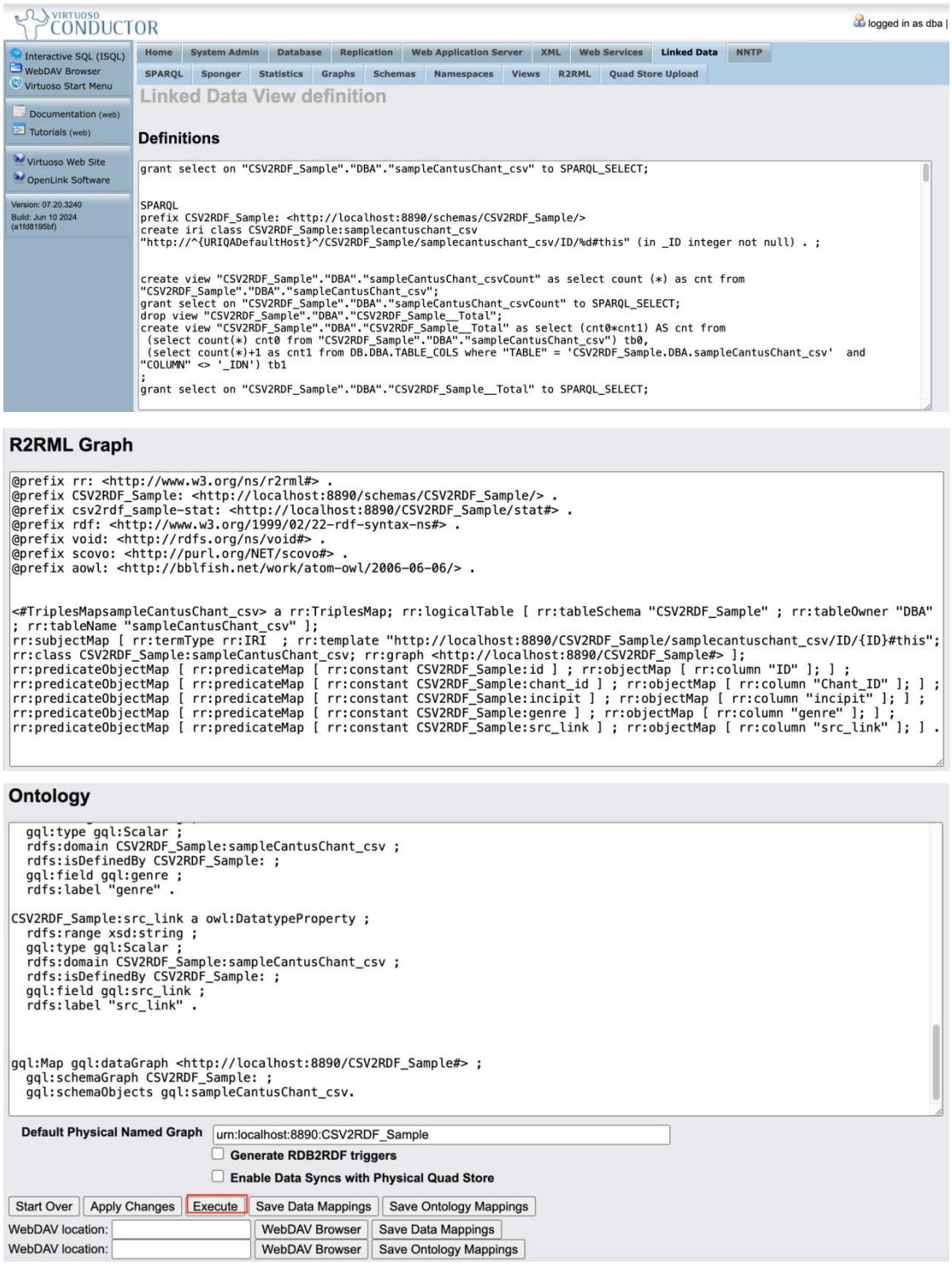


Figure *

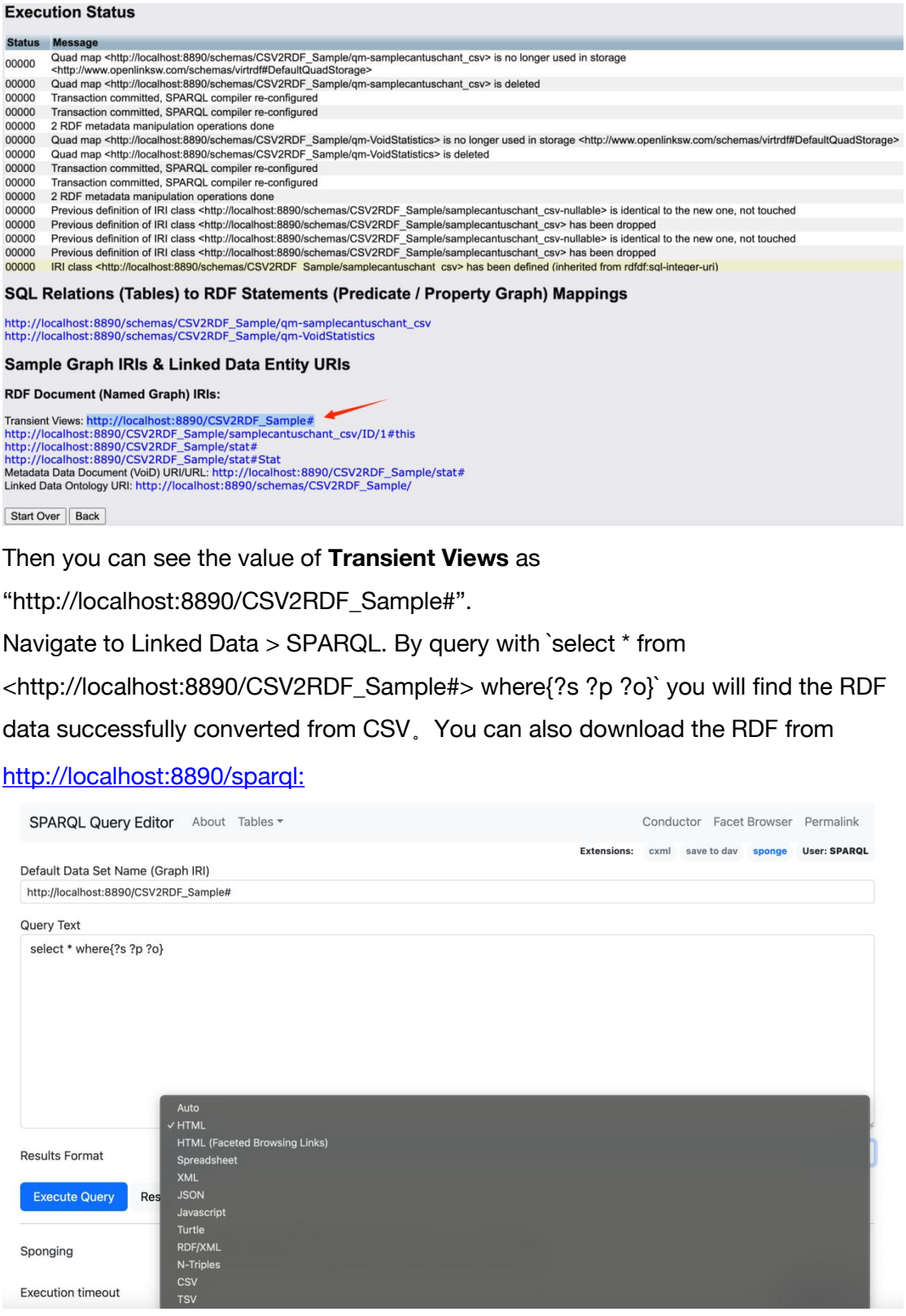
From the drop-down list of “Select Qualifier”, choose the value of “CSV2RDF_Sample”. Then you can define Base URL in a customized way. For example, we define it as “http://localhost:8890/CSV2RDF_Sample”. Then tick the left check box before clicking “Generate via Wizard”; go on and see:

click on “Prepare to Execute” to proceed:



There are 3 sections above: (1) Within “Definitions” there is a hybrid of SPARQL and SQL etc. (some Virtuoso specific language); (2) R2RML Graph: that is for mapping between attributes of CSV file and RDF URIs. (3)Ontology: to map the schema of the CSV to an ontology.

Click “Execute” in the screenshot above to generate the linked data view, and go to “Linked Data View definition”:



Then you can see the value of **Transient Views** as “http://localhost:8890/CSV2RDF_Sample#”.

Navigate to Linked Data > SPARQL. By query with `select * from <http://localhost:8890/CSV2RDF_Sample#> where{?s ?p ?o}` you will find the RDF data successfully converted from CSV。You can also download the RDF from http://localhost:8890/sparql:

Figure 0

Please notice that the URI of each instance/entity is defaulted like this: http://localhost:8890/CSV2RDF_Sample/samplecantuschant_csv/ID/1#this

(where the blue part is defined via step shown in Figure *. The green part is just the name of the CSV file).

For further reference, you may watch an online Open Link Virtuoso Walk-through / Tutorial https://www.youtube.com/watch?v=A8gVp1Wjmso.

1.2 Pros and Cons and Supplementary Solution

1.2.1 Pros

This is the most straightforward way.

If you want to customize the URIs of the properties, it only demands some skill of SPARQL to update the data. By using ISQL (which will be introduced further), you may combine multiple commands of SPARQL together.

1.2.2 Cons

At the Conductor interface , (afaik) you can not customize the URI for the entity types and the properties freely; modifications must be made using SPARQL. In addition, by default, all the values for any property are only literal instead of URI even if they are reconciled URLs already.

1.2.3 Supplementary Solution

To solve the issues from cons, follow these steps:

(1) Change/customize the URI of the entity

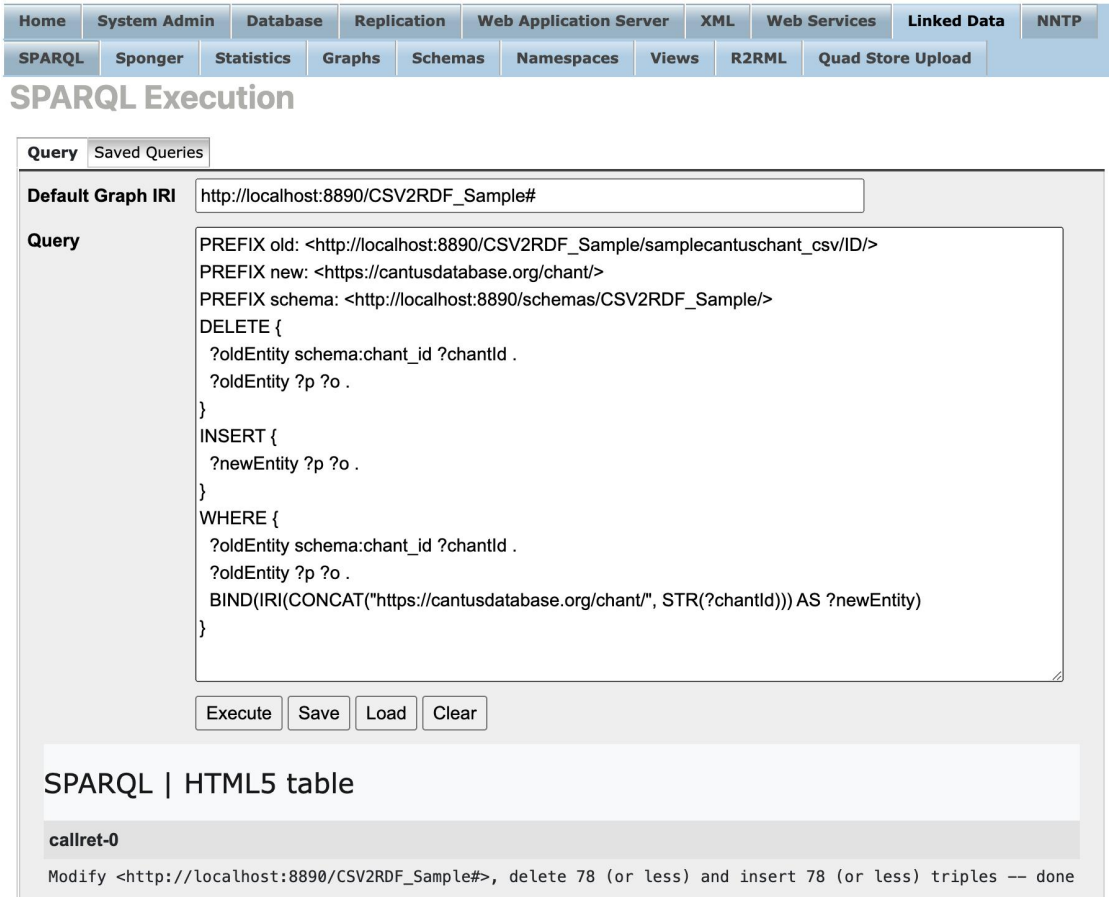
Navigate to tab Linked Data > SPARQL:

Fill in “Default Graph IRI” with “http://localhost:8890/CSV2RDF_Sample#” and execute:

```
PREFIX old: <http://localhost:8890/CSV2RDF_Sample/samplecantuschant_csv/ID/>
PREFIX new: <https://cantusdatabase.org/chant/>
PREFIX schema: <http://localhost:8890/schemas/CSV2RDF_Sample/>
DELETE {
  ?oldEntity schema:chant_id ?chantId .
  ?oldEntity ?p ?o .
}
INSERT {
  #?newEntity schema:chant_id ?chantId .
  ?newEntity ?p ?o .
```

```
}
WHERE {
  ?oldEntity schema:chant_id ?chantId .
  ?oldEntity ?p ?o .
  BIND(IRI(CONCAT("https://cantusdatabase.org/chant/", STR(?chantId)))
AS ?newEntity)
}
```

after execution it is as shown below:



(2) Export the graph and import it again

From ISQL(Interactive SQL, http://localhost:8890/conductor/isql.vspx), execute:

```
dump_one_graph ('http://localhost:8890/CSV2RDF_Sample#', './exported_dataset_',
1000000000);
```

(If is doesn't work, refer to https://github.com/DDMAL/linkedmusic-datalake/wiki/How-to-export-a-RDF-GRAPH-from-Virtuoso%3F for configuration.)

By executing above procedure, in your local virtuoso directory you will find the exported ".ttl" graph named "exported_dataset". Import the graph again to Virtuoso:

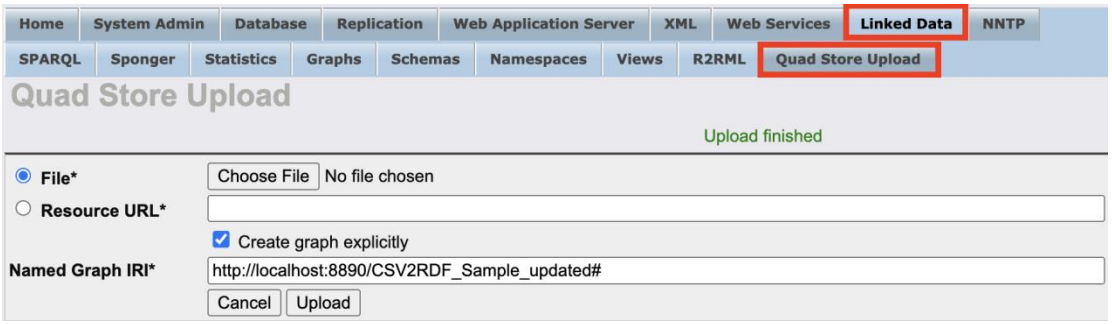


Figure 1

Please notice that we change the name of the graph to <http://localhost:8890/CSV2RDF_Sample_updated#> (just for contrast).

