Aligner une base de données avec une ontologie avec Protégé-Ontop

Auteur(s) : Florian Hivert, Olivier Marlet,
Institution émettrice : Consortium MASA
contact mail du projet :olivier.marlet@univ-tours.fr

I. Introduction

Protégé est un logiciel open-source gratuit utilisé pour éditer et construire des ontologies. L'ajout du plugin Ontop étend les fonctionnalités de Protégé avec un outil d'appariement (*mapping*) entre une base de données SQL et une ontologie (dans notre cas, le CIDOC CRM) afin de porter les données sur le Web sémantique sous forme de triplets RDF. Ontop est une extension open-source développée par l'Université de Bozen-Bolzano (Italie).

Ontop est l'outil privilégié pour générer des triplets RDF depuis les données d'une base de données SQL. Au préalable de l'utilisation d'Ontop, il est préférable de connaître l'ontologie servant de référence, d'avoir un modèle de données facilitant la construction des appariements et une bonne connaissance de la syntaxe des requêtes. Comme tout procédé d'appariement avec des ontologies, un nettoyage des données et/ou un enrichissement des données est souvent nécessaire en amont et ne pourra qu'améliorer la qualité du jeu de données et son exploitation sur le Web sémantique.

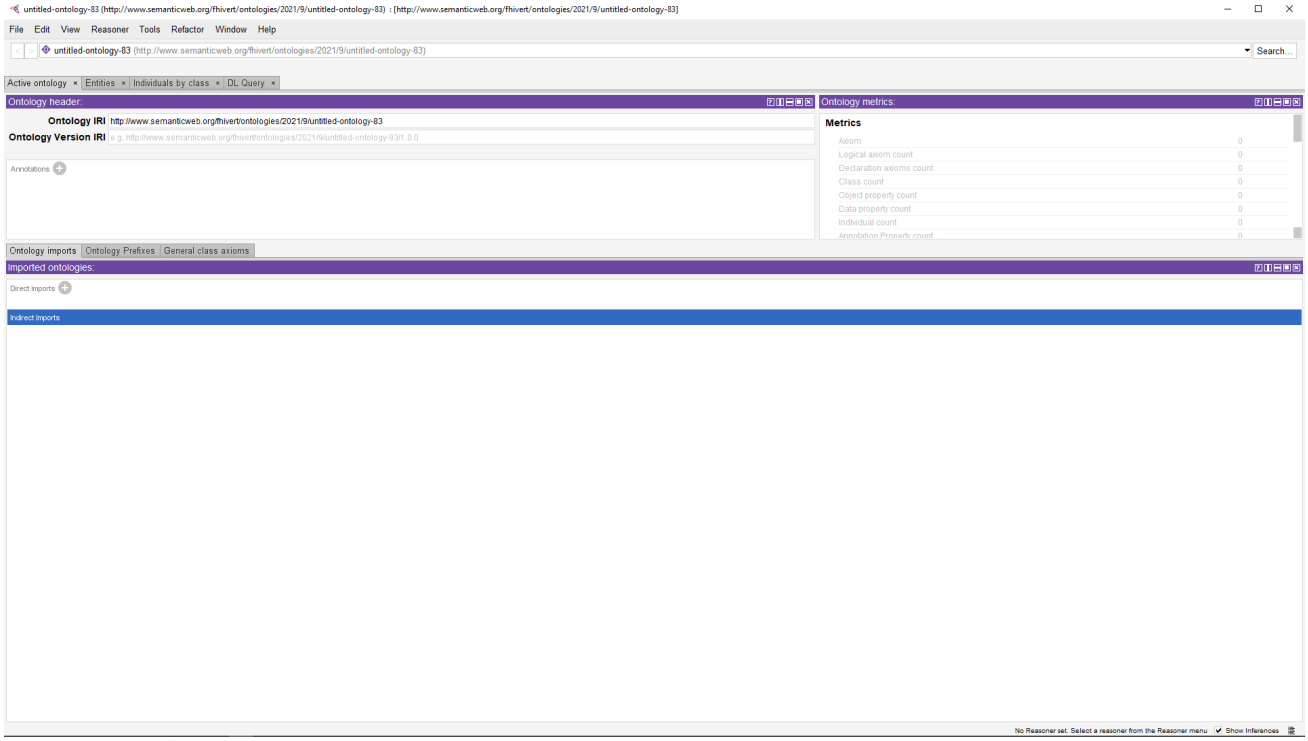
II. Installation et configuration de Protégé-Ontop

1. Installation de Protégé

Tout d'abord, télécharger et installer Protégé depuis ce site :

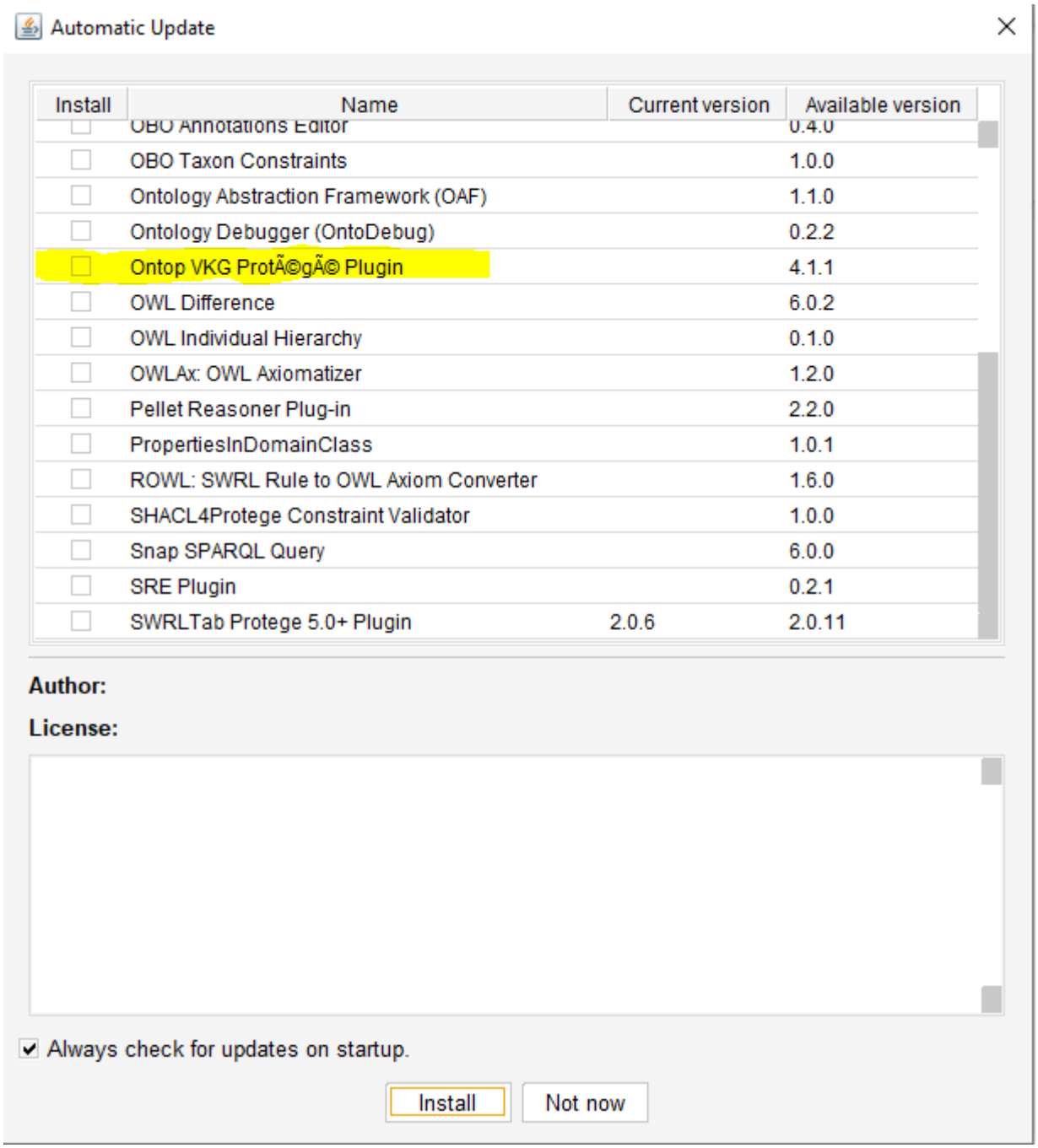
https://protege.stanford.edu/products.php#desktop-protege

Une fois le fichier téléchargé en local, aucune installation logicielle n'est nécessaire. Pour lancer Protégé, en fonction de l'OS, cliquer sur le fichier .exe pour Windows ou ouvrir un terminal en lignes de commandes et lancer la commande ./run.bat dans le dossier.



2. Activation de Ontop

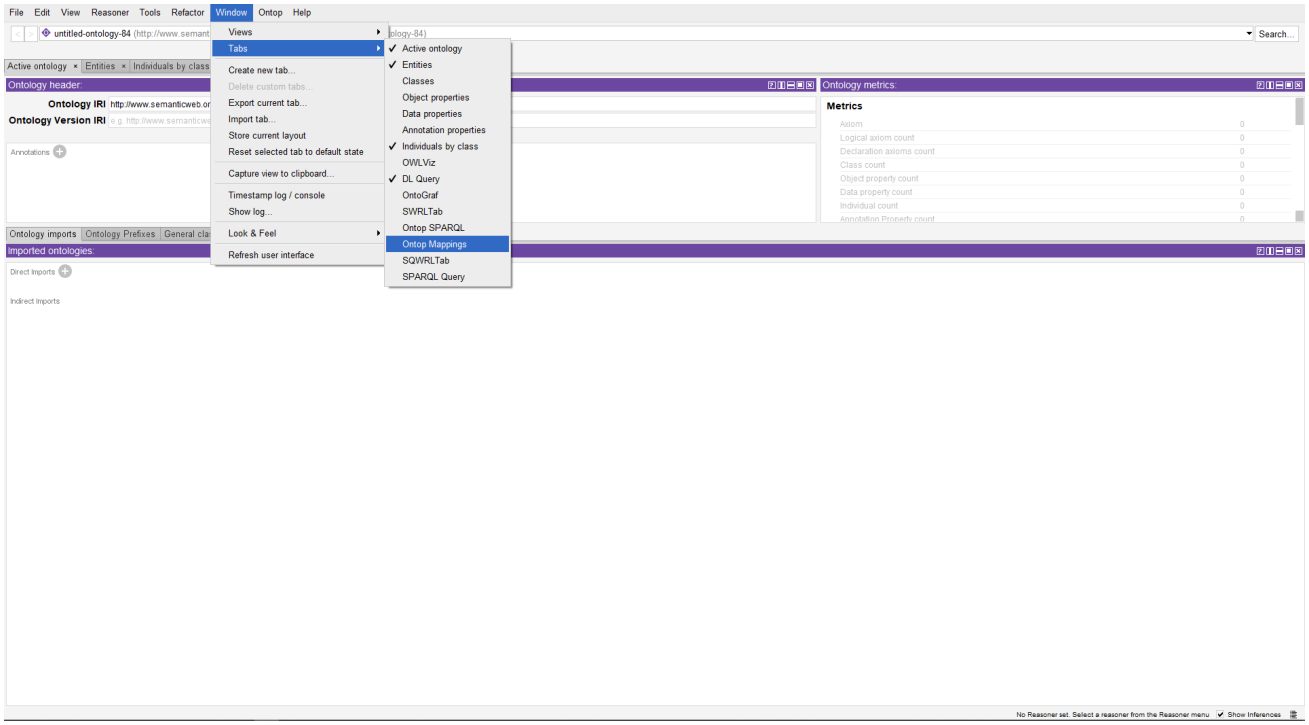
Une fois le logiciel lancé, il faut installer le plugin Ontop : « File -> Check for Plugins... ».



