SQLAlchemy 2.0 Documentation

Release: **2.0.36** **CURRENT RELEASE** | Release Date: October 15, 2024

SQLAlchemy 2.0 Documentation

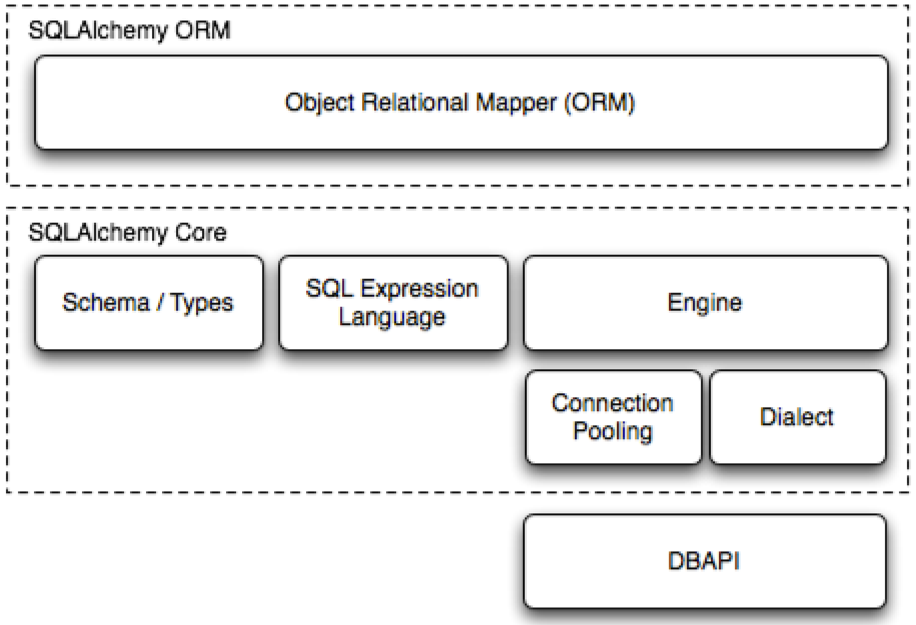
CURRENT RELEASE

Home | Download this Documentation

Search terms: search...

Overview

The SQLAlchemy SQL Toolkit and Object Relational Mapper is a comprehensive set of tools for working with databases and Python. It has several distinct areas of functionality which can be used individually or combined together. Its major components are illustrated below, with component dependencies organized into layers:



Above, the two most significant front-facing portions of SQLAlchemy are the **Object Relational Mapper (ORM)** and the **Core**.

Core contains the breadth of SQLAlchemy's SQL and database integration and description services, the most prominent part of this being the **SQL Expression Language**.

The SQL Expression Language is a toolkit on its own, independent of the ORM package, which provides a system of constructing SQL expressions represented by composable objects, which can then be "executed" against a target database within the scope of a specific transaction, returning a result set. Inserts, updates and deletes (i.e. DML) are achieved by passing SQL expression objects representing these statements along with dictionaries that represent parameters to be used with each statement.

The ORM builds upon Core to provide a means of working with a domain object model mapped to a database schema. When using the ORM, SQL statements are constructed in mostly the same way as when using Core, however the task of DML, which here refers to the persistence of business objects in a database, is automated using a pattern called unit of work, which translates changes in state against mutable objects into INSERT, UPDATE and DELETE constructs which are then invoked in terms of those objects. SELECT statements are also augmented by ORM-specific automations and object-centric querying capabilities.

Whereas working with Core and the SQL Expression language presents a schema-centric view of the database, along with a programming paradigm that is oriented around immutability, the ORM builds on top of this a domain-centric view of the database with a programming paradigm that is more explicitly object-oriented and reliant upon mutability. Since a relational database is itself a mutable service, the difference is that Core/SQL Expression language is command oriented whereas the ORM is state oriented.

Documentation Overview

The documentation is separated into four sections:

- SQLAlchemy Unified Tutorial - this all-new tutorial for the 1.4/2.0 series of SQLAlchemy introduces the entire library holistically, starting from a description of Core and working more and more towards ORM-specific concepts. New users, as well as users coming from the 1.x series of SQLAlchemy, should start here.
- SQLAlchemy ORM - In this section, reference documentation for the ORM is presented.
- SQLAlchemy Core - Here, reference documentation for everything else within Core is presented. SQLAlchemy engine, connection, and pooling services are also described here.

