**ORM Tools for Python**

* **Django**

**License:** BSD License (BSD-3-Clause)

* The **Django** web framework includes a default object-relational mapping layer (**ORM**) that can be used to interact with application data from various relational databases such as SQLite, PostgreSQL and MySQL.
* The **Django ORM** is an implementation of the object-relational mapping (**ORM**) concept.
* **Django** is a [Python](https://en.wikipedia.org/wiki/Python_(programming_language))-based [free and open-source](https://en.wikipedia.org/wiki/Free_and_open-source_software) [web framework](https://en.wikipedia.org/wiki/Web_framework) that follows the model-template-views (MTV) [architectural pattern](https://en.wikipedia.org/wiki/Architectural_pattern_(computer_science)).
* Django's primary goal is to ease the creation of complex, database-driven websites. The framework emphasizes [reusability](https://en.wikipedia.org/wiki/Reusability) and "pluggability" of components, less code; low coupling, rapid development, and the principle of [don't repeat you](https://en.wikipedia.org/wiki/Don%27t_repeat_yourself)rself.
* Python is used throughout, even for settings, files, and data models. Django also provides an optional administrative [create, read, update and delete](https://en.wikipedia.org/wiki/Create,_read,_update_and_delete) interface that is generated dynamically through [introspection](https://en.wikipedia.org/wiki/Type_introspection) and configured via admin models.
* Some well-known sites that use Django include [PBS](https://en.wikipedia.org/wiki/PBS), [Instagram](https://en.wikipedia.org/wiki/Instagram), [Mozilla](https://en.wikipedia.org/wiki/Mozilla_Foundation), [The Washington Times](https://en.wikipedia.org/wiki/The_Washington_Times), [Disqus](https://en.wikipedia.org/wiki/Disqus), [Bitbucket](https://en.wikipedia.org/wiki/Bitbucket), and [Nextdoor](https://en.wikipedia.org/wiki/Nextdoor).

**Features**

## Rapid Development

Django was designed with the intention to make a framework which takes less time to build web application. The project implementation phase is a very time taken but Django creates it rapidly.

## Secure

Django takes security seriously and helps developers to avoid many common security mistakes, such as SQL injection, cross-site scripting, cross-site request forgery etc. Its user authentication system provides a secure way to manage user accounts and passwords.

## Scalable

Django is scalable in nature and has ability to quickly and flexibly switch from small to large scale application project.

## Fully loaded

Django includes various helping task modules and libraries which can be used to handle common Web development tasks. Django takes care of user authentication, content administration, site maps, RSS feeds etc.

## Versatile

Django is versatile in nature which allows it to build applications for different-different domains. Now days, Companies are using Django to build various types of applications like: content management systems, social networks sites or scientific computing platforms etc.

## Open Source

Django is an open source web application framework. It is publicly available without cost. It can be downloaded with source code from the public repository. Open source reduces the total cost of the application development.

## Vast and Supported Community

Django is a one of the most popular web framework. It has widely supportive community and channels to share and connect.

**Bundled applications**

The main Django distribution also bundles a number of applications in its "contrib" package, including:

* an extensible authentication system
* the dynamic administrative interface
* tools for generating [RSS](https://en.wikipedia.org/wiki/RSS_(file_format)) and [Atom](https://en.wikipedia.org/wiki/Atom_(standard)) syndication feeds
* a "Sites" framework that allows one Django installation to run multiple websites, each with their own content and applications
* tools for generating [Google Sitemaps](https://en.wikipedia.org/wiki/Google_Sitemaps)
* built-in mitigation for [cross-site request forgery](https://en.wikipedia.org/wiki/Cross-site_request_forgery), [cross-site scripting](https://en.wikipedia.org/wiki/Cross-site_scripting), [SQL injection](https://en.wikipedia.org/wiki/SQL_injection), [password cracking](https://en.wikipedia.org/wiki/Password_cracking) and other typical web attacks, most of them turned on by default
* a framework for creating [GIS](https://en.wikipedia.org/wiki/Geographic_information_system) applications