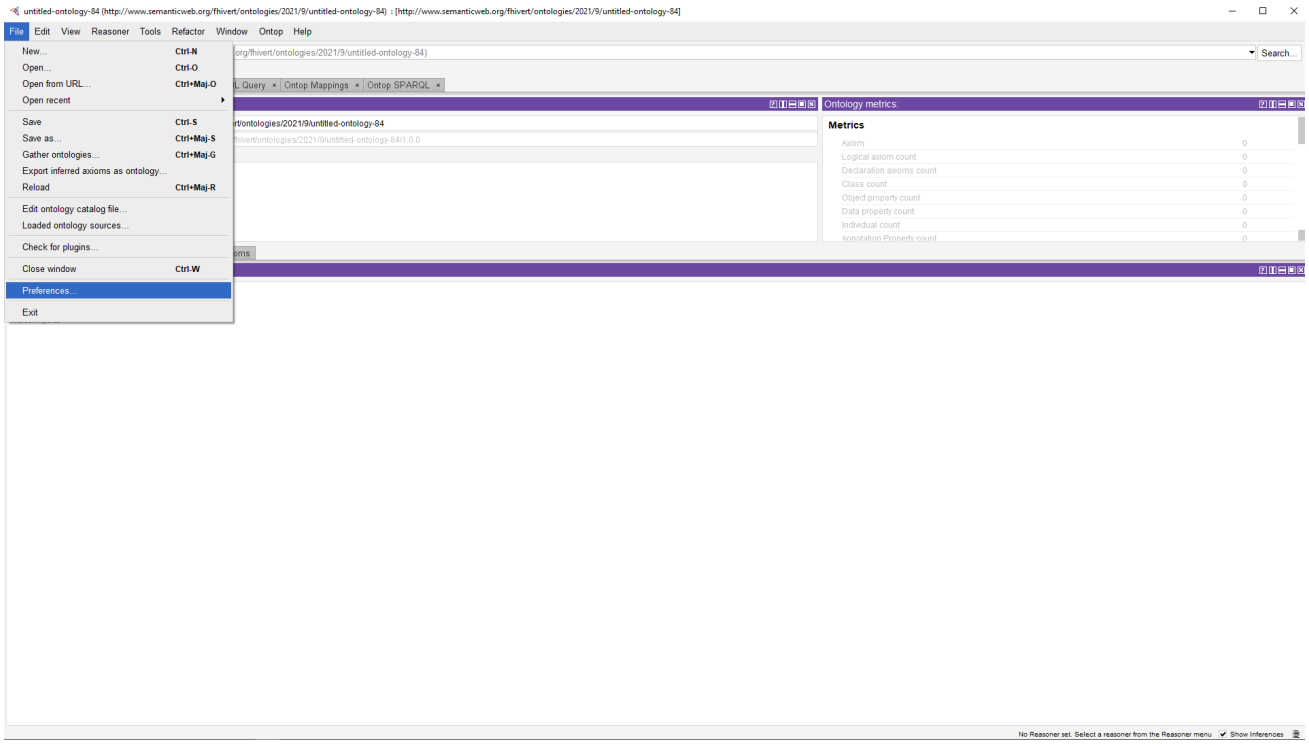
Chercher dans la liste et sélectionner le plugin « Ontop VKG Protégé Plugin » puis cliquer sur Install.

Attendre l'apparition, d'un pop-up, puis redémarrer Protégé pour finaliser l'installation du plugin Ontop.

À savoir, il existe également sur le site web d'Ontop une version Bundle Protégé-Ontop qui permet d'installer Protégé avec l'extension Ontop déjà chargée (https://github.com/ontop/ontop/releases).



À la réouverture de Protégé, aller dans Window -> Tabs et cocher les fenêtres Ontop Mappings et Ontop SPARQL pour avoir accès aux fonctionnalités de Ontop dans Protégé.



3. Chargement du driver

Ensuite pour permettre à Protégé d'avoir accès à la base de données, il faut charger le driver Java qui permettra de communiquer avec la base de données.

Télécharger les drivers Java pour se connecter à une base SQL (en fonction du SGBD-R – systèmes de gestion de bases de données relationnelles) :

Driver pour MySQL : https://mvnrepository.com/artifact/mysql/mysql-connector-java

Driver pour PostGreSQL : https://jdbc.postgresql.org/download.html

Driver pour MariaDB : https://downloads.mariadb.org/connector-java/

Driver pour SQLite : https://bitbucket.org/xerial/sqlite-jdbc/downloads/

Depuis la fenêtre « File ->Preferences », sélectionner l'onglet « JDBC Drivers ».

Cliquer sur le bouton « Add » en bas à gauche pour charger les drivers nécessaires.

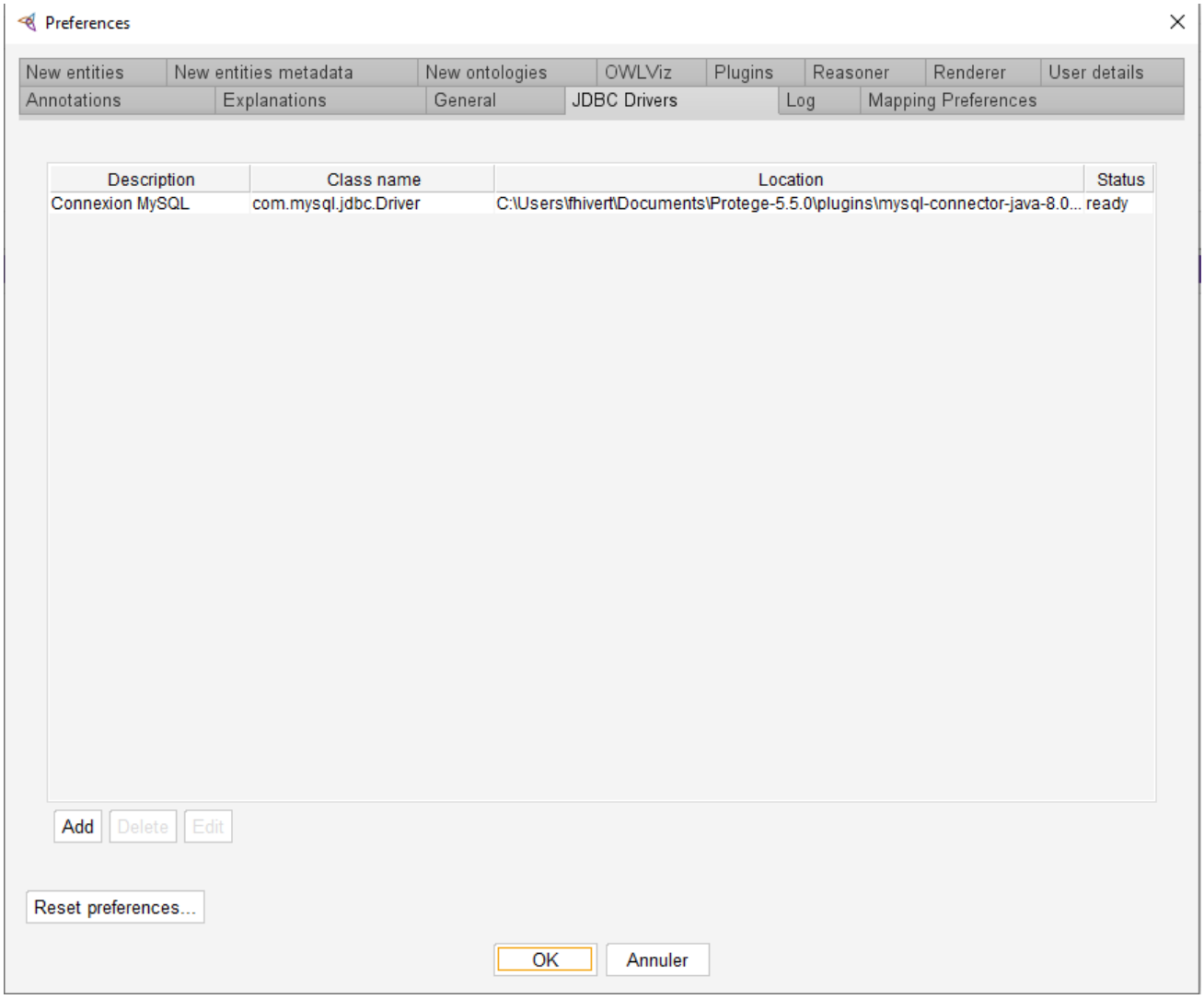
Remplir comme suit :

Description : Driver mySQL vx.x (nom libre pour identifier le driver)

Class Name : com.mysql.jdbc.Driver (pour mySQL) ou org.mariadb.jdbc.Driver (pour MariaDB) ou org.postgresql.Driver (pour postGreSQL) ou org.sqlite.JDBC (pour SQLite)

Driver: utiliser le bouton "Browse" pour aller chercher le fichier du driver (.jar) en local

Et valider avec « Ok », puis s'assurer que le statut du driver est bien « ready ».



Une fois le(s) driver(s) nécessaire(s) ajouté(s) dans l'onglet « JDBC Drivers », Protégé-Ontop est désormais configuré pour se connecter à la base de données et donner l'accès aux outils d'appariement.

Selon les versions de Protégé, il peut arriver que l'application « bugue » et qu'il faille la relance. Il n'y a pas de système de sauvegarde automatique, il donc essentiel de procéder à des sauvegardes très régulière du projet pour s'assurer de ne rien perdre.

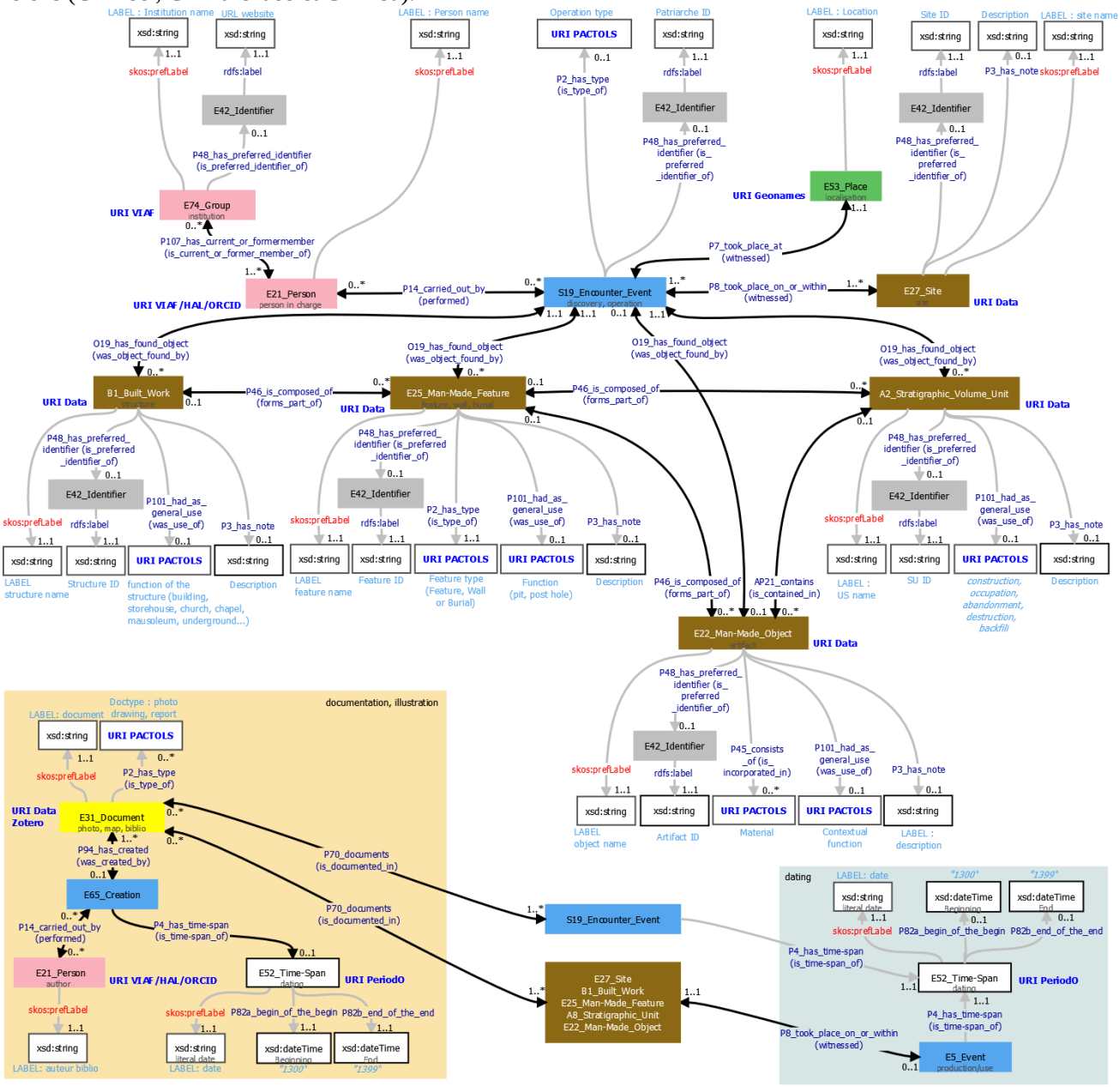
III. Gestion des ontologies avec Protégé

1. Charger une ontologie

Le paramétrage de Protégé étant terminé, l'étape suivante est de charger les ontologies nécessaires à l'appariement.

