

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 reconciliation with Wikidata using OpenRefine which works well with CSV formats.

Assume: 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, it's accessible at <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.

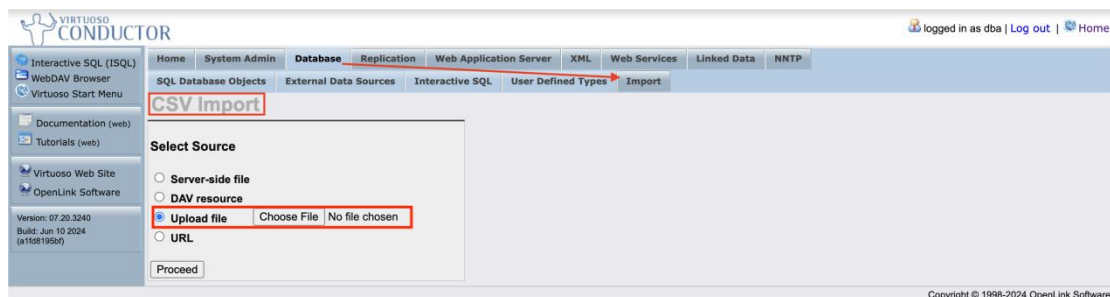
We demonstrate step by step:

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:

VIRTUOSO CONDUCTOR

Home System Admin **Database** Replication Web Application Server XML Web Services Linked Data NNTP

SQL Database Objects External Data Sources Interactive SQL User Defined Types Import

CSV Import

Select parse options

Mode: Strict
 Delimiter: ,
 Quote: Double
 Encoding: Default
 Header Row: 1 zero if no header
 Import from line: 2

Sample Data

#	Chant_ID	incipit	genre	src_link
2	562633	Ecce nunc palam loqueris et	http://www.wikidata.org/entity/Q582093	https://cantusdatabase.org/source/123756
3	671551	Dominus tamquam ovis ad victimam	http://www.wikidata.org/entity/Q582093	https://cantusdatabase.org/source/669163
4	562160	Seniores populi consilium fecerunt ut	http://www.wikidata.org/entity/Q604748	https://cantusdatabase.org/source/123756
5	467250	Ad te domine levavi animam	http://www.wikidata.org/entity/Q582093	https://cantusdatabase.org/source/123730
6	560736	Stephanus servus dei quem lapidabant	http://www.wikidata.org/entity/Q604748	https://cantusdatabase.org/source/123756
7	557534	Si diligis me Simon Petre	http://www.wikidata.org/entity/Q604748	https://cantusdatabase.org/source/123716
8	561787	Pater Abraham miserere mei et	http://www.wikidata.org/entity/Q582093	https://cantusdatabase.org/source/123756
9	670564	Sit nomen domini benedictum in	http://www.wikidata.org/entity/Q582093	https://cantusdatabase.org/source/669163
10	561208	Naturae genitor conserva morte redemptos	http://www.wikidata.org/entity/Q582093	https://cantusdatabase.org/source/123756
11	670715	Dives ille guttam aquae petiit	http://www.wikidata.org/entity/Q582093	https://cantusdatabase.org/source/669163

Back Refresh Next

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

VIRTUOSO CONDUCTOR

Home System Admin **Database** Replication Web Application Server XML Web Services Linked Data NNTP

SQL Database Objects External Data Sources Interactive SQL User Defined Types Import

CSV Import

Confirm Table details

Table Qualifier: CSV2RDF_Sample
 Table Owner: DBA
 Table Name: sampleCantusChant_cs

Columns

ID: INTEGER IDENTITY PRIMARY KEY
 Chant_ID: INTEGER
 incipit: VARCHAR
 genre: VARCHAR
 src_link: VARCHAR

☒ Auto-commit mode

Back Create Table & Import Data

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

VIRTUOSO CONDUCTOR

Home System Admin **Database** Replication Web Application Server XML Web Services Linked Data NNTP

SQL Database Objects External Data Sources Interactive SQL User Defined Types Import

