Skosprovider_sqlalchemy Documentation

Release 2.1.0

Flanders Heritage

Mar 30, 2023

Contents

This library offers an implementation of the `skosprovider.providers.VocabularyProvider` interface that uses a SQLALchemy backend. While a `VocabularyProvider` is a read-only interface, the underlying SQLAlchemy *`domain model`* is fully writeable.

This library is fully integrated into Atramhasis, an online open source editor for SKOS vocabularies.

CHAPTER 1

Support

If you have questions regarding Skosprovider_SQLAlchemy, feel free to contact us. Any bugs you find or feature requests you have, you can add to our issue tracker. If you're unsure if something is a bug or intentional, or you just want to have a chat about this library or SKOS in general, feel free to join the Atramhasis discussion forum. While these are separate software projects, they are being run by the same people and they integrate rather tightly.

1.1 Setup

1.1.1 Installation

Installation of Skosprovider_sqlalchemy is easily done using **pip**.

```
$ pip install skosprovider_sqlalchemy
```

1.1.2 Creating a database

Since Skosprovider_sqlalchemy implements the SkosProvider interface with a relational database as a backend, you first need to create this database. To do this, please follow the instructions of your database software. If you're working with SQLite, you don't need to do anything.

Note: Because Skosprovider_sqlalchemy uses SQLAlchemy as an ORM layer, it's not tailored to any specific database. The codebase is continuously tested on both SQLite and PostgreSQL. Other databases are untested by us, but as long as they are supported by SQLAlchemy, they should work.

Once your database has been created, you can initialise it with the necessary database tables that will contain your SKOS vocabularies and concepts.

```
$ init_skos_db sqlite:///vocabs.db
```

Let's have a look at what this script did.

```
$ sqlite3 vocabs.db
SQLite version 3.7.9 2011-11-01 00:52:41
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .tables
collection_concept            conceptscheme_note
concept                       label
concept_hierarchy_collection  labeltype
concept_hierarchy_concept     language
concept_label                 match
concept_note                  matchtype
concept_related_concept       note
conceptscheme                 notetype
conceptscheme_label           visitation
```

1.1.3 Upgrading from skosprovider_sqlalchemy 1.x to 2.x

A change in the models has been made which requires a database upgrade. The "concept" table's "concept_id" column has changed from being an int to a string.

Existing databases will therefor require a small change to update table scheme. Typically this will look like:

```
ALTER TABLE concept ALTER COLUMN concept_id TEXT NOT NULL;
```

1.2 API Documentation

1.2.1 Providers module

class skosprovider_sqlalchemy.providers.**SQLAlchemyProvider**(*metadata*, *session*, ***kwargs*)

A `skosprovider.providers.VocabularyProvider` that uses SQLAlchemy as backend.

expand(*concept_id*)

Expand a concept or collection to all it's narrower concepts.

This method should recurse and also return narrower concepts of narrower concepts.

If the id passed belongs to a `skosprovider.skos.Concept`, the id of the concept itself should be include in the return value.

If the id passed belongs to a `skosprovider.skos.Collection`, the id of the collection itself must not be present in the return value In this case the return value includes all the member concepts and their narrower concepts.

Parameters **id** – A concept or collection id.

Return type A list of id's or *False* if the concept or collection doesn't exist.

expand_strategy = 'recurse'

Determines how the expand method will operate. Options are:

- *recurse*: Determine all narrower concepts by recursivly querying the database. Can take a long time for concepts that are at the top of a large hierarchy.

- *visit*: Query the database's `Visitation` table. This table contains a nested set representation of each conceptscheme. Actually creating the data in this table needs to be scheduled.

find(*query*, ***kwargs*)

Find concepts that match a certain query.