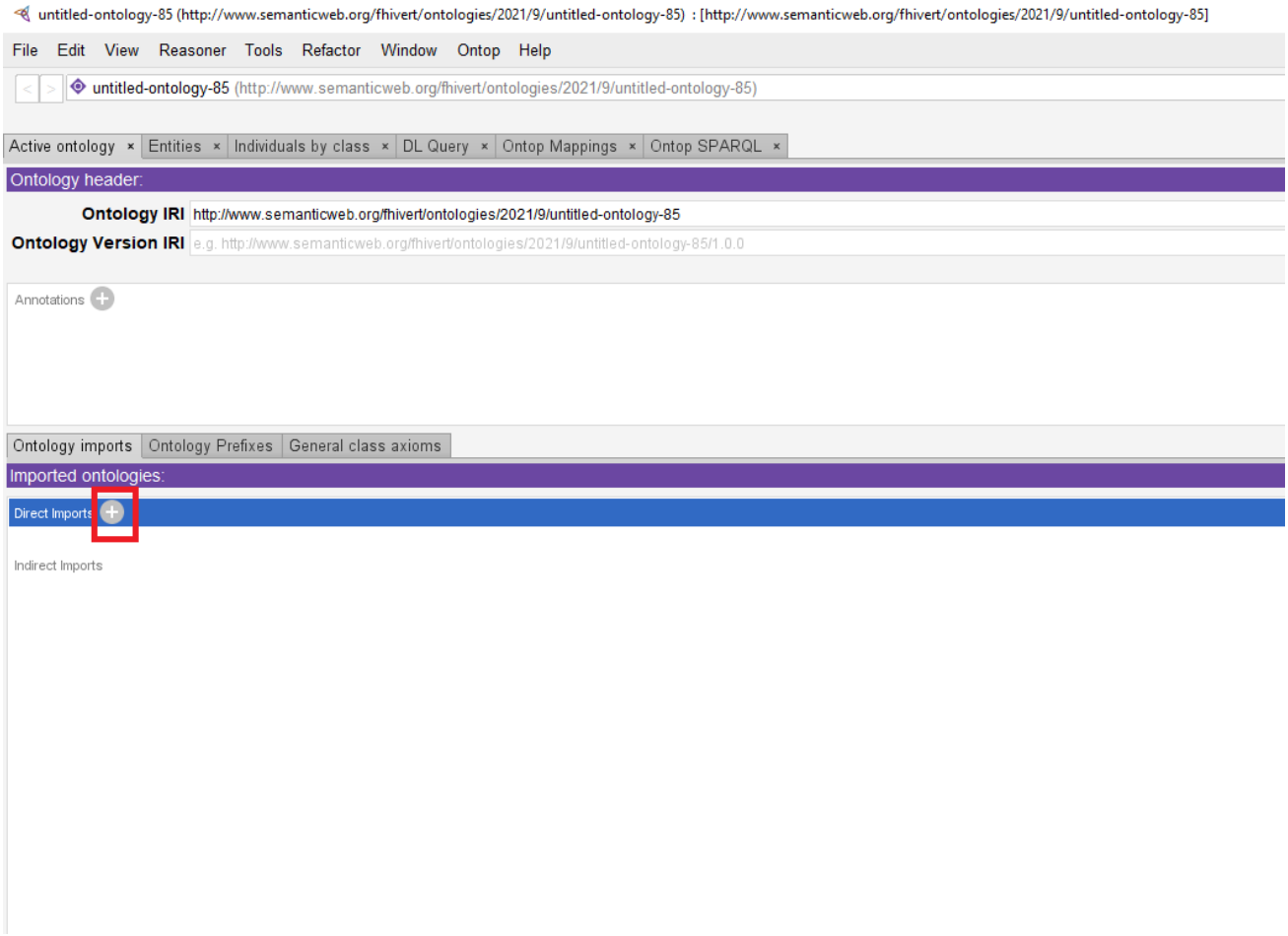
Il faut tout d'abord choisir les ontologies avec lesquelles apparier le jeu de données.

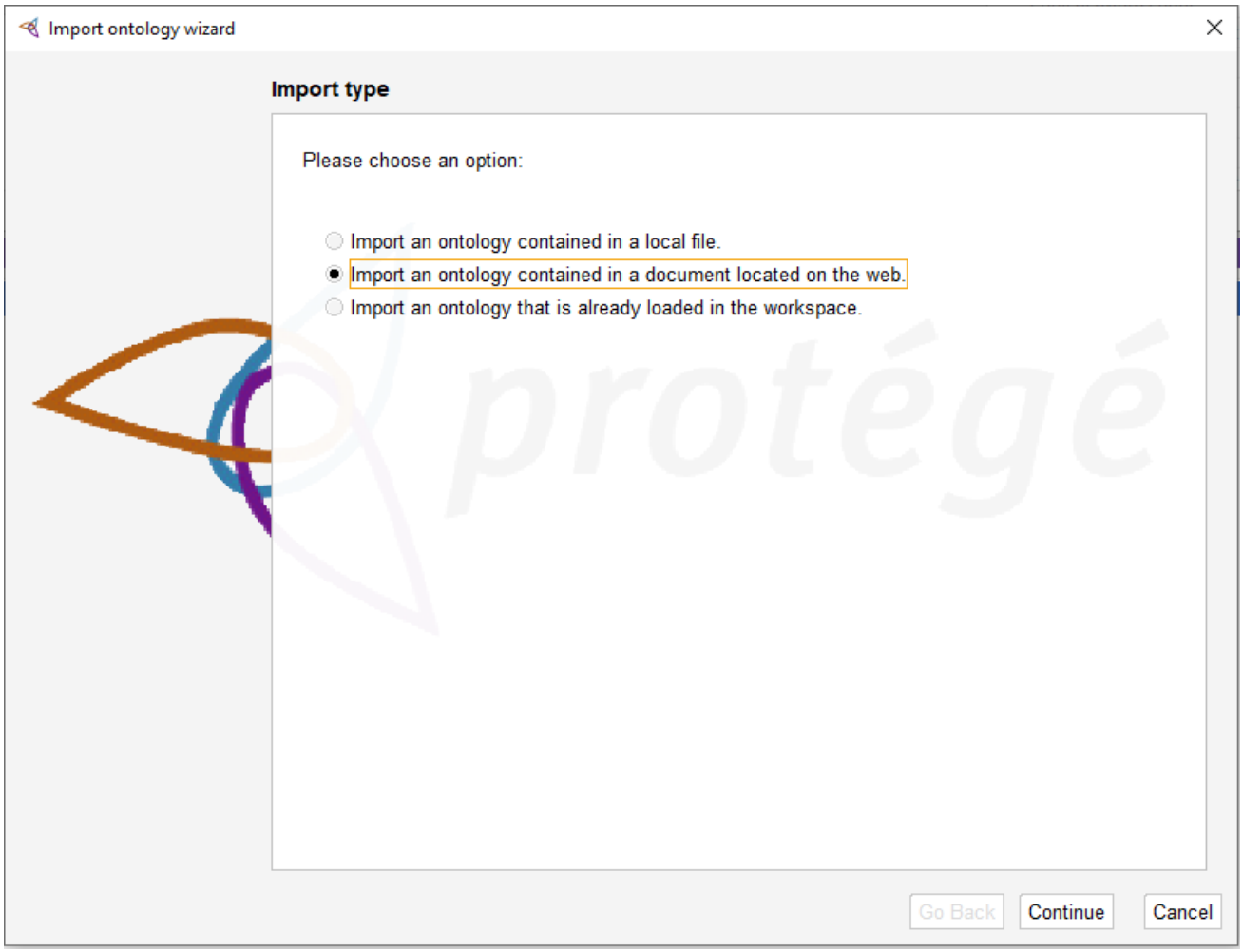
Nous allons ici prendre l'exemple de données à convertir en triplets RDF pour la plateforme OpenArchaeo, outil développé par le consortium MASA afin de porter sur le Web sémantique des graphes de données archéologiques. Pour OpenArchaeo, un modèle de données a été mis en place à partir de l'ontologie du CIDOC-CRM et quelques une de ces extensions (CRMsci, CRMarchaeo et CRMba).



Modèle de données pour OpenArchaeo - Olivier Marlet

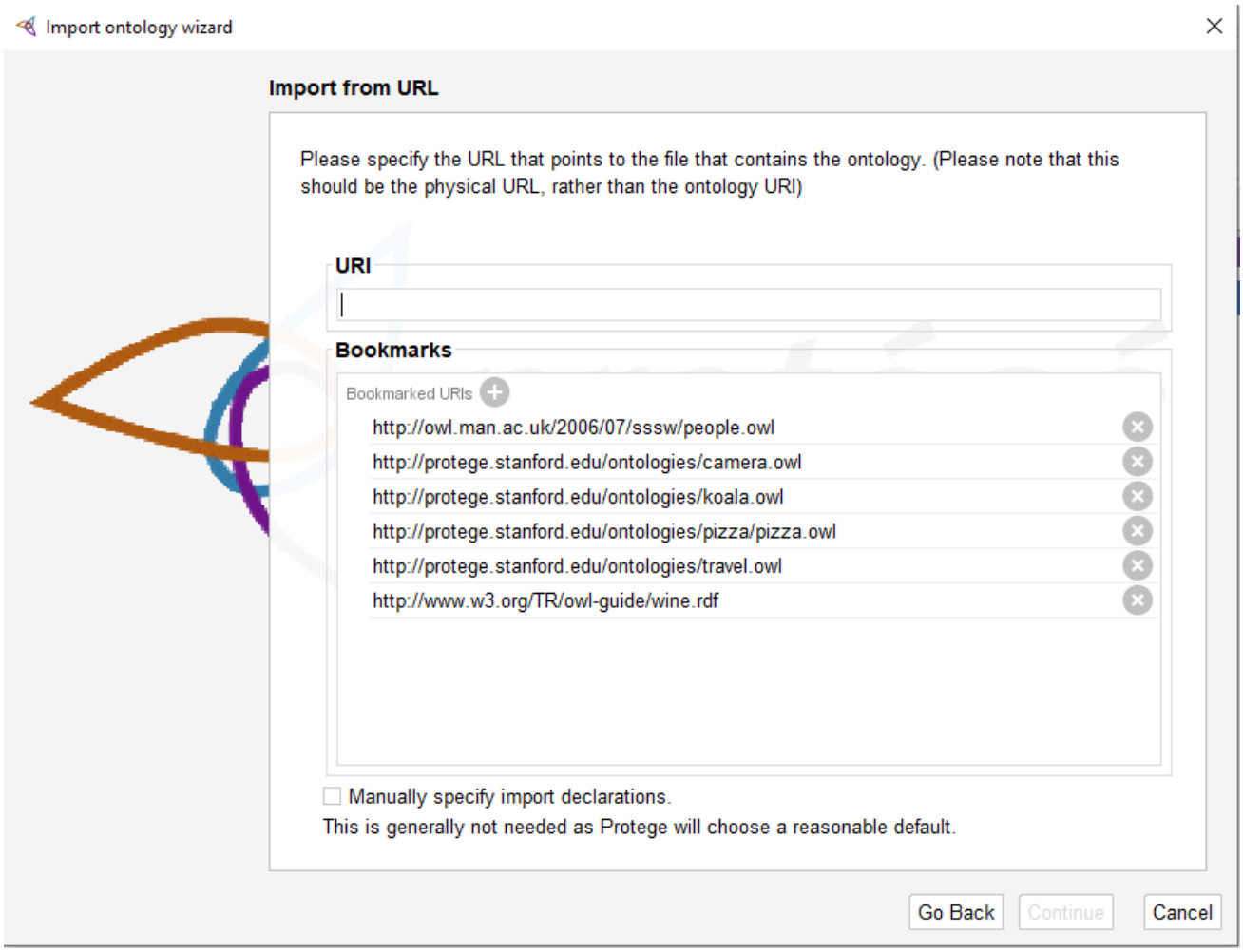
Pour ajouter une ontologie à Protégé, sélectionner l'onglet « Active ontology », puis dans la partie basse de l'écran, l'onglet « Ontology import ». Cliquer sur le bouton « + » à côté de « Direct Imports ».


untitled-ontology-85 (http://www.semanticweb.org/fhivert/ontologies/2021/9/untitled-ontology-85) : [http://www.semanticweb.org/fhivert/ontologies/2021/9/untitled-ontology-85]
File Edit View Reasoner Tools Refactor Window Ontop Help
untitled-ontology-85 (http://www.semanticweb.org/fhivert/ontologies/2021/9/untitled-ontology-85)
Active ontology
Entities
Individuals by class
DL Query
Ontop Mappings
Ontop SPARQL
Ontology header:
Ontology IRI
http://www.semanticweb.org/fhivert/ontologies/2021/9/untitled-ontology-85
Ontology Version IRI
e.g. http://www.semanticweb.org/fhivert/ontologies/2021/9/untitled-ontology-85/1.0.0
Annotations
Ontology imports
Ontology Prefixes
General class axioms
Imported ontologies:
Direct Imports
Indirect Imports



Plusieurs options existent pour ajouter une ontologie dans Protégé :

- « *Import an ontology contained in a local file* » : cette option permet d'importer des ontologies contenues dans un fichier téléchargeable, généralement au format .owl ou .rdfs. Il suffit de montrer le chemin du fichier en local pour importer l'ontologie.
- « *Import an ontology contained in a document located on the web* » : importe une ontologie depuis son URL en ligne.
- Charger une ontologie déjà associée à l'espace de travail.