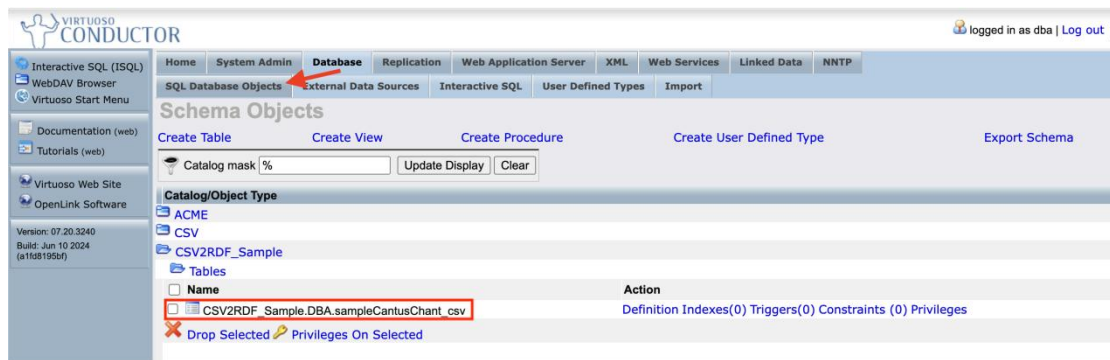
CSV Import

Status

The data is imported in table [CSV2RDF_Sample.DBA.sampleCantusChant_csv](#).
 Total records imported 13.

Start Over

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:

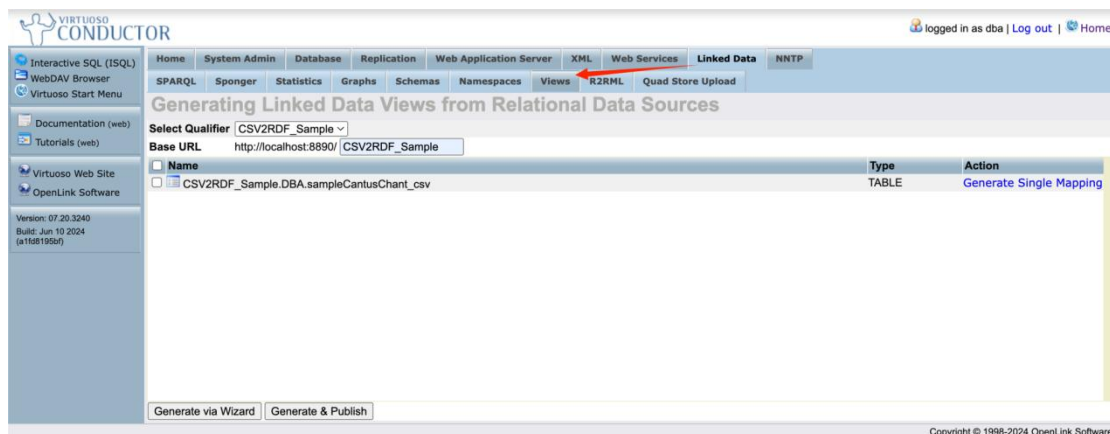
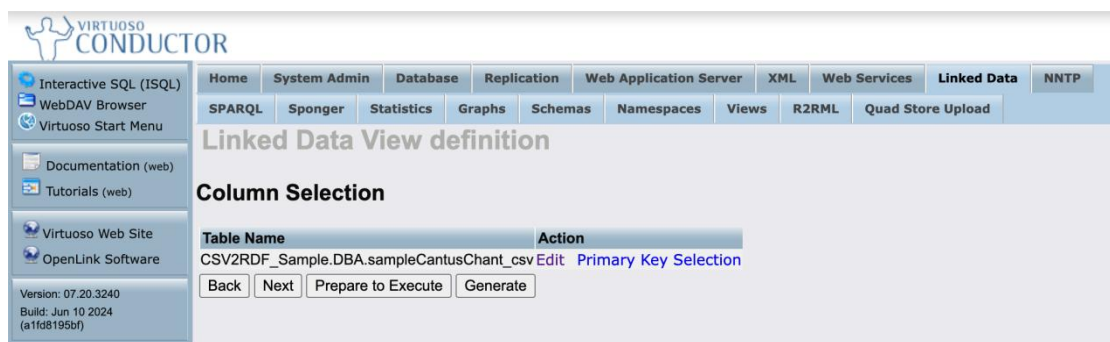


Figure *

From the drop-down list of “Select Qualifier”, choose the value of “CSV2RDF_Sample”. Then you can customize Base URL. 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:

VIRTUOSO CONDUCTOR logged in as dba

Home System Admin Database Replication Web Application Server XML Web Services **Linked Data** NNTP

SPARQL Sponger Statistics Graphs Schemas Namespaces Views R2RML Quad Store Upload