Currently query is expected to be a dict, so that complex queries can be passed. You can use this dict to search for concepts or collections with a certain label, with a certain type and for concepts that belong to a certain collection.

```
# Find anything that has a label of church.
provider.find({'label': 'church'})

# Find all concepts that are a part of collection 5.
provider.find({'type': 'concept', 'collection': {'id': 5}})

# Find all concepts, collections or children of these
# that belong to collection 5.
provider.find({'collection': {'id': 5, 'depth': 'all'}})

# Find anything that has a label of church.
# Preferentially display a label in Dutch.
provider.find({'label': 'church'}, language='nl')

# Find anything that has a match with an external concept
# Preferentially display a label in Dutch.
provider.find({
    'matches': {
        'uri': 'http://id.python.org/different/types/of/trees/nr/1/the/larch'
    }}, language='nl')

# Find anything that has a label of lariks with a close match to an external␣
↪concept
# Preferentially display a label in Dutch.
provider.find({
    'label': 'lariks',
    'matches': {
        'type': 'close',
        'uri': 'http://id.python.org/different/types/of/trees/nr/1/the/larch'
    }}, language='nl')
```

Parameters

- **query** – A dict that can be used to express a query. The following keys are permitted:
  - *label*: Search for something with this label value. An empty label is equal to searching for all concepts.
  - *type*: Limit the search to certain SKOS elements. If not present or *None*, *all* is assumed:
    - *concept*: Only return `skosprovider.skos.Concept` instances.
    - *collection*: Only return `skosprovider.skos.Collection` instances.
    - *all*: Return both `skosprovider.skos.Concept` and `skosprovider.skos.Collection` instances.
  - *collection*: Search only for concepts belonging to a certain collection. This argument should be a dict with two keys:
    - *id*: The id of a collection. Required.

* *depth*: Can be *members* or *all*. Optional. If not present, *members* is assumed, meaning only concepts or collections that are a direct member of the collection should be considered. When set to *all*, this method should return concepts and collections that are a member of the collection or are a narrower concept of a member of the collection.

– ***matches*: Search only for concepts having a match to a certain** external concept. Since collections can't have matches, this automatically excludes collections. The argument with two keys:

* *uri*: The uri of the concept to match. Required.

* *type*: The type of match, see `matchtypes` for the full list of options.

- **`language`** (*`string`*) – Optional. If present, it should be a language-tag. This language-tag is passed on to the underlying providers and used when selecting the label to display for each concept.
- **`sort`** (*`string`*) – Optional. If present, it should either be *id*, *label* or *sortlabel*. The *sortlabel* option means the providers should take into account any *sortLabel* if present, if not it will fallback to a regular label to sort on.
- **`sort_order`** (*`string`*) – Optional. What order to sort in: *asc* or *desc*. Defaults to *asc*

Returns

A `lst` of concepts and collections. Each of these is a dict with the following keys:

- id: id within the conceptscheme
- uri: URI of the concept or collection
- type: concept or collection
- label: A label to represent the concept or collection. It is determined by looking at the *language* parameter, the default language of the provider and finally falls back to *en*.

`get_all`(***kwargs*)

Returns all concepts and collections in this provider.

Parameters

- **`language`** (*`string`*) – Optional. If present, it should be a language-tag. This language-tag is passed on to the underlying providers and used when selecting the label to display for each concept.
- **`sort`** (*`string`*) – Optional. If present, it should either be *id*, *label* or *sortlabel*. The *sortlabel* option means the providers should take into account any *sortLabel* if present, if not it will fallback to a regular label to sort on.
- **`sort_order`** (*`string`*) – Optional. What order to sort in: *asc* or *desc*. Defaults to *asc*

Returns

A `lst` of concepts and collections. Each of these is a dict with the following keys:

- id: id within the conceptscheme
- uri: URI of the concept or collection
- type: concept or collection

- label: A label to represent the concept or collection. It is determined by looking at the *language* parameter, the default language of the provider and finally falls back to *en*.

get_by_id(*concept_id*)
Get all information on a concept or collection, based on id.

Providers should assume that all id's passed are strings. If a provider knows that internally it uses numeric identifiers, it's up to the provider to do the typecasting. Generally, this should not be done by changing the id's themselves (eg. from int to str), but by doing the id comparisons in a type agnostic way.