**Install using pip:**

pip install Django

**Community/Support:**

1. Dev Community (<https://dev.to/t/django>)
2. Codementor (<https://www.codementor.io/community/topic/django>).

* **SQLAlchemy**

**License: MIT License**

* **SQLAlchemy** is an [open-source](https://en.wikipedia.org/wiki/Open-source_software) [SQL](https://en.wikipedia.org/wiki/SQL) toolkit and [object-relational mapper](https://en.wikipedia.org/wiki/Object-relational_mapping) (ORM) for the [Python programming language](https://en.wikipedia.org/wiki/Python_(programming_language)) released under the [MIT License](https://en.wikipedia.org/wiki/MIT_License).
* [**SQLAlchemy**](http://www.sqlalchemy.org/)is a well-regarded Python ORM because it gets the abstraction level "just right" and seems to make complex database queries easier to write than the Django ORM in most cases.

**Features:**

* No ORM Required
* Mature, High Performing Architecture
* DBA Approved
* Non-Opinionated
* Unit Of Work
* Function-based query construction
* Modular and Extensible
* Separate mapping and class design
* Eager-loading and caching of related objects and collections
* Composite (multiple-column) primary keys
* Self-referential Object Mappings
* Inheritance Mapping
* Raw SQL statement mapping
* Pre- and post-processing of data
* Supported Platforms
* Supported Databases

**Install using pip:**

pip install SQLAlchemy

**Community/Support:**

1. Gitter Room
2. IRC Channel-Freenode(<https://freenode.net/>)

* **SQLObject**

**License: GNU Library or Lesser General Public License (LGPL) (LGPL)**

* SQLObject is a popular *Object Relational Manager* for providing an object interface to your database, with tables as classes, rows as instances, and columns as attributes.
* SQLObject includes a Python-object-based query language that makes SQL more abstract, and provides substantial database independence for applications.
* Supports MySQL, PostgreSQL, SQLite, Firebird, Sybase, MSSQL and MaxDB (SAPDB). Python 2.7 or 3.4+ is required.
* **SQLObject** is a [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) [object-relational mapper](https://en.wikipedia.org/wiki/Object-relational_mapping) between a [SQL](https://en.wikipedia.org/wiki/SQL) [database](https://en.wikipedia.org/wiki/Database) and Python objects.
* It is experiencing community popularity, and forms a part of many applications (e.g., [TurboGears](https://en.wikipedia.org/wiki/TurboGears)).
* It is very similar to [Ruby on Rails](https://en.wikipedia.org/wiki/Ruby_on_Rails)' [ActiveRecord](https://en.wikipedia.org/wiki/ActiveRecord_(Rails)) in operation in that it uses class definitions to form table schemas, and utilizes the language's reflection and dynamism to be useful.
* SQLObject supports a number of common database backend’s: included in the distribution are [MySQL](https://en.wikipedia.org/wiki/MySQL), [PostgreSQL](https://en.wikipedia.org/wiki/PostgreSQL), [SQLite](https://en.wikipedia.org/wiki/SQLite), [Sybase SQL Server](https://en.wikipedia.org/wiki/Sybase_SQL_Server), [MaxDB](https://en.wikipedia.org/wiki/MaxDB), [Microsoft SQL Server](https://en.wikipedia.org/wiki/Microsoft_SQL_Server) and [Firebird](https://en.wikipedia.org/wiki/Firebird_(database_server)).

**Install using pip:**

pip install SQLObject

**Community/Support:**

1. [SQLObject-discuss mailing list](SQLObject-discuss%20mailing%20list)(<https://sourceforge.net/projects/sqlobject/lists/sqlobject-discuss>)
2. Stack overflow
3. [GitHub issue tracker](https://github.com/sqlobject/sqlobject/issues)(<https://github.com/sqlobject/sqlobject/issues>)
4. Sourceforge (<https://sourceforge.net/p/sqlobject/bugs/)>