Linked Data View definition

Definitions

```
grant select on "CSV2RDF_Sample"."DBA"."sampleCantusChant_csv" to SPARQL_SELECT;

SPARQL
prefix CSV2RDF_Sample: <http://localhost:8890/schemas/CSV2RDF_Sample/>
create iri class CSV2RDF_Sample:samplecantuschant_csv
"http://^URIQADefaultHost~/CSV2RDF_Sample/samplecantuschant_csv/ID/%d#this" (in_ID integer not null) . ;

create view "CSV2RDF_Sample"."DBA"."sampleCantusChant_csvCount" as select count (*) as cnt from
"CSV2RDF_Sample"."DBA"."sampleCantusChant_csv";
grant select on "CSV2RDF_Sample"."DBA"."sampleCantusChant_csvCount" to SPARQL_SELECT;
drop view "CSV2RDF_Sample"."DBA"."CSV2RDF_Sample_Total";
create view "CSV2RDF_Sample"."DBA"."CSV2RDF_Sample_Total" as select (cnt0*cnt1) AS cnt from
(select count(*) cnt0 from "CSV2RDF_Sample"."DBA"."sampleCantusChant_csv") tb0,
(select count(*)+1 as cnt1 from DB.DBA.TABLE_COLS where "TABLE" = 'CSV2RDF_Sample.DBA.sampleCantusChant_csv' and
"COLUMN" <> '_IDN') tb1
;
grant select on "CSV2RDF_Sample"."DBA"."CSV2RDF_Sample_Total" to SPARQL_SELECT;
```

R2RML Graph

```
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix CSV2RDF_Sample: <http://localhost:8890/schemas/CSV2RDF_Sample/> .
@prefix csv2rdf_sample-stat: <http://localhost:8890/CSV2RDF_Sample/stat#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix void: <http://rdfs.org/ns/void#> .
@prefix scovo: <http://purl.org/NET/scovo#> .
@prefix aowl: <http://bblfish.net/work/atom-owl/2006-06-06/> .

<#TriplesMapsampleCantusChant_csv> a rr:TriplesMap; rr:logicalTable [ rr:tableSchema "CSV2RDF_Sample" ; rr:tableOwner "DBA"
; rr:tableName "sampleCantusChant_csv" ] ;
rr:subjectMap [ rr:termType rr:IRI ; rr:template "http://localhost:8890/CSV2RDF_Sample/samplecantuschant_csv/ID/{ID}#this";
rr:class CSV2RDF_Sample:sampleCantusChant_csv; rr:graph <http://localhost:8890/CSV2RDF_Sample#> ] ;
rr:predicateObjectMap [ rr:predicateMap [ rr:constant CSV2RDF_Sample:id ] ; rr:objectMap [ rr:column "ID" ] ; ] ;
rr:predicateObjectMap [ rr:predicateMap [ rr:constant CSV2RDF_Sample:chant_id ] ; rr:objectMap [ rr:column "Chant_ID" ] ; ] ;
rr:predicateObjectMap [ rr:predicateMap [ rr:constant CSV2RDF_Sample:incipit ] ; rr:objectMap [ rr:column "incipit" ] ; ] ;
rr:predicateObjectMap [ rr:predicateMap [ rr:constant CSV2RDF_Sample:genre ] ; rr:objectMap [ rr:column "genre" ] ; ] ;
rr:predicateObjectMap [ rr:predicateMap [ rr:constant CSV2RDF_Sample:src_link ] ; rr:objectMap [ rr:column "src_link" ] ; ] .
```

Ontology

```
gql:type gql:Scalar ;
rdfs:domain CSV2RDF_Sample:sampleCantusChant_csv ;
rdfs:isDefinedBy CSV2RDF_Sample: ;
gql:field gql:genre ;
rdfs:label "genre" .

CSV2RDF_Sample:src_link a owl:DatatypeProperty ;
rdfs:range xsd:string ;
gql:type gql:Scalar ;
rdfs:domain CSV2RDF_Sample:sampleCantusChant_csv ;
rdfs:isDefinedBy CSV2RDF_Sample: ;
gql:field gql:src_link ;
rdfs:label "src_link" .

gql:Map gql:dataGraph <http://localhost:8890/CSV2RDF_Sample#> ;
gql:schemaGraph CSV2RDF_Sample: ;
gql:schemaObjects gql:sampleCantusChant_csv.
```