Since this method could be used to find both concepts and collections, it's assumed that there are no id collisions between concepts and collections.

Return type `skosprovider.skos.Concept` or `skosprovider.skos.Collection` or *False* if the concept or collection is unknown to the provider.

get_by_uri(*uri*)
Get all information on a concept or collection, based on a URI.

This method will only find concepts or collections whose URI is actually stored in the database. It will not find anything that has no URI in the database, but does have a matching URI after generation.

Return type `skosprovider.skos.Concept` or `skosprovider.skos.Collection` or *False* if the concept or collection is unknown to the provider.

get_children_display(*thing_id*, ***kwargs*)
Return a list of concepts or collections that should be displayed under this concept or collection.

Parameters **thing_id** – A concept or collection id.

Return type A list of concepts and collections. For each an id is present and a label. The label is determined by looking at the ***kwargs* parameter, the default language of the provider and falls back to *en* if nothing is present. If the id does not exist, return *False*.

get_top_concepts(***kwargs*)
Returns all top-level concepts in this provider.

Top-level concepts are concepts that have no broader concepts themselves. They might have narrower concepts, but this is not mandatory.

Parameters

- **language** (*string*) – Optional. If present, it should be a language-tag. This language-tag is passed on to the underlying providers and used when selecting the label to display for each concept.
- **sort** (*string*) – Optional. If present, it should either be *id*, *label* or *sortlabel*. The *sortlabel* option means the providers should take into account any *sortLabel* if present, if not it will fallback to a regular label to sort on.
- **sort_order** (*string*) – Optional. What order to sort in: *asc* or *desc*. Defaults to *asc*

Returns

A `lst` of concepts, NOT collections. Each of these is a dict with the following keys:

- id: id within the conceptscheme
- uri: URI of the concept or collection
- type: concept or collection

- label: A label to represent the concept or collection. It is determined by looking at the *language* parameter, the default language of the provider and finally falls back to *en*.

get_top_display(***kwargs*)
Returns all concepts or collections that form the top-level of a display hierarchy.

As opposed to the *get_top_concepts()*, this method can possibly return both concepts and collections.

Return type Returns a list of concepts and collections. For each an id is present and a label. The label is determined by looking at the ***kwargs* parameter, the default language of the provider and falls back to *en* if nothing is present.

1.2.2 Models module

class skosprovider_sqlalchemy.models.**Collection**(***kwargs*)
A collection as know by skos.

class skosprovider_sqlalchemy.models.**Concept**(***kwargs*)
A concept as know by skos.

class skosprovider_sqlalchemy.models.**ConceptScheme**(***kwargs*)
A skos conceptscheme.

class skosprovider_sqlalchemy.models.**Initialiser**(*session*)
Initialises a database.

Adds necessary values for labelType, noteType and language to the database.

The list of languages added by default is very small and will probably need to be expanded for your local needs.

init_all()
Initialise all objects (labeltype, notetype, language).

init_labeltype()
Initialise the labeltypes.

init_languages()
Initialise the languages.

Only adds a small set of languages. Will probably not be sufficient for most use cases.

init_matchtypes()
Initialise the matchtypes.

init_notetype()
Initialise the notetypes.

class skosprovider_sqlalchemy.models.**Label**(*label*, *labeltype_id='prefLabel'*, *language_id=None*)
A label for a *Concept*, *Collection* or *ConceptScheme*.

class skosprovider_sqlalchemy.models.**LabelType**(*name*, *description*)
A labelType according to skos.

class skosprovider_sqlalchemy.models.**Language**(*id*, *name*)
A Language.

class skosprovider_sqlalchemy.models.**Match**(***kwargs*)
A match between a *Concept* in one ConceptScheme and those in another one.

class skosprovider_sqlalchemy.models.**MatchType**(*name*, *description*)
A matchType according to skos.