(3) Other adjustments (via ISQL):

Navigate to ISQL and execute the following code as a whole:

```
--①Delete triples with property <http://localhost:8890/schemas/CSV2RDF_Sample/chant_id>
SPARQL
DELETE WHERE {GRAPH <http://localhost:8890/CSV2RDF_Sample_updated#> {?s <http://localhost:8890/schemas/CSV2RDF_Sample/chant_id> ?o}};
--②Define the type for the entity
SPARQL
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
DELETE{GRAPH <http://localhost:8890/CSV2RDF_Sample_updated#> {?s rdf:type ?o}}
INSERT{GRAPH <http://localhost:8890/CSV2RDF_Sample_updated#> {?s rdf:type <https://cantusdatabase.org/chant/>}}
WHERE{GRAPH <http://localhost:8890/CSV2RDF_Sample_updated#> {?s rdf:type ?o}};
--③Change the properties
SPARQL
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
DELETE {GRAPH <http://localhost:8890/CSV2RDF_Sample_updated#>{?s <http://localhost:8890/schemas/CSV2RDF_Sample/incipit> ?o}}
```

```
INSERT {GRAPH <http://localhost:8890/CSV2RDF_Sample_updated#>{?s
wdt:P1922 ?o}}
WHERE {GRAPH <http://localhost:8890/CSV2RDF_Sample_updated#>{?s
<http://localhost:8890/schemas/CSV2RDF_Sample/incipit> ?o}};
SPARQL
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
DELETE{GRAPH <http://localhost:8890/CSV2RDF_Sample_updated#>{?s
<http://localhost:8890/schemas/CSV2RDF_Sample/genre> ?o}}
INSERT{GRAPH <http://localhost:8890/CSV2RDF_Sample_updated#>{?s
wdt:P136 ?o}}
WHERE{GRAPH <http://localhost:8890/CSV2RDF_Sample_updated#>{?s
<http://localhost:8890/schemas/CSV2RDF_Sample/genre> ?o}};
SPARQL
DELETE{GRAPH <http://localhost:8890/CSV2RDF_Sample_updated#>{?s
<http://localhost:8890/schemas/CSV2RDF_Sample/src_link> ?o}}
INSERT{GRAPH <http://localhost:8890/CSV2RDF_Sample_updated#>{?s
<https://cantusdatabase.org/sources> ?o}}
WHERE{GRAPH <http://localhost:8890/CSV2RDF_Sample_updated#>{?s
<http://localhost:8890/schemas/CSV2RDF_Sample/src_link> ?o}};
--④Change the strings to URI
SPARQL
DELETE{GRAPH <http://localhost:8890/CSV2RDF_Sample_updated#>{?s ?p ?o}}
INSERT{GRAPH
<http://localhost:8890/CSV2RDF_Sample_updated#>{?s ?p ?newO}}
WHERE{GRAPH
<http://localhost:8890/CSV2RDF_Sample_updated#>{?s ?p ?o.FILTER(STRSTARTS
(STR(?o), "http"))BIND(IRI(?o) AS ?newO)}};
```

After all the SPARQL procedures the URLs will be well customized.
However, it's still possible for refining the SPARQL code to be more concise.
Additionally, it remains to be tested how the complexity and workload will increase with the addition of more CSVs.

1.2.3 Future to Do

(Up to now, I put the “Future to Do” to the end of the article. I may adjust the framework and content eventually.)

2. ISQL of Terminal + R2RML

This method complements the previous one by allowing a direct customization of URIs, which the former method can not achieve without using SPARQL. Another RDF language--RDB to RDF Mapping Language, abbreviated as R2RML, is a W3C standard that enables this customization. To use R2RML, you have to install a package called rdb2rdf_dav.vad. (Later I upload it on GitHub)

2.1 Demonstration

2.1.1 Create a .sql file for R2RML

Create a document with a .sql suffix, where the R2RML procedure is defined. Please refer to How to use R2RML, where it prepares a basic.sql test file for example. Adjust the R2RML code to match your CSV file content. We still use “CSV2RDF_Sample.DBA.sampleCantusChant_csv” aforementioned as an example.

Prepare the content of a file named “cantusR2RML.sql”:

```
--the Name of the R2RML Graph:
SPARQL CLEAR GRAPH <http://temp/product>;
--the Name of the Graph:
SPARQL CLEAR GRAPH <http://example.com/trial>;
DB.DBA.TTLP ('
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix cantusDB: <https://cantusdatabase.org/> .
@prefix schema: <http://schema.org/> .
@prefix wdt: <http://www.wikidata.org/prop/direct/> .
<http://example.com/ns#TriplesMap1>
    a rr:TriplesMap;
    rr:logicalTable
    [
      rr:tableSchema "CSV2RDF_Sample";
```

```
        rr:tableOwner "DBA";
        rr:tableName "sampleCantusChant_csv"
    ];
    rr:subjectMap
    [
        rr:template "https://cantusdatabase.org/chant/{Chant_ID}";
        rr:class cantusDB:chant;
        rr:graph <http://example.com/trial>;
    ];
    rr:predicateObjectMap
    [
        rr:predicate wdt:P1922;
        rr:objectMap [ rr:column "incipit" ];
    ];
    rr:predicateObjectMap
    [
        rr:predicate wdt:P136;
        rr:objectMap [ rr:column "genre" ];
    ];
    rr:predicateObjectMap
    [
        rr:predicate cantusDB:sources;
        rr:objectMap [ rr:column "src_link" ];
    ];
.
', 'http://temp/product', 'http://temp/product' )
;
exec ('sparql ' || DB.DBA.R2RML_MAKE_QM_FROM_G ('http://temp/product'));

SPARQL
SELECT * FROM <http://example.com/trial>
WHERE {?s ?p ?o .};
```

You can see The R2RML script is included as a parameter of the `DB.DBA.TTLP()` function.

2.1.2 Execute the .sql script

Please Locate to the bin folder of Virtuoso, and refer to Virtuoso-Setup-Guide to open “ISQL CLI”, where it begins with “SQL>”.

Then place the “cantusR2RML.sql” script in the bin folder. Then load cantusR2RML.sql.