Default Physical Named Graph urn:localhost:8890:CSV2RDF_Sample

☐ Generate RDB2RDF triggers

☐ Enable Data Syncs with Physical Quad Store

Start Over Apply Changes **Execute** Save Data Mappings Save Ontology Mappings

WebDAV location: WebDAV Browser Save Data Mappings

WebDAV location: WebDAV Browser Save Ontology Mappings

There are 3 sections above: (1) Within “Definitions” 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”:

Execution Status

Status	Message
00000	Quad map <http://localhost:8890/schemas/CSV2RDF_Sample/qm-samplecantuschant_csv> is no longer used in storage <http://www.openlinksw.com/schemas/virtrdf#DefaultQuadStorage>
00000	Quad map <http://localhost:8890/schemas/CSV2RDF_Sample/qm-samplecantuschant_csv> is deleted
00000	Transaction committed, SPARQL compiler re-configured
00000	Transaction committed, SPARQL compiler re-configured
00000	2 RDF metadata manipulation operations done
00000	Quad map <http://localhost:8890/schemas/CSV2RDF_Sample/qm-VoidStatistics> is no longer used in storage <http://www.openlinksw.com/schemas/virtrdf#DefaultQuadStorage>
00000	Quad map <http://localhost:8890/schemas/CSV2RDF_Sample/qm-VoidStatistics> is deleted
00000	Transaction committed, SPARQL compiler re-configured
00000	Transaction committed, SPARQL compiler re-configured
00000	2 RDF metadata manipulation operations done
00000	Previous definition of IRI class <http://localhost:8890/schemas/CSV2RDF_Sample/samplecantuschant_csv-nullable> is identical to the new one, not touched
00000	Previous definition of IRI class <http://localhost:8890/schemas/CSV2RDF_Sample/samplecantuschant_csv> has been dropped
00000	Previous definition of IRI class <http://localhost:8890/schemas/CSV2RDF_Sample/samplecantuschant_csv-nullable> is identical to the new one, not touched
00000	Previous definition of IRI class <http://localhost:8890/schemas/CSV2RDF_Sample/samplecantuschant_csv> has been dropped
00000	IRI class <http://localhost:8890/schemas/CSV2RDF_Sample/samplecantuschant_csv> has been defined (inherited from rdfs:sql-integer-uri)

SQL Relations (Tables) to RDF Statements (Predicate / Property Graph) Mappings

http://localhost:8890/schemas/CSV2RDF_Sample/qm-samplecantuschant_csv
http://localhost:8890/schemas/CSV2RDF_Sample/qm-VoidStatistics

Sample Graph IRIs & Linked Data Entity URIs

RDF Document (Named Graph) IRIs:

Transient Views: http://localhost:8890/CSV2RDF_Sample#
http://localhost:8890/CSV2RDF_Sample/samplecantuschant_csv/ID/1#this
http://localhost:8890/CSV2RDF_Sample/stat#
http://localhost:8890/CSV2RDF_Sample/stat#Stat
Metadata Data Document (VoID) URI/URL: http://localhost:8890/CSV2RDF_Sample/stat#
Linked Data Ontology URI: http://localhost:8890/schemas/CSV2RDF_Sample/

[Start Over](#) [Back](#)

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:>

SPARQL Query Editor About Tables ▾

Conductor Facet Browser Permalink

Extensions: cxml save to dav sponge User: SPARQL

Default Data Set Name (Graph IRI)

http://localhost:8890/CSV2RDF_Sample#

Query Text

select * where{?s ?p ?o}

Results Format

Execute Query Res

Sponging

Execution timeout

- Auto
- ✓ HTML
- HTML (Faceted Browsing Links)
- Spreadsheet
- XML
- JSON
- JavaScript
- Turtle
- RDF/XML
- N-Triples
- CSV
- TSV

Figure 0

Note that the URI of each instance/entity is defaulted like this:

http://localhost:8890/CSV2RDF_Sample/samplecantuschant_csv/ID/1#this

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

For further reference, please watch an online Open Link Virtuoso Walk-through / Tutorial <https://www.youtube.com/watch?v=A8gVp1Wjms0>.