class skosprovider_sqlalchemy.models.**Note**(*note*, *notetype_id*, *language_id*, *markup=None*)
A note for a *Concept*, *Collection* or *ConceptScheme*.

class skosprovider_sqlalchemy.models.**NoteType**(*name*, *description*)
A noteType according to skos.

class skosprovider_sqlalchemy.models.**Source**(*citation*, *markup=None*)
The source where a certain piece of information came from.

class skosprovider_sqlalchemy.models.**Thing**(***kwargs*)
Abstract class for both *Concept* and *Collection*.

class skosprovider_sqlalchemy.models.**Visitation**(***kwargs*)
Holds several nested sets.

The visitation object and table hold several nested sets. Each *skosprovider_sqlalchemy.models.Visitation* holds the positional information for one conceptplacement in a certain nested set.

Each conceptscheme gets its own separate nested set.

skosprovider_sqlalchemy.models.**label**(*labels=[]*, *language='any'*, *sortLabel=False*)
Provide a label for a list of labels.

Deprecated since version 0.5.0: Please use skosprovider.skos.label(). Starting with *skosprovider 0.6.0*, the function can function on *skosprovider_sqlalchemy.models.Label* instances as well.

Parameters

- **labels** (*list*) – A list of *labels*.
- **language** (*str*) – The language for which a label should preferentially be returned. This should be a valid IANA language tag.
- **sortLabel** (*boolean*) – Should sortLabels be considered or not? If True, sortLabels will be preferred over prefLabels. Bear in mind that these are still language dependent. So, it's possible to have a different sortLabel per language.

Return type A *Label* or *None* if no label could be found.

skosprovider_sqlalchemy.models.**related_concepts_append_listener**(*target*, *value*, *initiator*)
Listener that ensures related concepts have a bidirectional relationship.

skosprovider_sqlalchemy.models.**related_concepts_remove_listener**(*target*, *value*, *initiator*)
Listener to remove a related concept from both ends of the relationship.

1.2.3 Utils module

class skosprovider_sqlalchemy.utils.**VisitationCalculator**(*session*)
Generates a nested set for a conceptscheme.

visit(*conceptscheme*)
Visit a skosprovider_sqlalchemy.models.Conceptscheme and calculate a nested set representation.

Parameters **conceptscheme** – A skosprovider_sqlalchemy.models.Conceptscheme for which the nested set will be calculated.

skosprovider_sqlalchemy.utils.**import_provider**(*provider: skosprovider.providers.VocabularyProvider*, *session: sqlalchemy.orm.session.Session*, *conceptscheme: skosprovider_sqlalchemy.models.ConceptScheme = None*) → skosprovider_sqlalchemy.models.ConceptScheme

Import a provider into a SQLAlchemy database.

Parameters

- **provider** – The skosprovider.providers.VocabularyProvider to import. Since the SQLAlchemy backend uses integers as keys, this backend should have id values that can be cast to int.
- **conceptscheme** – A skosprovider_sqlalchemy.models.Conceptscheme to import the provider into. This should be an empty scheme so that there are no possible id clashes. If no conceptscheme is provided, one will be created.
- **session** – A sqlalchemy.orm.session.Session.

Returns The conceptscheme that holds the concepts and collections. Either the same conceptscheme that was passed into the provider, or the one that was created by this function.

Return type skosprovider_sqlalchemy.models.Conceptscheme

1.3 History

1.3.1 2.1.0 (2023-03-30)

- **Minor BC break**: Changed the order of parameters to the import_provider function and make the conceptscheme argument optional. (#100)

1.3.2 2.0.1 (2023-03-20)

- Fixed an issue with import_provider still assuming ids are numeric. (#97)

1.3.3 2.0.0 (2023-01-19)