Par exemple, l'ontologie CIDOC-CRM est importée en utilisant l'URL : http://www.cidoc-crm.org/cidoc-crm/

En cliquant sur « Continue » puis « Finish », l'ontologie sera téléchargée et ajoutée à Protégé.

Certaines ontologies ne sont pas disponibles en ligne via une URL. Il faut alors télécharger les fichiers et les ajouter dans Protégé à l'aide de la première option vue ci-dessus. C'est le cas des extensions du CIDOC-CRM :

- CRMarchaeo : http://www.cidoc-crm.org/crmarchaeo/sites/default/files/CRMarchaeo_v1.4.1.rdfs
- CRMsci : http://www.cidoc-crm.org/crmsci/sites/default/files/CRMsci_v1.2.6.rdfs
- CRMba : (pas encore disponible en ligne [décembre 2021])

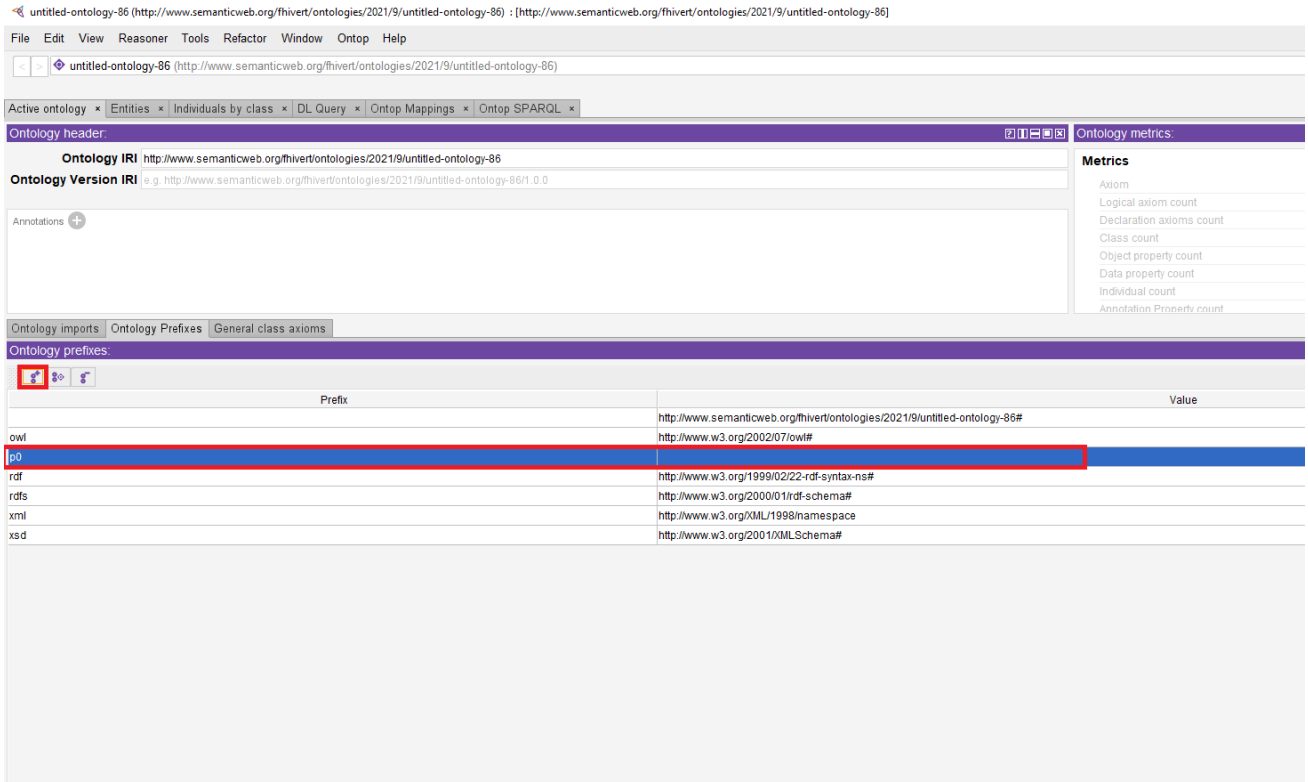
On peut ajouter à ces ontologies, SKOS, SKOS-XL, Dublin Core et Dublin Core Terms :

- SKOS : http://www.w3.org/2004/02/skos/core#
- SKOS-XL : http://www.w3.org/2008/05/skos-xl#
- Dublin Core : https://www.dublincore.org/specifications/dublin-core/dcmi-terms/dublin_core_elements.rdf
- Dublin Core Terms : https://www.dublincore.org/specifications/dublin-core/dcmi-terms/dublin_core_terms.rdf

2. Préfixes des ontologies

Une fois les ontologies téléchargées, il faut configurer les préfixes. Un préfixe est un raccourci qui renvoie à l'URL associée à l'ontologie (le *namespace*). Ainsi lors de l'appariement, c'est ce préfixe qui sera utilisé pour appeler les

classes et les propriétés de l'ontologie plutôt que d'utiliser le *namespace* parfois très long. Par exemple, le préfixe « crm: » fait directement référence au *namespace* « http://www.cidoc-crm.org/cidoc-crm/ » et permettra d'écrire « crm:E22_Man-made_Object » au lieu de « http://www.cidoc-crm.org/cidoc-crm/:E22_Man-made_Object ».



À partir de l'onglet « *Ontology Prefixes* », cliquer sur le premier bouton (« *Add prefix mapping* ») comme indiqué ci-dessus. Ce bouton ajoute le préfixe « p0 » et une *Value* qui vont pouvoir être modifiés.

Le préfixe est obligatoire pour chacune des ontologies chargées :

Prefix	Value
crm	http://www.cidoc-crm.org/cidoc-crm/
crminf	http://www.ics.forth.gr/isl/CRMinf/
crmarchaeo	http://www.ics.forth.gr/isl/CRMarchaeo/
crmba	http://www.ics.forth.gr/isl/CRMba/
skos	http://www.w3.org/2004/02/skos/core#
skosxl	http://www.w3.org/2008/05/skos-xl#
dc	http://purl.org/dc/elements/1.1/
dcterms	http://purl.org/dc/terms/

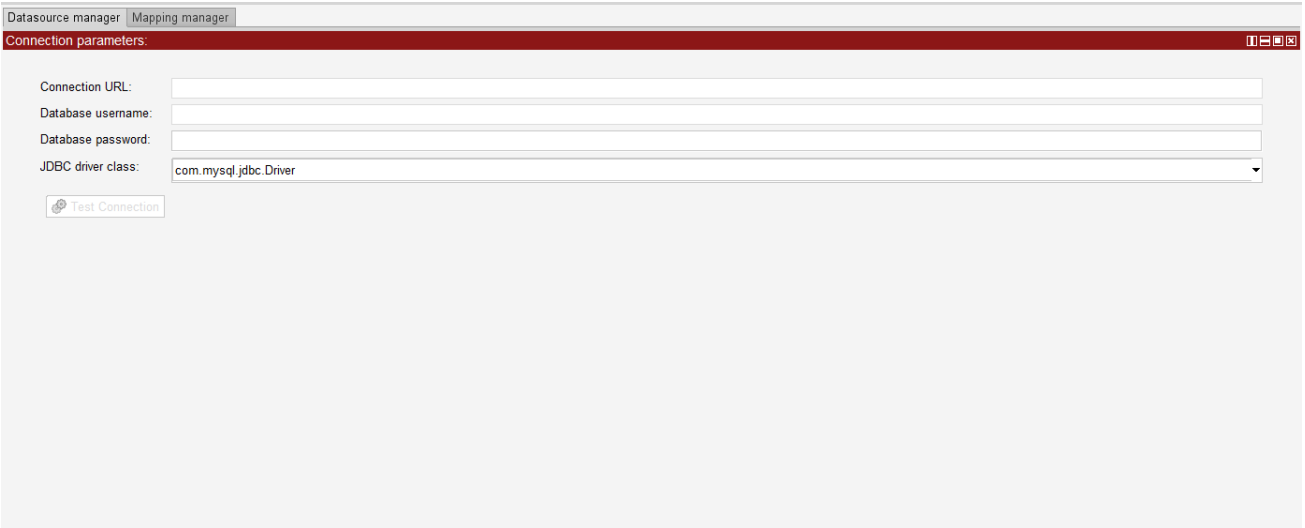
Le même procédé est utilisé pour les données à apparier en utilisant l'adresse du site web où les données sont accessibles en ligne. Par exemple, pour ArSol, base de données du Laboratoire Archéologique et Territoires de Tours accessible sur le site http://arsol.univ-tours.fr/, le préfixe pourra être « arsol » et renverra à l'adresse à laquelle les données sont accessibles http://arsol.univ-tours.fr/4DACTION/WFICHEWEB/.

Avec un préfixe arsol: qui reprend la valeur http://arsol.univ-tours.fr/4DACTION/WFICHEWEB, on peut écrire *arsol:{idobjet}*pour créer une URI pour l'objet en question dans notre mapping (voir partie 5) sans avoir à taper l'URL complète.

IV. Connexion à une base de données

1. Paramétrage de la connexion

Si le driver Java pour la base de données à traiter est correctement installé, il est possible maintenant de se connecter à la base de données, qu'elle soit sur MySQL, MariaDB ou PostGreSQL. Cette étape permet la récupération des données depuis la base de données à l'aide de requêtes SQL SELECT pour intégrer les occurrences de la base de données dans la construction des triplets RDF.



Depuis l'onglet en haut « *Ontop Mappings* » dans Protégé, le sous-onglet « *Datasource manager* » demande de remplir 4 champs pour mettre en place les paramètres de connexion à la base de données :

Connection URL : (selon le driver chargé, voir Partie 1)

- jdbc:mysql://localhost/ma_base (mySQL)
- jdbc:mariadb:// localhost/ma_base (MariaDBL)
- jdbc://localhost/ma_base (avec PostGreSQL)
- jdbc:sqlite:/url_locale/ma_base.db (SQLite)
- Database Username : root
- Database Password :
- Driver Class : (le nom de la classe indiqué dans les paramètres du driver, voir Partie 1)

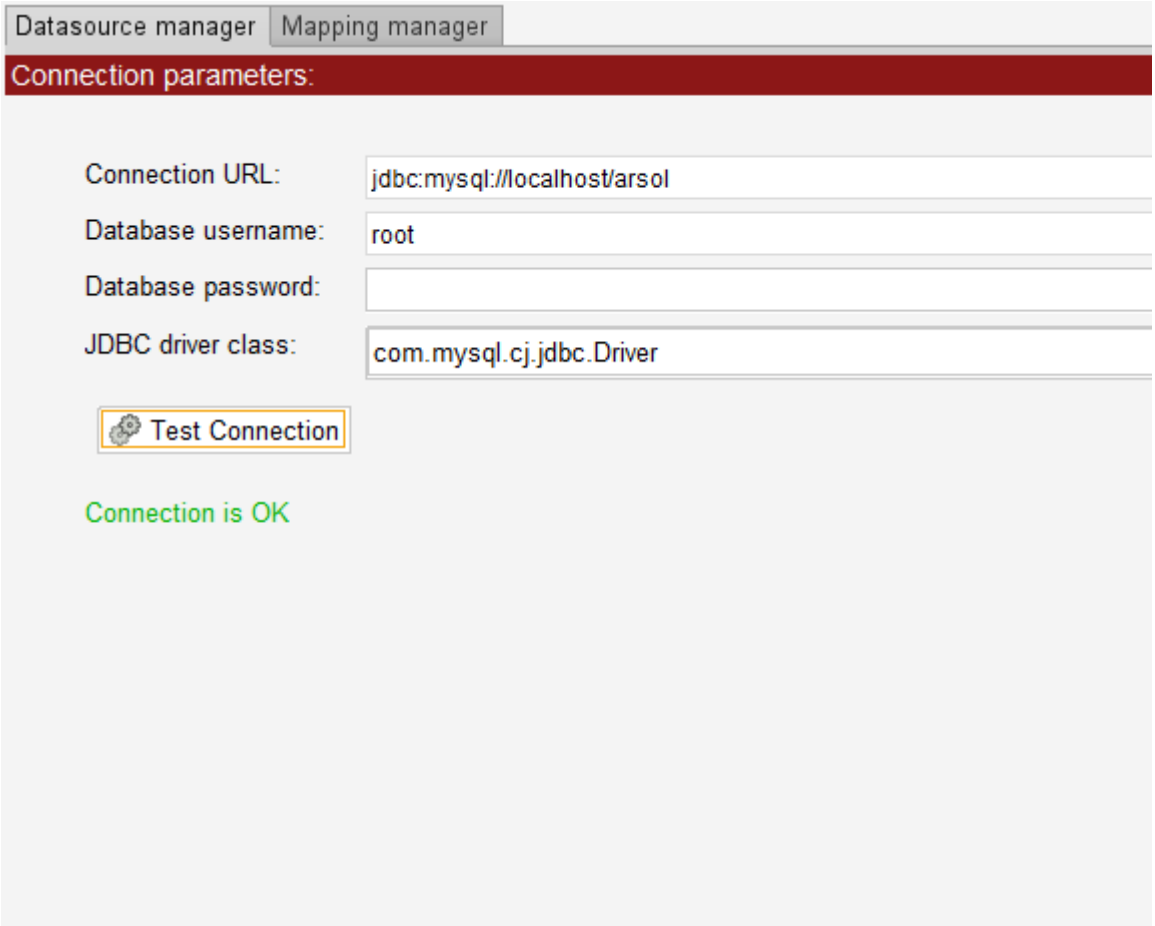
Puis cliquer sur "Test Connection" pour s'assurer que la connexion fonctionne bien.