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 requires using 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(URI(CONCAT("https://cantusdatabase.org/chant/", STR(?chantId)))
AS ?newEntity)
}

```

after execution it is as shown below:

Home	System Admin	Database	Replication	Web Application Server	XML	Web Services	Linked Data	NNTP
SPARQL	Sponger	Statistics	Graphs	Schemas	Namespaces	Views	R2RML	Quad Store Upload

SPARQL Execution

Query Saved Queries

Default Graph IRI

http://localhost:8890/CSV2RDF_Sample#

Query

```

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

```

Execute

Save

Load

Clear

SPARQL | HTML5 table

callret-0

Modify <http://localhost:8890/CSV2RDF_Sample#>, delete 78 (or less) and insert 78 (or less) triples -- done

(2) Export the graph and import it again

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

```

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

```

(If it 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 `rd2rdf_dav.vad`. (I uploaded it in the same path where this document is stored 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>”. Place the “cantusR2RML.sql” script in the bin folder. Then load cantusR2RML.sql.
E.g. (Terminal):

...

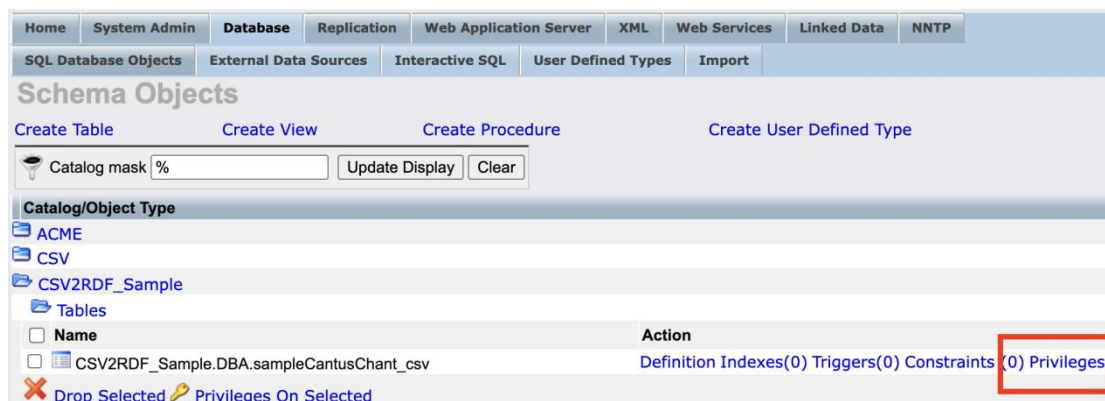
```
root@9e73ea930b6a:/opt/virtuoso-opensource/bin# isql -U dba -P mysecret
OpenLink Virtuoso Interactive SQL (Virtuoso)
Version 07.20.3240 as of Jun 10 2024
Type HELP; for help and EXIT; to exit.
Connected to OpenLink Virtuoso
Driver: 07.20.3240 OpenLink Virtuoso ODBC Driver
SQL> load cantusR2RML.sql;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

...

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:

Home	System Admin	Database	Replication	Web Application Server	XML	Web Services	Linked Data	NNTP
SQL Database Objects	External Data Sources	Interactive SQL	User Defined Types	Import				
Privileges for CSV2RDF_Sample.DBA.sampleCantusChant_csv								
Username	<input type="checkbox"/> Select	<input type="checkbox"/> Insert	<input type="checkbox"/> Update	<input type="checkbox"/> Delete				
GRAPHQL	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
PKI	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
PROXY	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
SIMILE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
SOAP	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
SPARQL	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>				

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 URI customization with direct R2RML edition.
- (2) It is advantageous for converting relational databases to RDF and can 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 mix of SQL, SPARQL, R2RML, some Virtuoso-specific function, which adds 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 set the Results Format to CSV. (2)Execute `select * where {?s ?p ?o}` and get the downloaded CSV file. (3) The CSV data structure matches N-triples in the {?s ?p ?o} pattern, making it easy to convert to a standard RDF N-triples file.

3. Sponger of Virtuoso

On using Sponger, please refer to [Virtuoso-Setup-Guide](#) (scroll down to the bottom), where its functions are outlined.

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 URL, such as the GitHub's "Raw" display:

https://github.com/DDMAL/linkedmusic-datalake/blob/main/csv2rdf/documentation_HowToConvertCSV2RDFIntoVirtuoso/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, Sponger retrieves 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 needed, which is aforementioned already.

3.2 Pros and Cons

This method is suitable for quick, fragmented data incorporation. Its cons are similar to using “Virtuoso Conductor + SPARQL”.

4. Custom Scripts, using a RDFLib package of Python

4.1 Demonstration

This method distinctively relies on Python's "[RDFLib](#)" (Lib is short for Library) package.

With programming skills, 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 RDB entity. 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 requires 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](https://github.com/ldf/virtuoso-opensource/issues/1303)) Actually, we can use Bulk Loader to upload lots of either CSV files or 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 possible methods for CSV2RDF into Virtuoso. It will always answer with Bulk Loader as one of the solutions. Afaik, it's not an independent approach.