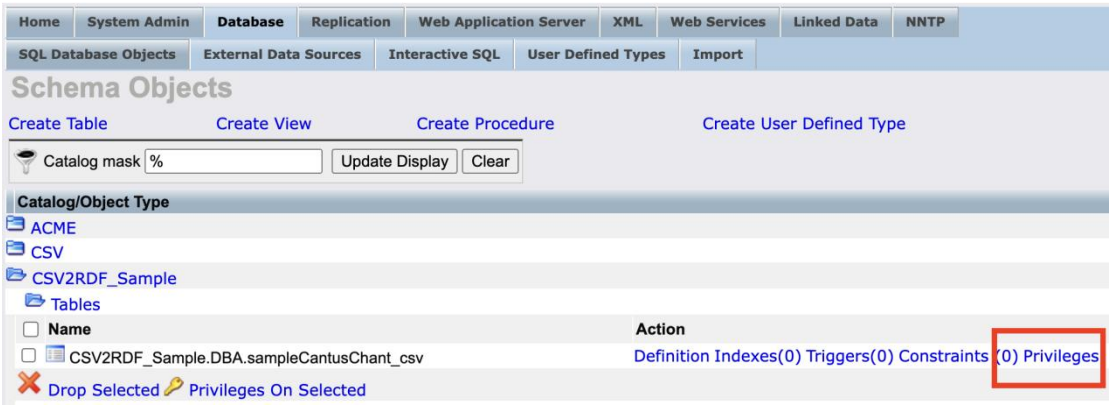
Eg.(Terminal):

```
root@9e73ea930b6a:/opt/virtuoso-opensource/bin# isql -U dba -P mysecret
OpenLink Virtuoso Interactive SQL (Virtuoso)
Version 07.20.3240 as of Jun 10 2024
Type HELP; for help and EXIT; to exit.
Connected to OpenLink Virtuoso
Driver: 07.20.3240 OpenLink Virtuoso ODBC Driver
SQL> load cantusR2RML.sql;
```

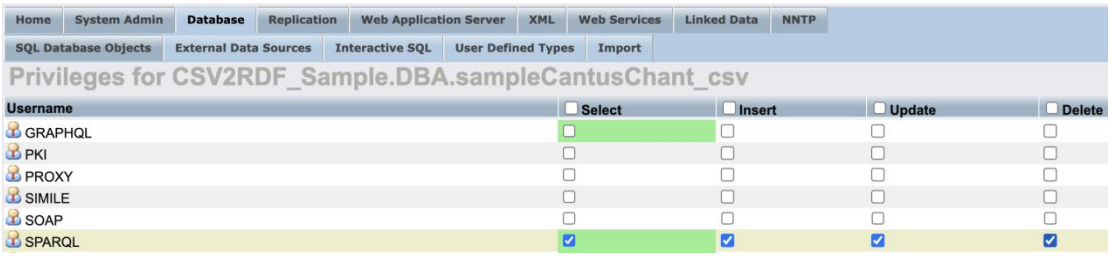
Attention: As shown in the result of execution, the linked data seems stored in “http://example.com/trial” named graph. However, that CSV2RDF process just only provided a virtual, readable-only view; the converted data is not put in the rdfstore(Linked Data). So you may not access the graph in a SPARQL endpoint yet, until you proceed with next step.

2.1.3 Give Privileges to the CSV file for SPARQL query.

Navigate to tab Database > SQL Database Objects. Find “Privileges” and click on that:



and then tick all the check boxes for the Username SPARQL, then save changes:



Then you can definitely query the converted RDF data in the graph http://example.com/trial.

2.2 Summary, Pros and Cons

This method is initially more intended for converting a Relational Database(RDB) with Schema to RDF with Ontology.

In other words: **any imported CSV is actually regarded as a single spreadsheet of an RDB. Importing a CSV to Virtuoso's LinkedData is essentially no different from importing a RDB to that of Virtuoso.**

2.2.1 Pros

(1)This method allows the customization of URIs by directly editing R2RML.

(2) It is advantageous for converting relational databases to RDF and can presumably handle multiple CSV files more efficiently.

(3)In addition, R2RML is a standard of W3C, generic and understood by various tools, including Chat GPT.

2.2.2 Cons

(1)The .sql script may involve a hybrid code of SQL, SPARQL, R2RML, some Virtuoso-specific function, which means a certain complexity. (2)Presumably also due to generating a “transient view”, the graph cannot be exported as RDF smoothly unless you take a workaround by exporting the graph in form of CSV from the SPARQL endpoint:

(1)Refer to Figure 0 and switch the value of Results Format to CSV. (2)Execute `select * where {?s ?p ?o}` and you get the downloaded CSV file. (3) The data structure of the CSV file is the same as N-triples matching the {?s ?p ?o} pattern. This makes it straightforward to convert the CSV data into a standard RDF N-triples file.

3. Sponger of Virtuoso

Regarding use of Sponger, please refer to Virtuoso-Setup-Guide, (scroll down to the bottom) where it primarily illustrates the function of Sponger.

