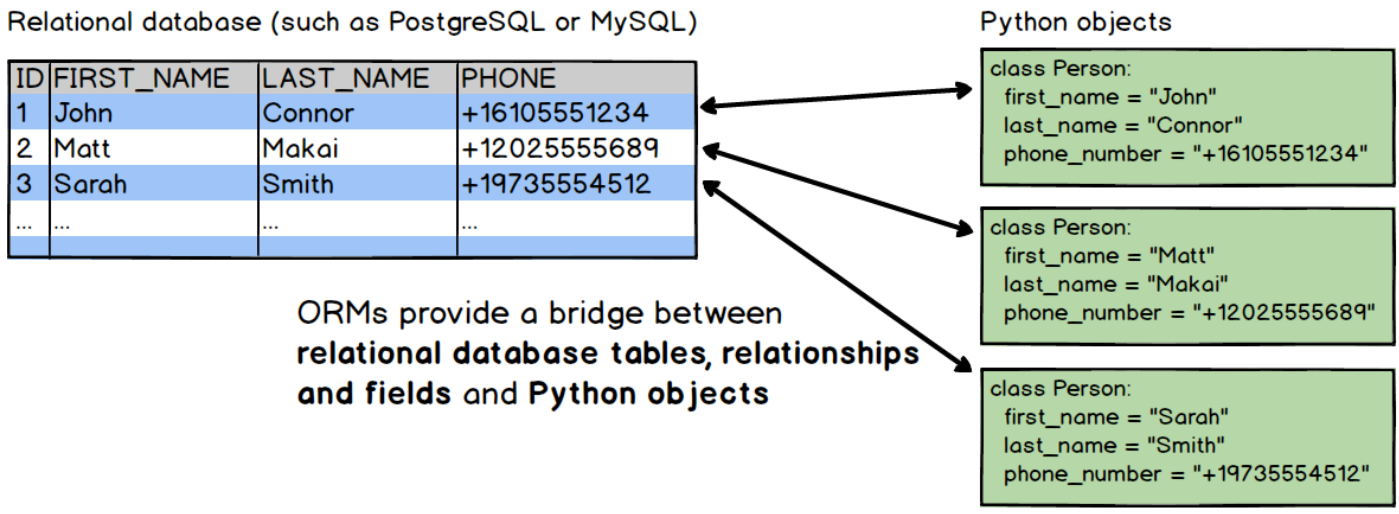
Full Stack Python

Object-relational Mappers (ORMs)

An object-relational mapper (ORM) is a code library that automates the transfer of data stored in relational database tables into objects that are more commonly used in application code.



Why are ORMs useful?

ORMs provide a high-level abstraction upon a relational database that allows a developer to write Python code instead of SQL to create, read, update and delete data and schemas in their database. Developers can use the programming language they are comfortable with to work with a database instead of writing SQL statements or stored procedures.

For example, without an ORM a developer would write the following SQL statement to retrieve every row in the USERS table where the `zip_code` column is 94107:

```
SELECT * FROM USERS WHERE zip_code=94107;
```

The equivalent Django ORM query would instead look like the following Python code:

```
# obtain everyone in the 94107 zip code and assign to users variable
users = Users.objects.filter(zip_code=94107)
```

The ability to write Python code instead of SQL can speed up web application development, especially at the beginning of a project. The potential development speed

boost comes from not having to switch from Python code into writing declarative paradigm SQL statements. While some software developers may not mind switching back and forth between languages, it's typically easier to knock out a prototype or start a web application using a single programming language.

ORMs also make it theoretically possible to switch an application between various relational databases. For example, a developer could use SQLite for local development and MySQL in production. A production application could be switched from MySQL to PostgreSQL with minimal code modifications.

In practice however, it's best to use the same database for local development as is used in production. Otherwise unexpected errors could hit in production that were not seen in a local development environment. Also, it's rare that a project would switch from one database in production to another one unless there was a pressing reason.