Des problème d'encodages des caractères spéciaux peuvent survenir lors de la récupération des données selon l'encodage de la BDD. Il peut alors être nécessaire de passer des paramètres dans l'URL de connexion. En voici un exemple :

```
jdbc:mysql://localhost/arsol?sessionVariables=sql_mode='ANSI'&useCursorFetch=
```

2. Exemple de connexion à une base mySQL

L'exemple ci-dessous présente la connexion à une base MySQL mais les principes restent les mêmes pour une base MariaDB ou PostGreSQL.



Dans le cadre d'une conversion de base vers des triplets RDF, il est possible de travailler sur une copie locale de la base (exemple ci-dessus) plutôt que sur la base en production (en ligne). Pour une base en ligne, il suffira de remplacer « //localhost/arsol » dans « Connection URL » avec l'adresse où la base de données se trouve (et pas le site Web qui publie cette base).

```
jdbc:mysql://localhost/arsol
```

La partie en rouge indique le driver sélectionné (ci-dessus mysql) et la partie en bleue donne l'adresse de la base de données (ici la base ArSol hébergée sur localhost).

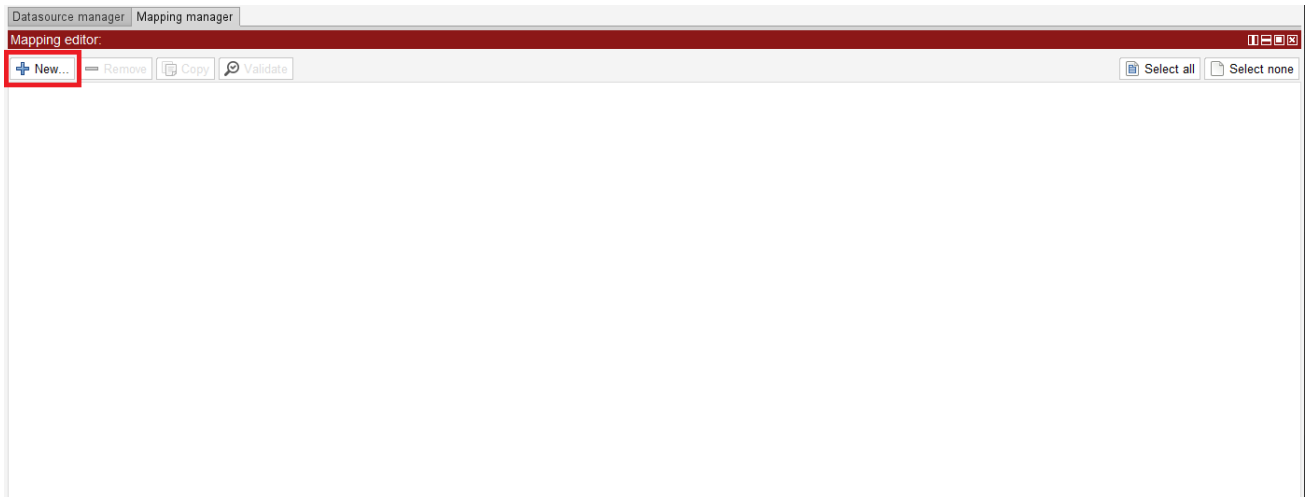
Le « *database username* » et le « *database password* » doivent seulement contenir les identifiants de connexion au SGBD-R, que cela soit MySQL, MariaDB ou PostGreSQL.

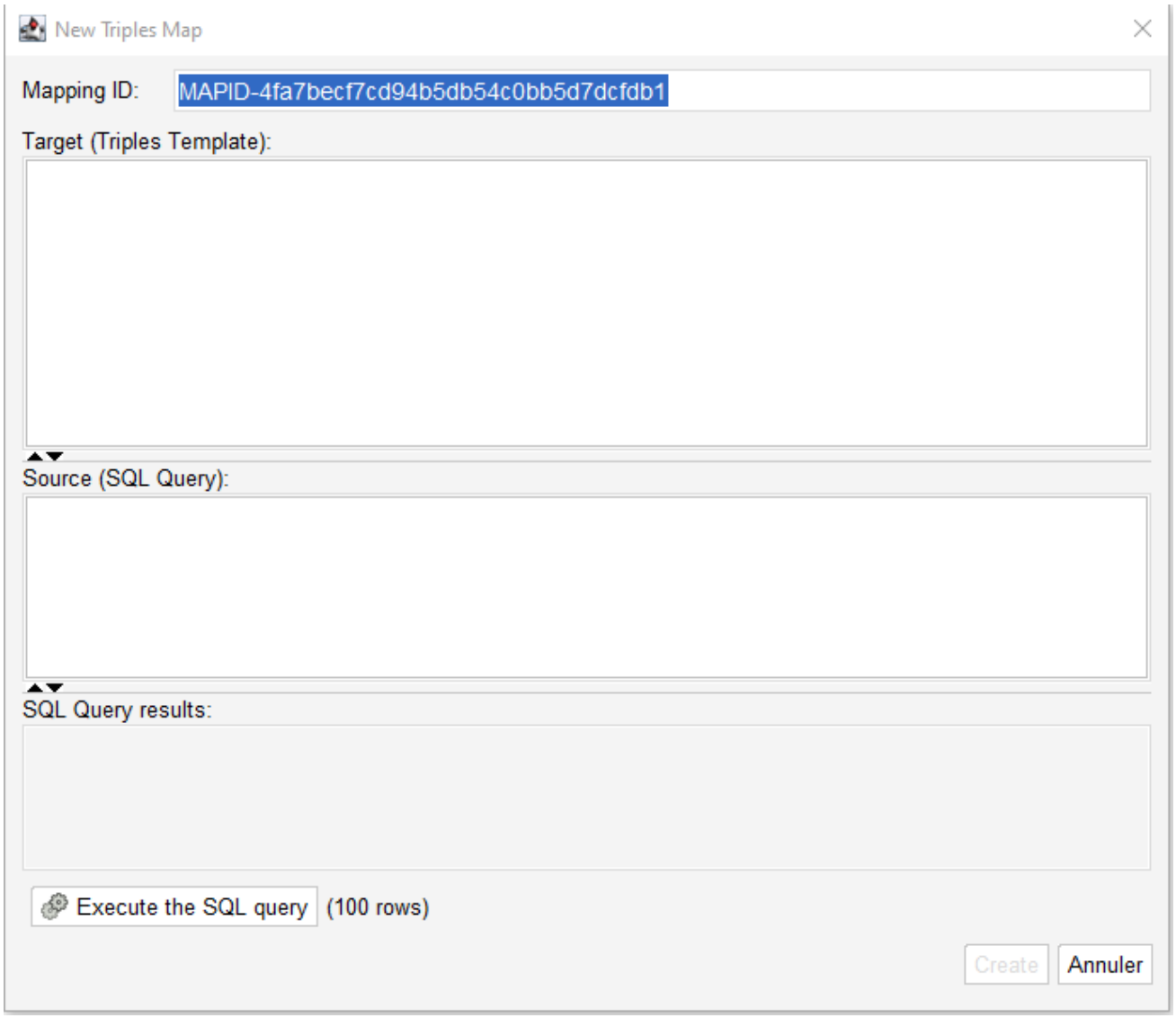
Dans « *JDBC driver class* », il suffit de choisir le driver installé précédemment et qui correspond au SGBD-R utilisé.

Une fois tous les champs remplis, cliquer sur le bouton « *Test Connection* » pour vérifier que Ontop a bien accès à la base de données. Si le message « Connection is OK » s'affiche, alors l'accès à la base de données est fonctionnel.

V. Structuration d'un mapping

On accède à l'écriture des *mappings* en allant dans l'onglet « *Ontop Mappings* » et en ouvrant la sous-fenêtre « *Mapping Manager* ». En cliquant sur le bouton « + New », une fenêtre s'ouvre avec deux champs vides, un champ « *Target* » qui va recevoir les mappings et un champ « *Source* » pour écrire une requête SQL destinée à récupérer les données dans le SGBD-R.





Le champ « Mapping ID » permet de donner un nom au mapping afin de s'organiser et de s'y retrouver dans notre processus de mapping des données. Il faut organiser les mappings un minimum en choisissant des noms pertinents.

Source (triples template) : cet espace contient le modèle de triplets à générer selon une syntaxe très précise (voir ci-dessous).

Target (SQL Query) : il s'agit de la requête SQL à rédiger pour accéder aux données. Toute donnée mobilisée dans le modèle de triplets doit être récupérée dans le SELECT de la requête SQL.

Le bouton « Execute the SQL query » permet de vérifier les données récupérées par la requête SQL. Une erreur surviendra en cas d'erreur de syntaxe dans la requête.

VI. Syntaxe des triplets

1. Principes généraux pour écrire des triplets

L'écriture de *mapping* dans Protégé-Ontop s'appuie sur le principe de base du web sémantique et le fonctionnement d'un triplet RDF. Un triplet RDF est construit à l'aide de trois éléments :

Sujet - Prédicat – Objet

- Le sujet est toujours une instance définie par une entité ou une classe ;
- Le prédicat ou propriété est le verbe du triplet, il exprime l'association entre le sujet et l'objet, cette association est défini par rapport à la classe du sujet (domaine) et de l'objet (co-domaine) ;
- L'objet est quant à lui, une instance similaire au sujet OU une donnée littérale typée.

```
<URI Sujet (Domaine)> <URI Prédicat (Propriété)> <URI Objet (Co-Domaine)> ou "valeur"
```

La définition des classes se fait au début de l'écriture du triplet afin de respecter les règles de domaine et de co-domaine des ontologies, et d'attribuer une classe (entité de l'ontologie) à une instance (une donnée).

Dans Protégé-Ontop, le format de rédaction des triplets demandé est le format Turtle. Par exemple, si on veut typer le mobilier décrit dans ArSol, selon le modèle de OpenArchaeo (voir ci-dessus), le *mapping* serait :

```
arsol:imobi{mobilier_x}    a    crm:E22_Man-Made_Object    ;    crm:P2_has_type <{lien_pactols_pioche}> .
```

Ontop permet dans un même mapping de définir plusieurs triplets et objets à la suite :

```
arsol:imobi{mobilier_x}    a    crm:E22_Man-Made_Object    ;    crm:P2_has_type <{lien_pactols_pioche}>       ;       skos:prefLabel       {label}^^xsd:string .<{lien_pactols_pioche}> a crm:E55_Type ; skos:prefLabel “pioche”^^xsd:string .
```

- On termine un triplet avec un « . ».
- On sépare deux propriétés pour un même sujet avec un « ; ».
- On sépare deux objets pour un même sujet et une même propriété avec une « , ».
- Les éléments entre accolades sont les variables récupérées dans les requêtes SQL (l'intitulé dans le SELECT de la requête SQL doit être rigoureusement identique au label indiqué entre accolades).
- Les “< >” indiquent que la chaîne entre chevron est une URI. Le résultat doit permettre d'obtenir un lien. Ici, {lien_pactols_pioche} récupère un lien ark dans les données, lien qui est une URI. Les “< >” ne doivent pas être utilisés lorsque l'on utilise un préfixe comme pour arsol:imobi{mobilier_x}.