3.1 Demonstration

3.1.1 Locate the CSV File

Ensure that a CSV file(we also use the “sampleCantusChant.csv” as an example) be accessible via a URL, directly, such as rendering it from GitHub's "Raw" display of a CSV file:
https://raw.githubusercontent.com/DDMAL/linkedmusic-datalake/main/cantusdb/sampleCantusChant.csv

3.1.2 Prepare the Sponging Request

On SPARQL interface, input:

```
DEFINE get:soft "soft"
DEFINE input:grab-all "yes"
DEFINE input:grab-limit 1
LOAD <https://raw.githubusercontent.com/DDMAL/linkedmusic-datalake/main/cantusdb/sampleCantusChant.csv> INTO GRAPH <http://your-virtuoso-instance/graph/for/csv/data>
```

3.1.3 Inspection with SPARQL

When the query above is executed, the Sponger will retrieve the CSV file and process it. The Sponger uses a built-in CSV parser to convert the file into RDF triples. It will automatically generate a URI for each row and map columns to properties based on the headers.
At the same time, it will generate lots of metadata(you can ignore it.) in form of triples in the GRAPH <http://your-virtuoso-instance/graph/for/csv/data>.
You can review the data by SPARQL but will find the URI not in a custom way.
Just use SPARQL to modify the URI as you will, which is aforementioned already.

3.2 Pros and Cons

This method might be suitable for incorporating data in a immediate but fragmented manner. Its cons are similar to using “Virtuoso Conductor + SPARQL”.

4. Custom Scripts, using a RDFLib package of Python

4.1 Demonstration

This method distinctively replies on a package for Python called “RDFLib”(Lib is short for Library) as shown: RDBLib.

With this package as well as a decent python programming skill, one can convert many CSVs to any RDF format.

Taking a conversion for MusicBrainz into RDF for example(merge corresponding CSVs of different Entities into one RDF, see csv2rdf):

Please refer to README.md for more details. The general coding procedures are:

4.1.1 Prepare the CSV files

Every CSV represents an entity in a RDB. The first column is the instance, the other columns are the values of properties. Our code is based on this structure.

4.1.2 Mapping for entities and properties

Use a Python script(see get_relations.py) to generate a “mapping.json”, then fill in the mapped URI for the entities and properties, which is a manual task.

e.g.:

```
{
    "area_id": "https://www.wikidata.org/wiki/Property:P982",
    "sort-name": "https://www.wikidata.org/wiki/Property:P2561",
    "relations_wiki": "https://www.wikidata.org/wiki/Property:P1687",
    "type": "https://www.wikidata.org/wiki/Property:P2308",
    "annotation": "https://www.wikidata.org/wiki/Property:P2916",
    "name": "https://www.wikidata.org/wiki/Property:P2561",
    "disambiguation": "https://schema.org/disambiguatingDescription",
    "type-id": "https://musicbrainz.org/doc/Label/Type",
    "genres_name": "https://www.wikidata.org/wiki/Property:P136",
    "genres_id": "https://www.wikidata.org/wiki/Property:P8052",
    "entity_type": [
        "https://www.wikidata.org/wiki/Q11500",
        "https://www.wikidata.org/wiki/Q1656682",
        "https://www.wikidata.org/wiki/Q34379",
```

```
            "https://www.wikidata.org/wiki/Q18127",
            "https://www.wikidata.org/wiki/Q155171"
        ],
        "event_id": "https://www.wikidata.org/wiki/Property:P6423"
}
```


--this is just like a simple ontology for the reconciliation.

4.1.3 Generate RDF in turtle format

Execute another python script with RDFLib(see csv2rdf_single_subject.py) to generate the turtle data called “out_rdf.ttl”.

Example execution code in terminal:

`python3 csv2rdf_single_subject.py mapping.json area.csv artist.csv genre.csv recording.csv` ...

--the first parameter is the name of the python script, the second parameter is the mapping.json script, as of the third parameter, all the other are file names for each individual CSV files.

4.1.4 Import the out_rdf.ttl to Virtuoso.

Please refer to Figure 1 aforementioned.

4.2 Pros and Cons

(1) It demands a certain proficiency of python programming but may be universal or flexible in a way.

(2) The mapping procedure is still a bit manual.(Taking the mapping of MusicBrainz for example: MusicBrainz/mapping.json)

(3) In terms of RDB2RDF, this method is less efficient than “ISQL of Terminal + R2RML” aforementioned.

(4) By the end of Oct 2024, we recognized this method had to be refined timely with its code in order to accommodate different occasions where the original schema may impose constraints, such as the data format, management on a “relationship set” in which some attributes are describing the relationship between more than 2 coexistent entities.

5. Other Methods to complement or for reference

There are some other methods which are not directly capable of CSV2RDF but could be of complementary value.

5.1 Bulk Loader

This is not an independent method with respect to CSV2RDF.

If you ask Chat GPT "How to use Bulk Loader to convert a CSV to RDF?", it may cause hallucination. (See virtuoso-opensource/issues/1303) Actually, we can either use Bulk Loader to upload lots of CSV files simultaneously or lots of RDF files simultaneously, but not for conversion of CSV2RDF. So this method can be a supplement for “Virtuoso Conductor + SPARQL” or other cases where only one CSV can be uploaded at a time.

Please refer to the official tutorial Virtuoso CSV File Bulk Loader.

Here we give some tips:

(1)Before any motion, please run the scripts shown under the headline “CSV Bulk Loader scripts” on the official tutorial(If it reports errors upon execution, please separate the scripts by the unindented ";" then execute them consecutively).

(2)Note the saying of "*Note: Before reloading a CSV file, its entry must be removed from the...*" in the tutorial.

> You can execute the DELETE statement to remove the entry:

`DELETE FROM "DB.DBA.csv_load_list" WHERE cl_file = 'the entry of cl_file column of the CSV file';`

> --It’s required that the value of cl_file be single-quoted.

Note: If you ask ChatGPT about CSV2RDF into Virtuoso. It will always answer with Bulk Loader as one of the solutions. Afaik, ...