- **Major BC break**: Change concept.concept_id from Integer to String to allow storing concepts and collections with a non-numeric id. Existing instance will need to update their SQL database. Please consult the docs or the README for some help in doing so. (#87)
- Skosprovider_sqlalchemy now depends on SQLAlchemy 1.4 or higher and should be compatible with SQLAlchemy 2. Older versions of SQLAlchemy are no longer supported. (#90)
- Refactored the Skosprovider_sqlalchemy constructor to call the super constructor. (#95)
- Drop support for Python 3.6 and 3.7. Add support for Python 3.11. (#86)
- Drop pyup support. (#85)

1.3.4 1.0.0 (2021-12-21)

- Drop python 2 support (#80)
- Upgrade requirements (#78)
- Add a CITATION.cff file

1.3.5 0.6.0 (2020-07-29)

- Update to the latest skosprovider version and implement the *infer_concept_relations* attribute. (#53)
- Add the ability to query on matches in line with the latest skosprovider version. (#57)
- Drop the session decorator that was added in 0.4.0 since it did not fix the issue we wanted it to fix and it added a lot of overhead. A provider should now be passed a `sqlachemy.orm.session.Session` at startup, or a callable that returns such a session. (#64)
- Improved performance of getting the concept_scheme by caching it. (#71)
- Make querying a collection with depth=all possible. Before the provider would only provide the direct members of a collection. (#76)
- Drop support for Python 3.4 and 3.5. Add support for Python 3.7 and 3.8. This is also the last version to support Python 2. (#62)

1.3.6 0.5.2 (2018-11-13)

- Update a lot of dependencies.
- Add __str__ implementations to the model classes. (#43)

1.3.7 0.5.1 (2016-10-05)

- Catch linking errors when importing a provider and turn them into log warning. By linking errors we mean cases where one concept has a relation to a non-existing other concept. (#25)
- Allow building as wheel.

1.3.8 0.5.0 (2016-08-11)

- Update to skosprovider 0.6.0
- **Minor BC break**: A *skosprovider_sqlalchemy.models.Language* that gets cast to a string, now returns the language's ID (the IANA language code),as opposed to the language's description it would previously return.
- **Minor BC break**: The URI attribute has been made required for a *skosprovider_sqlalchemy.models.ConceptScheme*. Before it was optional, but it probably would have caused problems with skosprovider anyway.
- Due to the update to skosprovider 0.6.0, a new field *markup*, was added to a *skosprovider_sqlalchemy.models.Note*. When upgrading from a previous version of *skosprovider_sqlalchemy*, any databases created in that previous verions will need to be updated as well. Please add a field called *markup* to the *note* table.

- Inline with the skosprovider 0.6.0 update, a *languages* attribute was added to *skosprovider_sqlalchemy.models.ConceptScheme*. When upgrading from a previous version of *skosprovider_sqlalchemy*, any databases created with that previous verions will need to be updated as well. Please add a table called *conceptscheme_language* with fields *conceptscheme_id* and *language_id*. (#18)
- To comply with the skosprovider 0.6.0 update, the *sources* attribute was added to `skosprovider_sqlalchemy.models.Conceptscheme`, *skosprovider_sqlalchemy.models.Concept* and *skosprovider_sqlalchemy.models.Collection*. When upgrading from a previous version of *skosprovider_sqlalchemy*, any databases created with that previous verions will need to be updated as well. Please add a table *source* with fields *id*, *citation* and *markup*, a table *concept_source* with fields *concept_id* and *source_id* and a table *conceptscheme_source* with fields *conceptscheme_id* and *source_id*.
- All methodes that return a list have been modified in line with skosprovider 0.6.0 to support sorting. Sorting is possible on *id*, *uri*, *label* and *sortlabel*. The last two are language dependent. The *sortlabel* allows custom sorting of concepts. This can be used to eg. sort concepts representing chronological periods in chronological in stead of alphabetical order. (#20)
- To comply with the skosprovider 0.6.0 update, the deprecated `skosprovider_sqlalchemy.providers.SQLAlchemyProvider.expand_concept()` was removed.
- When importing a provider, check if the languages that are being used in the provider are already in our database. If not, validate them and add them to the database. In the past the entire import would fail if not all languages had previously been added to the database. (#14)
- When importing a provider, try to import as much information as possible about the concept_scheme that's attached to the provider. (#19)
- When querying for indvidual an conceptscheme or concept, use *joinedload* to reduce the number of queries needed to collect everything. (#15)
- Deprecated the *skosprovider_sqlalchemy.models.label()* function. Please use `skosprovider.skos.label()` from now once, since this function can now operate on both `skosprovider.skos.Label` and *skosprovider_sqlalchemy.models.Label* instances. This was the reason for the BC break in this release.