La coloration syntaxique de Ontop aide à vérifier la nature de l'Objet attendu :

- Si la propriété est turquoise/vert, l'Objet sera une chaîne () ou une date ().
- Si la propriété est bleue, l'Objet sera nécessairement un URI vers un autre concept qui deviendra éventuellement « Sujet » d'un autre mapping.

Quand la chaîne stockée en BDD est un URI, il faut l'appeler avec des chevrons dans le mapping : `<{link}>`

Si cette syntaxe n'est pas respectée, la fenêtre de rédaction du mapping sera encadrée de rouge et le bouton « Create » restera grisé. Il faut également être vigilant aux préfixes et aux noms des classes et des propriétés : une mauvaise

orthographe empêchera Ontop d'identifier la classe ou la propriété mobilisée. Au fur et à mesure de la rédaction, Ontop explore l'ontologie pour s'assurer que les classes et propriétés existent bien et les met en couleur si c'est le cas. L'absence de mise en couleur peut être le signe d'une mauvaise graphie pour l'élément.

2. Exemple commenté de modèles de triplets

```
iceramm:bdresultats.php?site={idsite} a crm:E27_Site ;
```

uri d'une instance définie comme étant un site (tel que dans le CIDOC CRM)

```
skos:prefLabel {nom}^^xsd:string ;
```

(on reste sur le même sujet) dont le label préféré est une chaîne de caractère (= champ nom de la BDD)

```
crm:P48_has_preferred_identifier iceramm:bdresultats.php?site={idsite}/num ;
```

dont l'identifiant préféré est un numéro de site (= champ idsite de la BDD), on introduit en objet un nouveau concept, l'identifiant.

```
crm:P53_has_former_or_current_location geonames:{geoname_id} ;
```

dont la localisation est un URI GeoNames (= champ geoname_id de la BDD), ce n'est pas une chaîne, on introduit en obbjet un nouveau concept.

```
crm:P3_has_note {nature}^^xsd:string ;
```

qui est décrit par une chaîne (= champ nature de la BDD)

```
crm:P8i_witnessed iceramm:bdresultats.php?site={idsite}/decouverte .
```

qui a fait l'objet d'une découverte (on introduit un nouveau concept qui fera l'objet d'un autre mapping où on indiquera que cette découverte a été faite par un Responsable à une Date et à un Lieu). Le point indique qu'on en a fini avec le sujet « Site ».

```
geonames:{geoname_id} a crm:E53_Place .
```

le concept de Lieu est précisé : l'URI GeoNames est un Lieu.

```
iceramm:bdresultats.php?site={idsite}/num a crm:E42_Identifier ;
```

le concept d'Identifiant de Site est précisé selon le concept du CIDOC

```
rdfs:label {num}^^xsd:string .
```

la valeur de cet identifiant est la chaîne de caractère « num » issue de la BDD.

/!\ Attention : les commentaires précédés de # dans cet exemple ne sont pas autorisés lors de la rédaction des mappings et ne doivent pas apparaître dans le fichier .obda.

3. Le typage des données

On remarque également l'utilisation de préfixes (skos:, crm:, xsd:) et la typologie de données littérales. Par exemple, avec `^^xsd:string` , on précise que le label du modèle est une chaîne (string). Voici une liste des typologies possibles :

- http://www.w3.org/2001/XMLSchema#string
- http://www.w3.org/2001/XMLSchema#int
- http://www.w3.org/2001/XMLSchema#integer
- http://www.w3.org/2001/XMLSchema#decimal
- http://www.w3.org/2001/XMLSchema#float
- http://www.w3.org/2001/XMLSchema#double
- http://www.w3.org/2001/XMLSchema#dateTime
- http://www.w3.org/2001/XMLSchema#boolean

Skos:prefLabel est à réserver à l'identification lisible d'un concept, celui qui sera affiché dans les résultats (le nom retenu pour le site, par exemple). Chaque concept doit dont nécessairement avoir un skos:prefLabel.

Rdfs:label est à réserver aux littéraux de E41_Appellation (et sous-classes), E42_Identifier et les dates

VII. Requêtes SQL

1. Principes généraux

Le langage utilisé pour sélectionner les éléments dans la base de données est le SQL (Structured Query Language) reconnu par la plupart des systèmes de gestion de base de données et particulièrement mySQL, PostGreSQL, SQLite, etc. Ce tutoriel ne détaille pas la syntaxe du langage SQL et nous conseillons de se rapporter aux multiples tutoriaux disponibles en ligne pour se familiariser avec ce langage.

Par exemple, pour récupérer dans la base de données les id des mobiliers, le mot-clé Pactols associé et le nom du mobilier, on écrira la requête SQL suivante dans la fenêtre Source :

```
SELECT id_mob, lien_pactols, concat('Mobilier ', id_mob) as label FROM mobilier
```

Ici, la requête récupère deux champs (« id_mob » et « lien_pactols ») de la table « mobilier » dans la base ArSol. Les champs sélectionnés dans cette requête servent directement à définir les instances qui vont alimenter notre modèle de triplets. Si le *mapping* l'exige, des requêtes plus complexes sont possibles en utilisant des jointures pour récupérer plusieurs champs issus de tables distinctes.

Edit Triples Map

Mapping ID: Mobilier

Target (Triples Template):

```
arsol:isite{Site}#Decouverte a sci:S19_Encounter_Event ; sci:O19_has_found_object
arsol:imobi{SiUsObj} . arsol:imobi{SiUsObj} a crm:E22_Man-Made_Object ;
crm:P48_has_preferred_identifier arsol:imobi{SiUsObj}#identifiant ; crm:P3_has_note
{G_descr}^^xsd:string ; skos:prefLabel {label}^^xsd:string ; sci:O19i_was_object_found_by
arsol:isite{Site}#Decouverte . arsol:imobi{SiUsObj}#identifiant a crm:E42_Identifier ; rdfs:label
{SiUsObj}^^xsd:string .
```

Source (SQL Query):

```
SELECT a.SiUsObj, a.Site, a.G_descr,
CONCAT('Mobilier ',a.SiUSObj) AS label
FROM mobilier AS a INNER JOIN sites AS s ON a.Site = s.Site
WHERE a.CoSiUsNum <>'' AND a.Site <> '' and s.Site <> ''
```

SQL Query results:

SiUsObj	Site	G_descr	label
AA1118.1_a	AA	?NI	Mobilier AA1118.1_a
AA3084.4_a	AA	?NI	Mobilier AA3084.4_a
AA5075.1_a	AA	?NI	Mobilier AA5075.1_a
AI1671.5	AI	V - bleu cobalt - 1 tesson s...	Mobilier AI1671.5
1.1	AJ	"potin turone au taureau co...	Mobilier 1.1
AA1738.3_a	AA	//as/	Mobilier AA1738.3_a
AA1880.15_a	AA	////	Mobilier AA1880.15_a

Execute the SQL query (100 rows)

Update Annuler

L'exemple ci-dessus présente un *mapping* pour la table « mobilier » de ArSol en appuyant le *mapping* sur le modèle de OpenArchaeo. La requête SQL récupère (SELECT) les champs de plusieurs tables via une jointure (INNER JOIN) et plusieurs conditions (WHERE) pour effectuer un filtre et ne récupérer que les données voulues (on écarte les enregistrements non renseignés).

Edit Triples Map

Mapping ID: Datation Mobilier

Target (Triples Template):

```
arsol:imobi{SiUsObj} a crm:E22_Man-Made_Object ; crm:P8i_witnessed arsol:imobi{SiUsObj}#Production_Use .
arsol:imobi{SiUsObj}#Production_Use a crm:E5_Event ; crm:P8_took_place_on_or_within arsol:imobi{SiUsObj} ;
crm:P4_has_time-span arsol:imobi{SiUsObj}#Datation . arsol:imobi{SiUsObj}#Datation a crm:E52_Time-Span ;
crm:P4i_is_time-span_of arsol:imobi{SiUsObj}#Production_Use ; skos:prefLabel {datelitt}^^xsd:string ;
crm:P82a_begin_of_the_begin {datedebut}^^xsd:dateTime ; crm:P82b_end_of_the_end {datefin}^^xsd:dateTime .
```

Source (SQL Query):

```
SELECT m.SiUsObj, m.datelitt,
CASE
WHEN m.dateinf LIKE '-%' THEN CONCAT(LPAD(LPAD(REPLACE(m.dateinf, '-', 0),4,'0'),5,'-'),'-01-01T00:00:00')
ELSE CONCAT(LPAD(m.dateinf, 4, '0'),'-01-01T00:00:00')
END as datedebut,
CASE
WHEN m.dateinf <> ' ' AND m.datesup = ' ' THEN CONCAT(LPAD(m.dateinf, 4, '0'), '-12-31T23:59:59')
WHEN m.dateinf <> ' ' AND m.datesup LIKE '-%'  THEN CONCAT(LPAD(LPAD(REPLACE(m.datesup, '-',
0),4,'0'),5,'-'),'-12-31T23:59:59')
ELSE CONCAT(LPAD(m.datesup, 4, '0'), '-12-31T23:59:59')
END as datefin
FROM mobilier as m WHERE m.CoSiUsNum <> '' AND m.dateinf <> '' AND m.datelitt <> '' AND m.Site <> ''
```

SQL Query results:

SiUsObj	datelitt	datedebut	datefin
AA2093.2_a	400/200	-0400-01-01T00:00:00	-0200-12-31T23:59:59
AA2095.2_a	400/200	-0400-01-01T00:00:00	-0200-12-31T23:59:59
AA2096.2_a	400/200	-0400-01-01T00:00:00	-0200-12-31T23:59:59
AA2097.1_a	400/200	-0400-01-01T00:00:00	-0200-12-31T23:59:59
AA2097.2_a	400/200	-0400-01-01T00:00:00	-0200-12-31T23:59:59
AA2098.1_a	400/200	-0400-01-01T00:00:00	-0200-12-31T23:59:59
AA2112.2_a	400/300	-0400-01-01T00:00:00	-0300-12-31T23:59:59
AA2426.3_a	400/300	-0400-01-01T00:00:00	-0300-12-31T23:59:59

Execute the SQL query (100 rows)

Update Annuler

L'exemple ci-dessus illustre l'utilisation d'une requête SQL pour filtrer et nettoyer les données un maximum avec des conditions CASE. Cela permet de créer des conditions de filtres et de modifier les données selon leur forme afin de les rendre utilisables dans le mapping. Pour les dates, on cherche à scinder la donnée en deux pour récupérer la date de début ou de fin de l'intervalle et la transformer dans un format xsd:dateTime (format de date pour le Web sémantique parmi les types de format pour les données littérales).

En fonction du modèle de données (ici OpenArchaeo) et de la structure de la base de données, il faudra scinder les *mappings* par unités logiques (voir Partie 7).

2. Restrictions SQL

- La fonction TRIM ne passe pas lors de la matérialisation (création des triplets) dans Ontop, même si ça semble passer lors du test de la requête SQL. Il faut jongler en utilisant REPLACE, SUBSTRING et LPAD pour effectuer les nettoyages de chaînes.

- GROUP BY : il faut faire un mapping spécifique juste pour le champ qu'on veut récupérer par GROUP BY (par exemple pour effectuer un COUNT).

- CONCAT : permet de concaténer plusieurs champs et éventuellement des chaînes fixes de caractères. Toutefois, ces chaînes fixes peuvent poser des problèmes d'encodage si des caractères spéciaux sont utilisés (ils sont remplacés par des « ? » lors de la matérialisation). Ce problème d'encodage n'apparaît pas pour les données récupérées dans la BDD. La

solution pour utiliser des caractères spéciaux dans une concaténation est de faire cette concaténation directement dans le mapping et non dans la requête SQL. La concaténation dans le mapping se fait simplement en encapsulant ce que l'il faut concaténer entre "guillemets doubles".

Exemple :

```
 mappingId      Lot-ceramtarget      iceramm:bdresultats.php?notice={notice}/lot
skos:prefLabel        "{ceram}        élément{pluriel}        céramique{pluriel}
typologique{pluriel}"^^xsd:string  .  source  SELECT  l.notice,  COUNT(t.lot)  AS
ceram, IF(COUNT(t.lot)>1,'s','') AS pluriel FROM lot AS l JOIN typologie AS t
ON l.notice = t.lot GROUP BY l.notice
```

Noter dans cet exemple la gestion des pluriels (s).