- Dialects - Provides reference documentation for all dialect implementations, including DBAPI specifics.

Code Examples

Working code examples, mostly regarding the ORM, are included in the SQLAlchemy distribution. A description of all the included example applications is at ORM Examples.

There is also a wide variety of examples involving both core SQLAlchemy constructs as well as the ORM on the wiki. See Theatrum Chemicum.

Installation Guide

Supported Platforms

SQLAlchemy supports the following platforms:

- cPython 3.7 and higher
- Python-3 compatible versions of PyPy

Changed in version 2.0: SQLAlchemy now targets Python 3.7 and above.

AsyncIO Support

SQLAlchemy's `asyncio` support depends upon the greenlet project. This dependency will be installed by default on common machine platforms, however is not supported on every architecture and also may not install by default on less common architectures. See the section Asyncio Platform Installation Notes (Including Apple M1) for additional details on ensuring asyncio support is present.

Supported Installation Methods

SQLAlchemy installation is via standard Python methodologies that are based on setuptools, either by referring to `setup.py` directly or by using pip or other setuptools-compatible approaches.

Install via pip

When `pip` is available, the distribution can be downloaded from PyPI and installed in one step:

```
pip install SQLAlchemy
```

This command will download the latest **released** version of SQLAlchemy from the Python Cheese Shop and install it to your system. For most common platforms, a Python Wheel file will be downloaded which provides native Cython / C extensions prebuilt.

In order to install the latest **prerelease** version, such as `2.0.0b1`, pip requires that the `--pre` flag be used:

```
pip install --pre SQLAlchemy
```

Where above, if the most recent version is a prerelease, it will be installed instead of the latest released version.

Installing manually from the source distribution

When not installing from pip, the source distribution may be installed using the `setup.py` script:

```
python setup.py install
```

The source install is platform agnostic and will install on any platform regardless of whether or not Cython / C build tools are installed. As the next section Building the Cython Extensions details, `setup.py` will attempt to build using Cython / C if possible but will fall back to a pure Python installation otherwise.

Building the Cython Extensions

SQLAlchemy includes Cython extensions which provide an extra speed boost within various areas, with a current emphasis on the speed of Core result sets.

Changed in version 2.0: The SQLAlchemy C extensions have been rewritten using Cython.

`setup.py` will automatically build the extensions if an appropriate platform is detected, assuming the Cython package is installed. A complete manual build looks like:

```
# cd into SQLAlchemy source distribution
cd path/to/sqlalchemy

# install cython
pip install cython

# optionally build Cython extensions ahead of install
python setup.py build_ext

# run the install
python setup.py install
```

Source builds may also be performed using **PEP 517** techniques, such as using build:

```
# cd into SQLAlchemy source distribution
cd path/to/sqlalchemy

# install build
pip install build

# build source / wheel dists
python -m build
```

If the build of the Cython extensions fails due to Cython not being installed, a missing compiler or other issue, the setup process will output a warning message and re-run the build without the Cython extensions upon completion, reporting final status.

To run the build/install without even attempting to compile the Cython extensions, the `DISABLE_SQLALCHEMY_CEXT` environment variable may be specified. The use case for this is either for special testing circumstances, or in the rare case of compatibility/build issues not overcome by the usual “rebuild” mechanism:

```
export DISABLE_SQLALCHEMY_CEXT=1; python setup.py install
```

Installing a Database API

SQLAlchemy is designed to operate with a DBAPI implementation built for a particular database, and includes support for the most popular databases. The individual database sections in Dialects enumerate the available DBAPIs for each database, including external links.

Checking the Installed SQLAlchemy Version

This documentation covers SQLAlchemy version 2.0. If you're working on a system that already has SQLAlchemy installed, check the version from your Python prompt like this:

```
>>> import sqlalchemy
>>> sqlalchemy.__version__
2.0.0
```

Next Steps

With SQLAlchemy installed, new and old users alike can Proceed to the SQLAlchemy Tutorial.

1.x to 2.0 Migration

Notes on the new API released in SQLAlchemy 2.0 is available here at SQLAlchemy 2.0 - Major Migration Guide.

Previous: Table of Contents Next: SQLAlchemy Unified Tutorial