* **Storm**

**License:** **GNU Library or Lesser General Public License (LGPL) (LGPL)**

* Storm is a [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) programming [library](https://en.wikipedia.org/wiki/Library_(computing)) for [object-relational mapping](https://en.wikipedia.org/wiki/Object-relational_mapping) between one or more [SQL](https://en.wikipedia.org/wiki/SQL) [databases](https://en.wikipedia.org/wiki/Databases) and Python objects.
* It allows Python developers to formulate complex queries spanning multiple database tables to support dynamic storage and retrieval of object information.
* [MySQL](https://en.wikipedia.org/wiki/MySQL), [PostgreSQL](https://en.wikipedia.org/wiki/PostgreSQL) and [SQLite](https://en.wikipedia.org/wiki/SQLite) [database](https://en.wikipedia.org/wiki/Databases) support is built into Storm, and the [API](https://en.wikipedia.org/wiki/API) allows for support for others. Storm also supports the [Django](https://en.wikipedia.org/wiki/Django_(web_framework)) and [Zope](https://en.wikipedia.org/wiki/Zope) [web frameworks](https://en.wikipedia.org/wiki/Web_framework) natively.

**Features:**

* Storm is fast.
* Storm lets you efficiently access and update large datasets by allowing you to formulate complex queries spanning multiple tables using Python.
* Storm handles composed primary keys with ease (no need for surrogate keys).
* Storm doesn’t do schema management, and as a result you’re free to manage the schema as wanted, and creating classes that work with Storm is clean and simple.
* Storm works very well connecting to several databases and using the same Python types (or different ones) with all of them.
* Storm can handle obj.attr = <A SQL expression> assignments, when that’s really needed (the expression is executed at INSERT/UPDATE time).
* Storm handles relationships between objects even before they were added to a database.
* Storm works well with existing database schemas.
* Storm will flush changes to the database automatically when needed, so that queries made affect recently modified objects.

**Install using pip:** pip install storm

**Community/Support:**

1. Storm mailing list(<https://lists.canonical.com/mailman/listinfo/storm>)
2. Freenode(<https://lists.canonical.com/mailman/listinfo/storm>)

* **Tryton**

**License:** **GNU General Public License (GPL) (GPL-3)**

**Tryton** is a three-tier high-level general purpose computer application platform on top of which is built an Enterprise resource planning (ERP) business solution through a set of **Tryton** modules.

Tryton is written in Python and using PostgreSQL as a database engine. This is the heart of a complete solution for business that provides modularity, scalability and security.

**Technical Features:**

The kernel provides the technical foundations needed by most business applications. However it is not linked to any particular functional field hence constituting a general purpose framework:

* Data persistence: ensured by accessor objects called [Models](http://doc.tryton.org/1.2/trytond/doc/models.html), they allow easy creation, migration and access to records.
* User Management: the kernel comes with the base features of user management: user groups, access rules by models and records, etc.
* [Workflow Engine](https://en.wikipedia.org/wiki/Workflow_application): allows users to activate a workflow on any business model.
* Report Engine: the report engine is based on [relatorio](https://pypi.python.org/pypi/relatorio) that uses [ODT](https://en.wikipedia.org/wiki/OpenDocument) files as templates and generate ODT or [PDF](https://en.wikipedia.org/wiki/PDF) reports.
* Internationalisation: Tryton is available in English, French, German, Spanish, Catalan, and Italian. New translations can be added directly from the client interface.
* Historical data: data historization may be enabled on any business model allowing for example to get the list of all the past value of the cost price of any product. It also allows users to dynamically access historized record at any time in the past: for instance the customer information on each open invoice will be the ones of the day the invoice was opened.
* Support for [XML-RPC](https://en.wikipedia.org/wiki/XML-RPC) and [JSON-RPC](https://en.wikipedia.org/wiki/JSON-RPC) protocols.
* Database independence is allowed thanks to the [python-sql](https://pypi.python.org/pypi/python-sql) library and is used for testing with the [SQLite](https://en.wikipedia.org/wiki/SQLite) backend.
* Built-in automatic migration mechanism: it allows updating the underlying database scheme without any human manipulation. Migration is ensured from series to series (releases inside the same series do not require migration). This automation is possible because the migration process is taken into account and tested continually within the development.
* Advanced modularity: the modularity allows a layered approach of the business concepts along with flexibility, which speeds up custom developments.