5.2 Prompting Engineering based on an ontology for the CSV(s)

This thinking is actually a reference for the method “Custom Scripts, using a RDFLib package of Python”. As we mentioned, the mapping.json is a bit similar to an ontology, therefore, If we have an existing ontology for CSV(s), we can feed it as a context to LLMs to prompt it to generate code for CSV2RDF.

Take aforementioned sampleCantusChant.csv as example, the corresponding ontology excerpt may be:

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix wdt: <http://www.wikidata.org/prop/direct/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix cantusDB: <https://cantusdatabase.org/> .

wdt:P136 rdf:type owl:ObjectProperty ;
            rdfs:domain cantusDB:chant ;
            rdfs:label "genre" .
cantusDB:sources rdf:type owl:ObjectProperty ;
                        rdfs:domain cantusDB:chant ;
                        rdfs:label "src_link" .
wdt:P1922 rdf:type owl:DatatypeProperty ;
              rdfs:domain cantusDB:chant ;
              rdfs:label "incipit" .
cantusDB:chant rdf:type owl:Class ;
                    rdfs:label "Chant" .
```

--it is an OWL file which is also an RDF file and in turtle format.
What worth mentioning is, the generated code sometimes also uses RDFLib package.
The potential pros are, if the relationship between several CSV files can be embodied in one ontology, the generated code will be well customized for different cases, and it doesn't demand a decent mastery of python for the manager. The LLM such as Chat GPT knows the structure of
The cons are obvious: If there is no predefined ontology, it's impossible to work.

5.3 Virtuoso PL Procedures

Please refer to virtuoso-opensource/issues/1305.
> Basically, this method will use a Virtuoso's Procedural Language (PL), which is a SQL-based language used for creating stored procedures, functions, and triggers

within the Virtuoso Universal Server. This language allows you to define complex operations and workflows that can be executed on the Virtuoso database.
It's a virtuoso-specific language.

5.4 RDF Generators

This is just a solution told casually by Chat GPT. According to this clue, please refer to:

*

(1)[digitalbuildings/tools/rdf_generator/](https://google.github.io/digitalbuildings/tools/rdf_generator/) which still uses the RDFLib.

*

(2)[Category:RDF_Generator](https://www.w3.org/2001/sw/wiki/Category:RDF_Generator)

*

(3)https://github.com/klotzbenjamin/rdf-generator, which still uses RDFLib package + Python + ontology. It may be fundamentally no different from our current Method “Custom Scripts...”. What's worth mentioning is, an ontology can be used by an RDFLib package so as to render the mapping of properties and type of entities.

*

(4)Chat GPT took “csv2rdf4lod” as an example which however, involves use of Java.

5.5 Others

Such as “RDFizers” “Java + R2RML to generate RDF”.

Yet as you refer to Virtuoso HowTo: Generating Linked Data from CSV version of Northwind Database, you may find that:

CSV2RDF still relies on SQL, Sponger, R2RML.

Therefore, there might not be other individual method of “CSV2RDF into Virtuoso”.

6. Conclusion and Outlook

The methods above have their respective advantages and defects.

Some of the methods are potentially complementary to others(like Bulk Loader to Virtuoso Conductor).Some are possibly not independent (like Virtuoso PL Procedures).

One key point is whether the conversion is done internally or externally. The External methods are like Custom Scripts using python and RDBLib package([Method II]) and RDF Generators(e.g., csv2rdf4lod). Others are basically internal(or Virtuoso-Specific to some extent).

Above all, we just propose the fore-mentioned first 3 methods(which also permeate/relate the rest of the methods in this article), they have their own unique characteristics and are independent of one another:

[Method I] (Internal)
It is a little W3C&Virtuoso-specific and going through R2RML but is friendly for those who don't excel at python programming.
Probably, with use of ontology, this method would be more automatic. Till July, 2024, this method is worth trying deep.
On the end of Oct, 2024, we have further experimented on importing more than 1 reconciled CSV files onto Virtuoso, which files have foreign key reference. It means that we can import various reconciled CSV files onto Virtuoso and "assemble them into a RDB(Relational Database)". So that with the facilities for RDB2RDF in Virtuoso, we may greatly improve our efficiency of "arbitrary conversion to RDF". (Refer to https://github.com/DDMAL/linkedmusic-datalake/issues/214)

[Method II] (External)
It demands a certain proficiency of python programming but may be universal or flexible in a way.
The mapping procedure is still a bit manual.(Taking the mapping of MusicBrainz for example: MusicBrainz / mapping.json)
This method is still favourable regardless of any schema indications from a relational database. However, most of the reconciled CSV files are still extracted as a part of a relational database which has a schema. So I believe we will rely less on this method as we have more mastery of [Method I].

[Method III]

It's specially suitable for (see above "incorporating data in a immediate but fragmented manner")...

But you need to customize the URI of entities using SPARQL.

6.1 Why we are currently (by the end of Oct, 2024) using RDFLib + Python + Conductor method?

The sentence below can partially explain it.

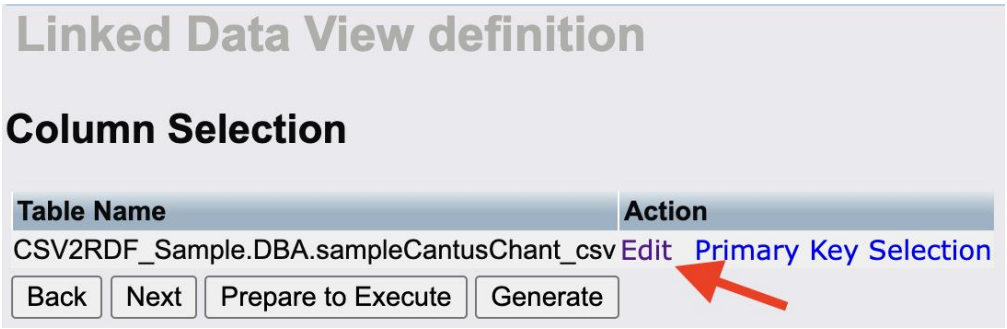
6.2 Ongoing or Future Work

Our initial phase of research aimed to avoid relying on the ontology. In the future, we may not be confined to the thinking, as we have already been witnessing the in-erasable value of ontology especially in terms of (1) NLQ2SPARQL with Chat GPT and (2) knowledge reasoning.

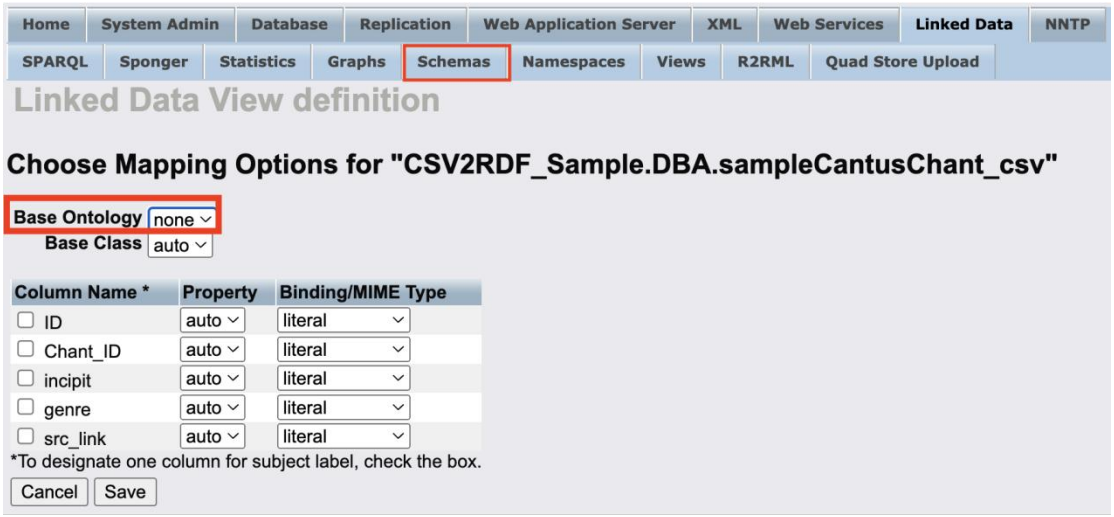
6.2.1 Virtuoso Conductor + SPARQL

Afaik, There is a common issue with Method 6.2.1 and 6.2.2. That is you can not export the generated RDF graph normally. In aforementioned procedures we actually adopted some relatively “flexible” (or stopgap) measure; though simpler methods may exist.

There are still a lot of options to be explored on Conductor Interface, especially this:



click on Edit to see:



--we may import different ontologies from tab Schemas for customization of URIs.

6.2.2 ISQL of Terminal + R2RML

It's also worth mentioning that we could place multiple CSV files in an RDB, allowing us to use a single R2RML script to facilitate the customization of properties across many CSV files. We have experimented with importing multiple CSV files that have foreign key relationships and test the conversion process.

Please see the sample data and the corresponding R2RML scripts on Issue 214 (https://github.com/DDMAL/linkedmusic-datalake/issues/214),

from which there are 2 points worth mentioning:

(1) You don't have to fill in the primary key field with a complete URI for each entries because it can be customized in R2RML code such as:

```
rr:subjectMap
    [
      rr:template "https://cantusdatabase.org/chant/{Chant_ID}";
      rr:class cantusDB:chant;
      rr:graph <http://example.com/trial>;
    ];
```

(2) This method actually did a mapping of foreign key reference, such as:

```
rr:predicateObjectMap
    [
      rr:predicate cantusDB:sources;
      rr:objectMap [
        rr:parentTriplesMap <http://example.com/ns#TriplesMap2>;
        rr:joinCondition [
```

```
            rr:child "src_link";
            rr:parent "Source_ID";
        ];
    ];
];
```

Others:
After installing the package rdb2rdf_dav.vad, a new R2RML tab appears under Linked Data, which is worth investigating further.

6.3 General Conclusion

For determining the best method for CSV2RDF, currently [Method I] and [Method II] may end up in a tie or being complementary to each other. However, CSV2RDF can be viewed as partial (preliminary) task of RDB2RDF; in practice, we are actually confronting various RDB2RDF tasks. It's just because it's more feasible for reconciliation against CSV files before uploading to Virtuoso that we work around to CSV2RDF. When we managed to upload more than one CSV files within which there exist foreign key references, it essentially mirrors uploading a reconciled RDB to Virtuoso.

Furthermore, in the past practice of CSV2RDF using [Method II], we realized referencing the schema could be helpful. Hence, it's reasonable and beneficial that we can also transfer any RDB entirely to Virtuoso before carrying out reconciliation. Additionally, during RDB2RDF on Virtuoso, an ontology can be generated from the corresponding schema, which aids in prompting Chat GPT to realize NLQ2SPARQL.

Overall, we believe as we delve further into uploading RDB2Virtuoso, and working with R2RML, probably [Method I] will prove more effective, though [Method II] is to serve in a complementary and flexible way.