Résultat après matérialisation :

```
 <https://iceramm.huma-num.fr/bdresultats.php?notice=22/lot>
<http://www.w3.org/2004/02/skos/core#prefLabel>    "42    éléments    céramiques
typologiques"       .<https://iceramm.huma-num.fr/bdresultats.php?notice=23/lot>
<http://www.w3.org/2004/02/skos/core#prefLabel>    "5    éléments    céramiques
typologiques"       .<https://iceramm.huma-num.fr/bdresultats.php?notice=24/lot>
<http://www.w3.org/2004/02/skos/core#prefLabel>     "1     élément     céramique
typologique" .
```

À noter que selon les configurations, les caractères spéciaux ne passeront pas non plus s'ils sont écrits directement dans les mappings. Il faut essayer alors de supprimer l'option « sql_mode=ANSI » dans l'URL de connexion à la base de données et privilégier l'écriture des caractères spéciaux directement dans la requête SQL.

Exemple :

```
 mappingId Lot-ceram
```

```
 target  iceramm:bdresultats.php?notice={notice}/lot  skos:prefLabel  "{ceram}
{ect}"^^xsd:string .
```

```
 source  SELECT  l.notice,  COUNT(t.lot)  AS  ceram,  IF(COUNT(t.lot)>1,'éléments
céramiques typologiques','élément céramique typologique') AS ect FROM lot AS l
LEFT OUTER JOIN typologie AS t ON l.notice = t.lot GROUP BY l.notice
```

VIII. Quelle unité de mapping dans Ontop ?

1. Règles de structuration des mappings

Un *mapping* va associer un ou plusieurs triplets avec une requête SQL. Jusqu'à combien de triplets peut-on mettre au sein d'un même *mapping* ? Théoriquement, il n'y a pas de limites, mais il y a une logique et quelques règles à respecter pour le découpage.

- D'une manière générale, il convient de prévoir un pour chaque entité identifiée au sein du corpus : E27_Site, E25_Human-Made_Feature, E22_Human-Made_Object, E21_Person, E74_Group, E31_Document, etc. Cela peut correspondre bien souvent aux tables de la base de données.

- Ensuite, il convient aussi de distinguer les faisant la jointure entre des concepts distincts. Par exemple, un pourra générer un spécifique pour la jointure entre S19_Encounter_Event (la fouille) et E22_Human-Made_Object (le mobilier). Un de liens doit contenir les triplets dans les deux sens : S19 > 019 > E22 et E22 > O19i > S19.

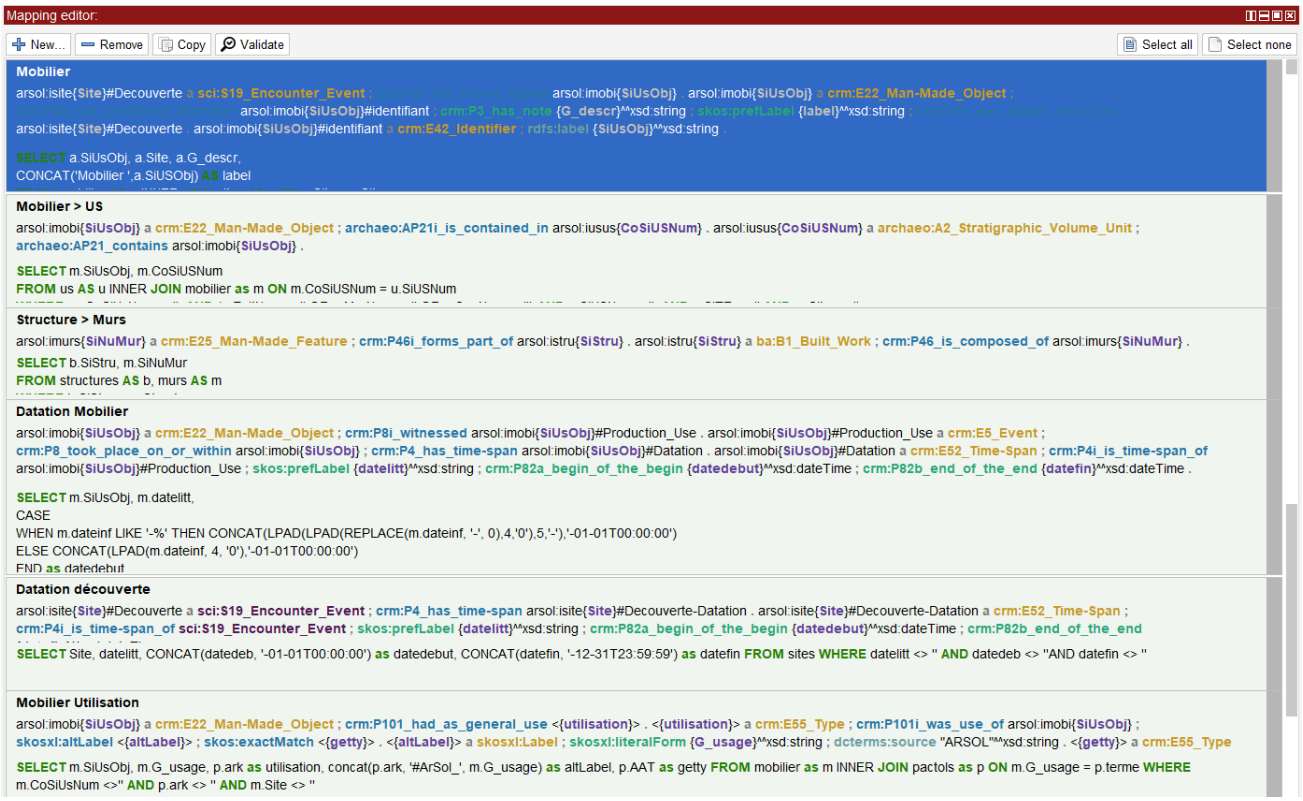
- Au sein d'une table de la base de données, une information peut être répétée (même si, dans l'idéal, il aurait fallu traiter l'information avec une table distincte, voire même avec aussi une table intermédiaire permettant de modéliser des liens de n à n), par exemple les auteurs de notice (une même personne peut être l'auteur de plusieurs notices). Dans ce cas, pour éviter de multiplier les triplets concernant ces auteurs, il convient de faire une requête SQL spécifique avec un GROUP BY pour ne pas multiplier les triplets identiques (voir partie sur les requêtes SQL).

- Si on veut récupérer une information de décompte (par exemple le nombre d'éléments constituant un lot pour fournir l'information dans la description du lot), la requête SQL passant par un COUNT() et GROUP BY

nécessitera d'avoir un spécifique (voir partie sur les requêtes SQL).

- Si des informations essentielles pour les sont absentes d'une table. Par exemple, il manque la date de quelques documents. Dans ce cas, si on fait un seul pour tous les documents qu'ils soient datés ou non, le génèrera des triplets avec des dates vides… ce qu'il faut absolument éviter. D'un autre côté, si on filtre lors de la requête SQL pour exclure les dates vides, les documents non datés n'apparaîtront pas dans les triplets alors qu'ils peuvent porter d'autres informations importantes, même s'ils n'ont pas de dates. La solution est encore d'externaliser le pour cette information. Pour reprendre l'exemple précédent, un est réalisé pour le document et ce qui le concerne, sauf la datation (partie exclue de la requête SQL) ; puis un second est réalisé pour les dates des documents en ne conservant que les documents datés (filtre SQL) ; le lien entre les deux se fait nécessairement par l'URI du document qui doit être commun dans les deux .

2. Exemple de structuration de mappings



Dans cet exemple de *mappings*, le Mobilier revient plusieurs fois dans la liste des *mappings*. En effet, si le *mapping* de la datation avait été également fait directement dans celui de Mobilier, Protégé-Ontop ne récupérerait que les mobiliers qui ont une datation. Or, dans notre cas, nous souhaitons récupérer dans un premier temps tous les mobiliers selon les conditions de requêtes présentés auparavant et ensuite récupérer l'information de la datation uniquement pour les mobiliers qui en disposent. Le raisonnement est similaire pour le lien entre le Mobilier et les liens Pactols pour l'« utilisation ». La séparation en plusieurs *mappings* pour un même type d'objet peut également être réalisée afin de bien de diviser la charge des requêtes SQL et éviter d'enchaîner des requêtes avec trop de jointures.

La rédaction d'un *mapping* demande donc de la rigueur, de la logique, de l'économie et une bonne connaissance de la structure des données.

Protégé pouvant montrer parfois quelques signes de faiblesse, il est indispensable de sauvegarder régulièrement les *mappings*.

3. Mappings pour les dates

Le cas des dates nécessite un peu d'attention. Le format attendu pour les dates dans le triplestore ressemble peu aux formats des dates stockées dans les bases des données (BDD).

Données de DATATION en entrée(telle que stockées dans la BDD)

- Soit une date seule (une année)
- Soit une fourchette : dans ce second cas, il faut stocker l'information dans 2 champs distincts (date_debut / date_fin). Si ce n'est pas le cas, il faut spliter l'information (soit en dur dans la BDD, soit dans la requête SQL si le formalisme est rigoureux). Dans le cas où une seule année est indiquée en guise de fourchette, on notera la même année pour le début et la fin.

Données de DATATION en sortie(fichier TTL pour OpenArchaeo)

- OpenArchaeo attend une date au format YYYY-MM-DDThh:mm:ss (ex : 1977-02-17T07:48:59)

Exemples de requêtes SQL

- Date de saisie d'une notice au format 2020-09-14 :

SQL :

```
SELECT notice, datesaisie, CONCAT(datesaisie,'T00:00:00') AS datesaisiedeb, CONCAT(datesaisie,'T23:59:59') AS datesaisiefin
FROM lot
```

[Dans ce cas, on se contente de normaliser la date pour la machine à partir de la date de saisie]

Mapping :

```
iceramm:bdresultats.php?notice={notice}/metadata/datation a crm:E52_Time-Span ; skos:prefLabel {datesaisie}^^xsd:string ; crm:P82a_begin_of_the_begin {datesaisiedeb}^^xsd:dateTime ; crm:P82b_end_of_the_end {datesaisiefin}^^xsd:dateTime .
```

- Fourchette de datation avec 2 années (ex : 0600 / 0800) :

SQL :

```
SELECT notice, IF(dateinf=datesup, IF(dateinf LIKE '0%',SUBSTRING(l.dateinf,2),dateinf), CONCAT(IF(dateinf LIKE '0%',SUBSTRING(l.dateinf,2),dateinf),'-', IF(datesup LIKE '0%',SUBSTRING(l.datesup,2),datesup))) AS datelitt,
CONCAT(dateinf,'-01-01T00:00:00') AS datedebut,
CONCAT(datesup,'-12-31T23:59:59') AS datefin
FROM lot WHERE dateinf <> '0000' AND datesup <> '0000'
```

[Dans ce cas, en plus de la normalisation, puisqu'on a deux champs, on reconstruit un champ littéral par concaténation, en supprimant les zéro de début de chaîne.]

Mapping :

```
iceramm:bdresultats.php?notice={notice}/contexte crm:P4_has_time-span iceramm:bdresultats.php?notice={notice}/contexte/datation .
iceramm:bdresultats.php?notice={notice}/contexte/datation a crm:E52_Time-Span ; rdfs:label {datelitt}^^xsd:string ; crm:P82a_begin_of_the_begin {datedebut}^^xsd:dateTime ; crm:P82b_end_of_the_end {datefin}^^xsd:dateTime .
```

A chaque fois, il faut la datation en littéralepour l'affichage, ainsi que les bornes (date inférieureet date supérieure) en format normalisé pour la machine.