> While you're learning about ORMs you should also read up on deployment and check out the application dependencies page.

Do I have to use an ORM for my web application?

Python ORM libraries are not required for accessing relational databases. In fact, the low-level access is typically provided by another library called a *database connector*, such as psycopg (for PostgreSQL) or MySQL-python (for MySQL). Take a look at the table below which shows how ORMs can work with different web frameworks and connectors and relational databases.

web framework	None	Flask	Flask	Django
ORM	SQLAlchemy	SQLAlchemy	SQLAlchemy	Django ORM
database connector	(built into Python stdlib)	MySQL-python	psycopg	psycopg
relational database	SQLite	MySQL	PostgreSQL	PostgreSQL

The above table shows for example that SQLAlchemy can work with varying web frameworks and database connectors. Developers can also use ORMs without a web framework, such as when creating a data analysis tool or a batch script without a user interface.

What are the downsides of using an ORM?

There are numerous downsides of ORMs, including

1. Impedance mismatch
2. Potential for reduced performance
3. Shifting complexity from the database into the application code

Impedance mismatch

The phrase "impedance mismatch" is commonly used in conjunction with ORMs. Impedance mismatch is a catch-all term for the difficulties that occur when moving data between relational tables and application objects. The gist is that the way a developer uses objects is different from how data is stored and joined in relational tables.

This article on ORM impedance mismatch does a solid job of explaing what the concept is at a high level and provides diagrams to visualize why the problem occurs.

Potential for reduced performance

One of the concerns that's associated with any higher-level abstraction or framework is potential for reduced performance. With ORMs, the performance hit comes from the translation of application code into a corresponding SQL statement which may not be tuned properly.

ORMs are also often easy to try but difficult to master. For example, a beginner using Django might not know about the `select_related()` function and how it can improve some queries' foreign key relationship performance. There are dozens of performance tips and tricks for every ORM. It's possible that investing time in learning those quirks may be better spent just learning SQL and how to write stored procedures.

There's a lot of hand-waving "may or may not" and "potential for" in this section. In large projects ORMs are good enough for roughly 80-90% of use cases but in 10-20% of a project's database interactions there can be major performance improvements by having a knowledgeable database administrator write tuned SQL statements to replace the ORM's generated SQL code.

Shifting complexity from the database into the app code

The code for working with an application's data has to live somewhere. Before ORMs were common, database stored procedures were used to encapsulate the database logic. With an ORM, the data manipulation code instead lives within the application's Python codebase. The addition of data handling logic in the codebase generally isn't an issue with a sound application design, but it does increase the total amount of Python code instead of splitting code between the application and the database stored procedures.

Python ORM Implementations

There are numerous ORM implementations written in Python, including

1. SQLAlchemy
2. Peewee
3. The Django ORM
4. PonyORM
5. SQLObject
6. Tortoise ORM (source code)

There are other ORMs, such as Canonical's Storm, but most of them do not appear to currently be under active development. Learn more about the major active ORMs below.

Django's ORM

The Django web framework comes with its own built-in object-relational mapping module, generally referred to as "the Django ORM" or "Django's ORM".

Django's ORM works well for simple and medium-complexity database operations. However, there are often complaints that the ORM makes complex queries much more complicated than writing straight SQL or using SQLAlchemy.

It is technically possible to drop down to SQL but it ties the queries to a specific database implementation. The ORM is coupled closely with Django so replacing the default ORM with SQLAlchemy is currently a hack workaround. Note though it is possible that swappable ORM backends will be possible in the future as it is now possible to change the template engine for rendering output in Django.

Since the majority of Django projects are tied to the default ORM, it is best to read up on advanced use cases and tools for doing your best work within the existing framework.

SQLAlchemy ORM

SQLAlchemy 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. There is an entire page on SQLAlchemy that you should read if you want to learn more about using the library.