Being a framework, Tryton can be used as a platform for the development of various other solutions than just business ERPs. A very prominent example is [GNU Health](https://en.wikipedia.org/wiki/GNU_Health), a [free](https://en.wikipedia.org/wiki/Free_Software) Health and Hospital Information System based on Tryton.

**Install using pip:**

Pip install tryton

**Community/Supports:**

1. Tryton(<https://discuss.tryton.org/c/support/17>)
2. Tryton Forum(<https://www.tryton.org/forum>)
3. GitHub(<https://github.com/tryton>)

* **Pony**

**License: Apache Software License (Apache License Version 2.0)**

* **Pony ORM** is a library for Python language that allows you to conveniently work with objects that are stored as rows in a relational database.
* Pony ORM enables people to work comfortably with objects kept as tuples in a relational database system.
* It enables you to deal with the information of the databank, in the form of substances/objects.
* In the database, there are tables having tuples of data. Conversely, when it is possible to view the data obtained from the databank in object form, it is far more useful when writing the code in an advanced-level object-oriented semantic.

**Features:**

* An exceptionally convenient syntax for writing queries
* Automatic query optimization
* An elegant solution for the N+1 problem
* The [online database schema editor](https://editor.ponyorm.com)
* The IdentityMap pattern
* Automatic transaction management
* Automatic caching of queries and objects
* Full support of composite keys
* The ability to easily write queries using LEFT JOIN, HAVING and other features of SQL

**Install using pip:**

pip install pony

**Community/Support:**

1. Stack overflow(<https://stackoverflow.com/questions/tagged/ponyorm>)
2. Telegram(<https://telegram.me/ponyorm>)

* **Peewee**

**License:** **MIT License (MIT License)**

* [Peewee](http://docs.peewee-orm.com/en/latest/) ([source code](https://github.com/coleifer/peewee)) is a [object-relational mapper (ORM)](https://www.fullstackpython.com/object-relational-mappers-orms.html) implementation for bridging data stored in [relational database tables](https://www.fullstackpython.com/databases.html) with Python objects.
* Peewee provides a lightweight, expressive Python API for interacting with relational databases. Peewee follows the active record pattern used by a number of other object-relational mappers. Peewee supports PostgreSQL, MySQL and SQLite and has many database-specific extensions included in the.

**Features:**

* Creating model tables.
* Model options and table metadata.
* Indexes and Constraints.
* Primary Keys, Composite Keys and other Tricks.
* Self-referential foreign keys.
* Circular foreign key dependencies.

**Install using pip:**

Pip install peewee

**Community/Support:**

1. Python Community(<https://www.python.org/community/>)

* **Tortoise**

**License:** **MIT License**

* **Tortoise ORM** is an easy-to-use asyncio **ORM** (Object Relational Mapper) inspired by Django.
* **Tortoise ORM** was built with relations in mind and admiration for the excellent and popular Django **ORM**.
* It's engraved in its design that you are working not with just tables, you work with relational data.

**Features:**

* Designed to be used in an existing project:
  + Testing framework uses existing Python Unittest framework, just requires that initializer () and finalizer () gets called to set up and tear down the test databases. (See [UnitTest support](https://tortoise-orm.readthedocs.io/en/latest/contrib/unittest.html#unittest))
  + ORM [Init app](https://tortoise-orm.readthedocs.io/en/latest/setup.html#init-app) configures entirely from provided parameters
* Composable, Django-inspired [Models](https://tortoise-orm.readthedocs.io/en/latest/models.html#models)
* Supports relations, such as ForeignKeyField and ManyToManyField
* Supports many standard [Fields](https://tortoise-orm.readthedocs.io/en/latest/fields.html#fields)
* Comprehensive [Query API](https://tortoise-orm.readthedocs.io/en/latest/query.html#query-api)
* Transactions
* [PyLint plugin](https://tortoise-orm.readthedocs.io/en/latest/contrib/linters.html#pylint)

**Install using pip:**

Pip install tortoise

**Community/Support:**

1. Gitter Chat(<https://gitter.im/tortoise/community>)
2. GitHub(<https://github.com/tortoise/tortoise-orm/issues>)