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 require 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/1305](https://virtuoso-opensource.com/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 discussed above each have distinct advantages and limitations. Some are potentially complementary to others (e.g., Bulk Loader to Virtuoso Conductor). Some are possibly not independent (e.g., Virtuoso PL Procedures).

One key consideration is whether the conversion is conducted internally or externally. External methods include custom scripts using Python and RDBLib package ([Method II]) and RDF Generators (e.g., csv2rdf4lod). Other methods are generally internal (or somewhat Virtuoso-Specific).

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 operate independently:

[Method I] (Internal)

It is somewhat W3C & Virtuoso-specific, utilizing R2RML and accessible for those less proficient in Python. As we delve deeper, this method could prove more efficient for automatic ontology generation. By the end of Oct, 2024, we further experimented with importing multiple reconciled CSV files with foreign key references into Virtuoso, enabling an "assembly" into a relational database. This could significantly enhance RDF conversion efficiency (See <https://github.com/DDMAL/linkedmusic-datalake/issues/214>).

[Method II] (External)

It requires a certain proficiency of python and offers a degree of universality and flexibility. The mapping procedure remains somewhat manual (e.g., see MusicBrainz/mapping.json). This method is still favourable regardless of any schema indications from a relational database. However, most reconciled CSV files are extracted as a part of a relational database with a schema. So we may rely on this method less as we gain mastery of [Method I], particularly when schema information is available.

[Method III]

This method is especially suitable for incorporating data in an immediate but fragmented manner. However, it requires customization of entity URIs using SPARQL.

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

The choice of RDFLib + Python + Conductor can be partly explained by our need for flexibility and specific project requirements. The sentence below can partially explain it.

6.2 Ongoing or Future Work

Initially, our research avoided reliance on ontology. Moving forward, we may not be confined to the thinking, and we recognize the in-erasable value of ontology, particularly for (1) NLQ2SPARQL with ChatGPT and (2) knowledge reasoning.

6.2.1 Virtuoso Conductor + SPARQL

A shared challenge with Methods 6.2.1 and 6.2.2 is that RDF graphs cannot always be exported normally. For now, we have used flexible (or stopgap) measures, though simpler methods may exist.

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

Linked Data View definition

Column Selection

Table Name	Action
CSV2RDF_Sample.DBA.sampleCantusChant_csv	Edit Primary Key Selection

Back Next Prepare to Execute Generate

click on Edit to see:

Choose Mapping Options for "CSV2RDF_Sample.DBA.sampleCantusChant_csv"

Base Ontology none ▾

Base Class auto ▾

Column Name *	Property	Binding/MIME Type
<input type="checkbox"/> ID	auto ▾	literal ▾
<input type="checkbox"/> Chant_ID	auto ▾	literal ▾
<input type="checkbox"/> incipit	auto ▾	literal ▾
<input type="checkbox"/> genre	auto ▾	literal ▾
<input type="checkbox"/> src_link	auto ▾	literal ▾

*To designate one column for subject label, check the box.

Cancel Save

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

6.2.2 ISQL of Terminal + R2RML

It's also worth mentioning, placing multiple CSV files in an RDB allows for a single R2RML script to customize properties across these files. We've tested this with foreign key-related CSV files [See the sample data and the corresponding R2RML scripts on Issue 214 (<https://github.com/DDMAL/linkedmusic-datalake/issues/214>), where 2 points are noteworthy:]

(1) The primary key field does not require a complete URI for each entry, as customization can be handled in R2RML code:

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

(2) Foreign key references can be mapped, 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 `rdb2rdf_dav.vad` package, a new R2RML tab appears under Linked Data, which warrants further investigation.

6.3 General Conclusion

For determining the best method for CSV2RDF, currently [Method I] and [Method II] may end up in a tie or complement each other. However,

CSV2RDF can be viewed as partial (preliminary) task of RDB2RDF; in practice, we are actually dealing with 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 multiple CSV files within which there exist foreign key references, it essentially mirrors uploading a reconciled RDB to Virtuoso.

Furthermore, while using [Method II] in the past, we realized referencing schemas could be advantageous. Hence, it's reasonable and beneficial to also transfer any RDB entirely to Virtuoso before reconciliation. Additionally, during RDB2RDF on Virtuoso, an ontology can be generated from the corresponding schema, aiding ChatGPT's NLQ2SPARQL capabilities.

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.