Peewee ORM

Peewee is a Python ORM implementation that is written to be "simpler, smaller and more hackable" than SQLAlchemy. Read the full Peewee page for more information on the Python ORM implementation.

Pony

Pony ORM is another Python ORM available as open source, under the Apache 2.0 license.

SQLObject ORM

SQLObject is an ORM that has been under active open source development for over 14 years, since before 2003.

Schema migrations

Schema migrations, for example when you need to add a new column to an existing table in your database, are not technically part of ORMs. However, since ORMs typically lead to a hands-off approach to the database (at the developers peril in many cases), libraries to perform schema migrations often go hand-in-hand with Python ORM usage on web application projects.

Database schema migrations are a complex topic and deserve their own page. For now, we'll lump schema migration resources under ORM links below.

General ORM resources

- This detailed overview of ORMs is a generic description of how ORMs work and how to use them.
- This example GitHub project implements the same Flask application with several different ORMs: SQLAlchemy, Peewee, MongoEngine, stdnet and PonyORM.
- Martin Fowler addresses the ORM hate in an essay about how ORMs are often misused but that they do provide benefits to developers.

- The Rise and Fall of Object Relational Mapping is a talk on the history of ORMs that doesn't shy away from some controversy. Overall I found the critique of conceptual ideas worth the time it took to read the presentation slides and companion text.
- If you're confused about the difference between a connector, such as MySQL-python and an ORM like SQLAlchemy, read this StackOverflow answer on the topic.
- What ORMs have taught me: just learn SQL is another angle in the ORM versus embedded SQL / stored procedures debate. The author's conclusion is that while working with ORMs such as SQLAlchemy and Hibernate (a Java-based ORM) can save time up front there are issues as a project evolves such as partial objects and schema redundancies. I think the author makes some valid points that some ORMs can be a shaky foundation for extremely complicated database-backed applications. However, I disagree with the overriding conclusion to eschew ORMs in favor of stored procedures. Stored procedures have their own issues and there are no perfect solutions, but I personally prefer using an ORM at the start of almost every project even if it later needs to be replaced with direct SQL queries.
- The Vietnam of Computer Science provides the perspective from Ted Neward, the originator of the phrase "Object/relational mapping is the Vietnam of Computer Science" that he first spoke about in 2004. The gist of the argument against ORMs is captured in Ted's quote that an ORM "represents a quagmire which starts well, gets more complicated as time passes, and before long entraps its users in a commitment that has no clear demarcation point, no clear win conditions, and no clear exit strategy." There are follow up posts on Coding Horror and another one from Ted entitled thoughts on Vietnam commentary.
- Turning the Tables: How to Get Along with your Object-Relational Mapper coins the funny but insightful phrase "database denial" to describe how some ORMs provide a usage model that can cause more issues than they solve over straight SQL queries. The post then goes into much more detail about the problems that can arise and how to mitigate or avoid them.

SQLAlchemy and Peewee resources

A comprehensive list of SQLAlchemy and Peewee ORM resources can be found on their respective pages.

Django ORM links

A curated list of resources can be found on the dedicated Django ORM resources page.

Pony ORM resources

All Pony ORM resources are listed on the dedicated Pony ORM page.

SQLObject resources

SQLObject has been around for a long time as an open source project but unfortunately there are not that many tutorials for it. The following talks and posts will get you started. If you take an interest in the project and write additional resources, file an issue ticket so we can get them added to this list.

- This post on Object-Relational Mapping with SQLObject explains the concept behind ORMs and shows the Python code for how they can be used.
- Ian Bicking presented on SQLObject back in 2004 with a talk on SQLObject and Database Programming in Python.
- Connecting databases to Python with SQLObject is an older post but still relevant with getting started basics.

What would you like to learn about building Python web apps?

Tell me about standard relational databases.

What're these NoSQL data stores hipster developers keep talking about?

I want to know about working with data in Python.