4. Gestion des doublons

Ce peut-être par exemple le cas des auteurs d'une notice : le modèle exige un seul skos :prefLabel, mais, si les données n'ont pas été correctement nettoyées et homogénéisées, on a parfois pour un même concept d'auteur (VIAF, HAL…)

plusieurs façons de l'orthographier dans la base (Husi Philippe, Husi, Husi Ph, Husi Ph., etc…). Il faut donc externaliser le mapping qui fait le lien entre le concept d'auteur et son nom avec une requête SLQ permettant de ne conserver qu'une seule façon de nommer l'auteur.

```
mappingIdNotice_Auteur
target <{auteur_id}> a crm:E21_Person ; skos:prefLabel {auteur}^^xsd:string .
source SELECT if(l.auteur='','anonyme',l.auteur) AS auteur, l.auteur_id FROM lot AS l GROUP BY l.auteur_id
```

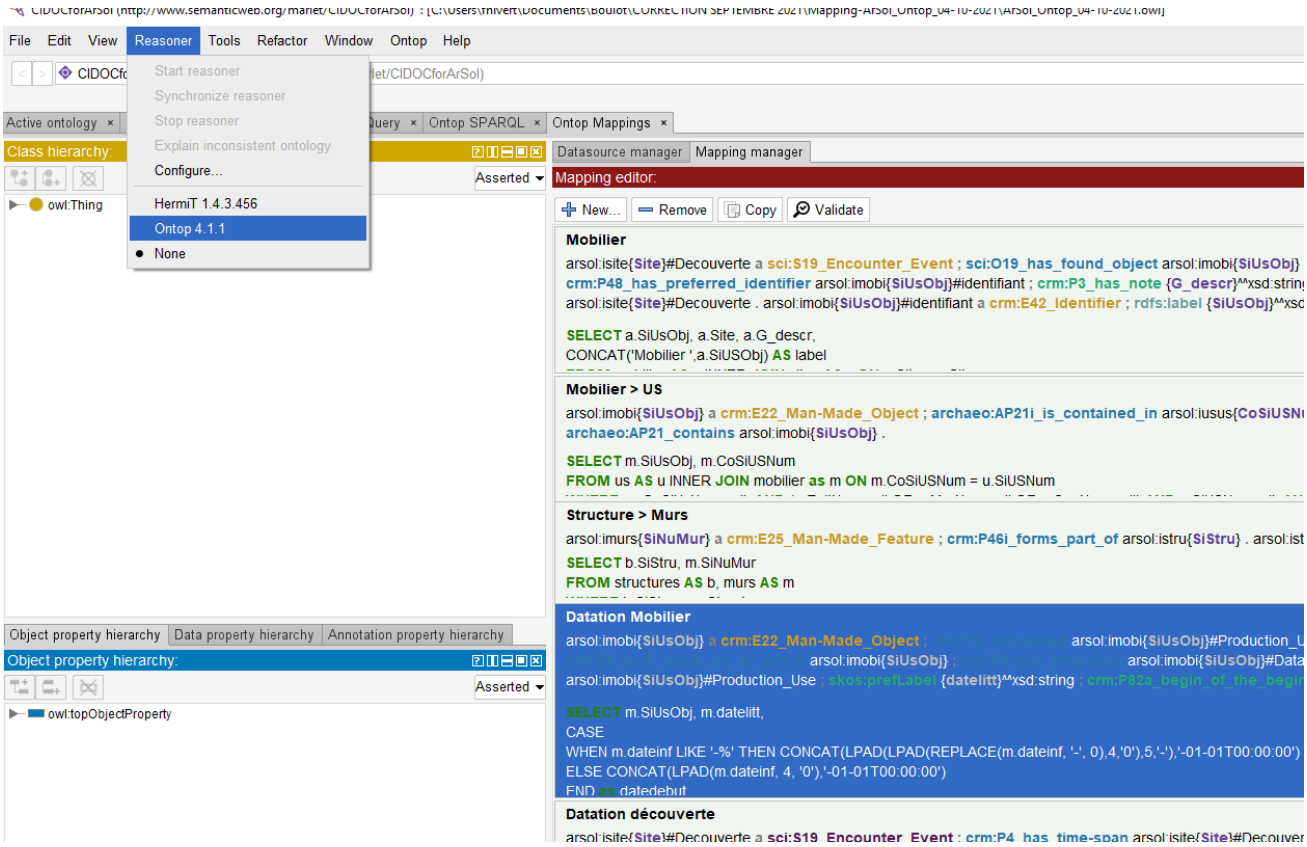
IX. Matérialisation à l'aide de Protégé-Ontop ou de Ontop CLI

Une fois le *mapping* terminé, comment transformer les données en triplet RDF ?

La matérialisation est le processus par lequel l'application va transformer des données issues d'une base de données en triplets RDF en se servant des modèles de triplets mis en place avec le *mapping*.

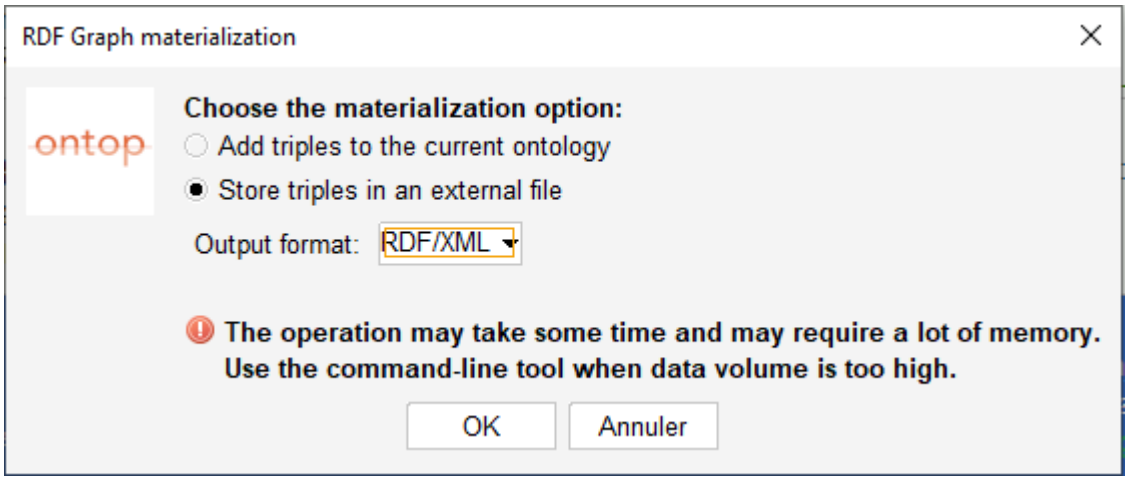
Deux méthodes de transformations sont possibles, l'une est accessible directement depuis Protégé, l'autre méthode demande de télécharger le client Ontop (Ontop CLI) et se lance depuis un terminal en lignes de commandes (cmd.exe par exemple sous Windows).

1. Transformer dans Protégé-Ontop



Dans le menu « *Reasoner* », choisir « Ontop x.x.x ». L'option « *Start Reasoner* » est alors disponible : l'activer.

Ensuite, dans le menu « Ontop », cliquer sur « *Materialize Triple …* » afin de démarrer le processus de transformation.



Dans la fenêtre qui s'ouvre, choisir l'option « *Store triples in an external file* » et choisir le format « Turtle ». Valider avec OK et choisir le chemin où le fichier sera enregistré. Cette méthode va créer un fichier *turtle* qui contiendra tous les triplets formalisés dans le *mapping* en les peuplant des instances de la base de données en fonction des requêtes SQL associées.

X. Transformer dans Ontop CLI

Un mapping génère plusieurs fichiers :

- un fichier .ttl ou .owl contenant les ontologies chargées dans le projet
- un fichier contenant les paramètres de connexion à la base de données
- un fichier .q contenant des requêtes SPARQL enregistrées pour tester le
- un fichier .obda () contenant les .

ArSol_Ontop_04-10-2021.obda	05/10/2021 09:11	Fichier OBDA	16 Ko
ArSol_Ontop_04-10-2021.owl	23/09/2021 11:40	Fichier OWL	1 005 Ko
ArSol_Ontop_04-10-2021.properties	23/09/2021 11:40	Fichier PROPERTIES	1 Ko
ArSol_Ontop_04-10-2021.q	23/09/2021 11:40	Fichier Q	1 Ko

Le fichier OBDA est modifiable directement depuis un éditeur de fichier TXT comme le blocnote ou les éditeurs de codes. Toutefois, il faut faire attention à ce qu'il n'y ait aucun retour chariot au sein du *mapping* (même si on trouve cela plus lisible). Les retours chariots au sein d'un *mapping* rendent le fichier impossible à ouvrir ensuite dans Ontop.

Il faut ensuite télécharger le .zip à ce lien : https://github.com/ontop/ontop/releases/ et choisir le lien ontop-cli-x.x.x.zip

L'archive ZIP contient les fichiers suivants. Dézipper le dossier.

Nom	Modifié le	Type	Taille
copyright	30/09/2021 09:07	Dossier de fichiers	
jdbc	30/09/2021 09:07	Dossier de fichiers	
lib	30/09/2021 09:07	Dossier de fichiers	
log	30/09/2021 09:07	Dossier de fichiers	
logback.xml	23/07/2021 17:29	Fichier XML	1 Ko
ontop	23/07/2021 17:29	Fichier	2 Ko
ontop.bat	23/07/2021 17:29	Fichier de comma...	1 Ko
ontop-completion.sh	23/07/2021 17:29	Shell Script	18 Ko
README.md	23/07/2021 17:29	Markdown file	3 Ko

Une fois le dossier dézippé, l'ouvrir. Positionner la souris dans la fenêtre mais pas sur un fichier ou dossier, maintenir enfoncée la touche *Shift*, faire un clic-droit dans la fenêtre du dossier et choisir « Ouvrir la fenêtre Powershell ici » ou bien lancer depuis Windows cmd.exe et indiquer le chemin du dossier pour se positionner dedans.

Pour plus de facilité, copier/coller les quatre fichiers obtenus du mapping dans ce dossier même dossier.

Dans le terminal, il faut écrire une commande pour lancer la transformation en triplet :

```
.\ontop.bat materialize -t fichier.owl -m fichier.obda -p fichier.properties -f turtle -o fichier_output.ttl --disable-reasoning
```

`materialize` est la commande lançant un processus de transformation avec plusieurs paramètres :

- « -t » doit être suivi du chemin vers le fichier d'ontologie, au format .ttl ou .owl dans cet exemple ;
- « -m » doit être suivi du chemin vers le fichier .obda qui contient les ;
- « -p » doit être suivi du chemin vers le fichier .properties pour se connecter à l-a base de données ;
- « -f » doit être suivi du format souhaité pour le fichier de sortie, par défaut le format sera en .rdf. Ici, on précise que l'on veut un fichier au format turtle ;
- « -o » doit être suivi du chemin et du nom de fichier avec son extension où le fichier de sortie sera enregistré (l'extension de ce fichier doit correspondre au format défini en « -f » ;
- « --disable-reasoning » est un paramètre qui indique que l'on ne souhaite pas d'inférences (conseillé pour ne généré que les triplets définis par le modèle de données et non tous les triplets inférés par la structure des ontologies utilisées).

Appuyer sur Enter pour valider. Attendre quelques instants et le fichier de sortie avec les triplets sera généré à l'emplacement indiqué.

1. Transformer par un fichier .bat

Une autre solution consiste à créer dans le dossier d'Ontop ou Ontop CLI un fichier TXT, le renommer en « do-triplets.bat » par exemple et à l'éditer (avec BlocNote ou éditeur de code) et copier l'exemple suivant en l'adaptant avec les paramétrages vus précédemment :

```
set SCRIPT_HOME=C:\MyPC\Ontop\ontop-distribution-x.x.0 [1]
cd %SCRIPT_HOME%
ontop materialize --ontology ../MyProject.ttl[2] --properties ../MyProject.properties[3] --mapping ../MyProject.obda[4] --output output.ttl[5] --format turtle --disable-reasoning
```

[1]=url du dossier contenant Ontop et où sera stocker le fichier de sortie

[2]=Fichier de référence de l’ontologie du projet (CIDOC CRM par exemple)

[3]=Fichier des propriétés (connexion JDBC avec la base de données)

[4]=Fichier OBDA contenant les mappings

[5]=nom du fichier de sortie (extension .TTL)

2. Méthode d’identification des erreurs

Tout le processus effectué par Ontop CLI est détaillé dans la fenêtre de commande. En cas d’erreur, le processus s’arrête et la génération du fichier est interrompue, laissant celui-ci incomplet, voire vide (0ko) et donc inexploitable. Lire les lignes de commandes juste avant que l’erreur ne survienne permet parfois d’identifier le problème. Plusieurs raisons peuvent expliquer l’erreur :

- Erreur dans le en lui-même. La syntaxe est forcément correcte puisque sinon Ontop n’aurait pas autorisé l’enregistrement du . Il faut donc orienter ses recherches du côté du modèle et de l’ontologie sur lequel s’appuie ce modèle. A-t-on bien respecté le modèle de données ? Celui-ci est-il robuste ? Les propriétés utilisées sont-elles autorisées entre les classes qui constituent son domaine et son co-domaine ? etc.

- Erreur dans la requête SQL. Il peut être utile de tester la requête SQL directement dans l’application de gestion de la base de données (phpMyAdmin, PGadmin, etc.) pour s’assurer que celle-ci est correcte (même si elle passe avec succès le test lors de la rédaction des). Certaines fonctionnalités SQL sont autorisées par les SGBD-R, par le processus de test d’Ontop au moment de la rédaction des mappings, mais pas pour la matérialisation (voir Partie 6.2).

- En cas d’erreur, la meilleure méthode est de procéder par étape en testant chaque un par un. Il s’agit de faire une copie du complet, puis de n’activer que le premier (en supprimant tous les autres – d’où l’intérêt de la copie) et de lancer la matérialisation pour vérifier que le fonctionne. Tant que la matérialisation s’effectue correctement, on continue à ajouter les un par un jusqu’à découvrir le fautif sur lequel porter notre attention.