1.3.9 0.4.2 (2015-03-02)

- Make README work better on pypi.
- Fix a further problem with the length of language identifiers. Previous fix in 0.3.0 only fixed the length of the identifiers in the languages table, but not in the links from the labels and the notes to the language table. [BartSaelen]
- Added some documentation about setting up a database.

1.3.10 0.4.1 (2014-12-18)

- Fix a bug with the deletion of a Concept not being possible without having it's matches deleted first. [BartSaelen]

1.3.11 0.4.0 (2014-10-28)

- **Major BC break**: A provider is no longer passed a database session, but a database session maker. This change was needed to get the provider to function properly in threaded web applications. This will mean changing the code where you're creating your provider. In the past, you probably called a session maker first and then passed the result of this call to the provider. Now you should just pass the session maker itself and let the provider create the sessions for you.

- Different way of fetching the `ConceptScheme` for a provider. No longer fetches a conceptscheme at provider instantiation, but when needed. Otherwise we end up with a possibly very long cached version of a conceptscheme.

1.3.12 0.3.0 (2014-10-17)

- Update to skosprovider 0.4.0.
- Add `ConceptScheme` information to a provider so it can be attached to `Concept` objects that are handled by the provider.
- Let provider handle superordinates and subordinate arrays.
- Let provider add notes to collections.
- Added a *`Match`* model to handle matches. Expand the provider to actually provide information on these matches.
- Expand the field length for language identifiers. IANA suggests that identifiers up to 35 characters should be permitted. Updated our field length to 64 to have a bit of an extra buffer.

1.3.13 0.2.1 (2014-08-25)

- Switch to py.test
- Add Coveralls support for code coverage.
- Add ability to configure the SQLAlchemy URL used for testing. Allows testing on multiple RDBMS systems.
- Run Travis tests for both SQLite and Postgresql.
- Fix a bug in *`skosprovider_sqlalchemy.utils.import_provider()`* when dealing with narrower collections (#8). [cahytinne]
- Make the provider actually generate a URI if there's none in the database.

1.3.14 0.2.0 (2014-05-14)

- Compatibility with skosprovider 0.3.0
- Implement `skosprovider.providers.VocabularyProvider.get_by_uri()`.
- Implement `skosprovider.providers.VocabularyProvider.get_top_concepts()`.
- Implement `skosprovider.providers.VocabularyProvider.get_top_display()` and `skosprovider.providers.VocabularyProvider.get_children_display()`.
- Add a UniqueConstraint(conceptscheme_id, concept_id) to Thing. (#3)
- Rename the *colletions* attribute of *`skosprovider_sqlalchemy.models.Thing`* to *member_of*. (#7)

1.3.15 0.1.2 (2013-12-06)

- Pinned dependency on skosprovider < 0.3.0
- Pass data to `skosprovider.skos.Concept` using keywords in stead of positions.

1.3.16 0.1.1 (2013-11-28)

- Fixed a bug with collection members being passed instead of their ids.
- Fixed another bug where model ids were used instead of concept ids.

1.3.17 0.1.0

- Initial version
- Implementation of a SKOS domain model in SQLAlchemy.
- Implementation of a `skosprovider.providers.VocabularyProvider` that uses this model.
- Can query a hierarchy recursively or using nested sets.
- Utility function to import a `skosprovider.providers.VocabularyProvider` in a database.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

s

Index

T

V