Object-Relational Mapping

Reading / References

- SQLAlchemy ORM Tutorial[1]

Notes

ORM Philosophy

In most applications we want to work and think with objects and the relationships between them. So for instance we might have *student objects* and *course objects* and also *enrollment objects* which contain in them a student object and a course object and maybe some more information about them. We may want to change the information of an enrollment for a student, for example when they withdraw from a class. We might want to check that each student is enrolled in a specific number of courses, and never exceed 4.5 credits, etc.

This is all what we might call the **business logic** of our application. This is the level at which we want to think about our application, and we want to write code that talks in these terms, so that it is easy for someone to read that code and understand the key business decisions. We often also refer to this understanding of our main objects as the **domain model**.

By contrast, at some point we need to *Persist* our data to a database. Databases typically don't talk in terms of "objects" and "relationships". They have tables, columns, primary keys, tuples, foreign keys and a host of other limitations. If possible, we would rather not have to program in those terms.

This is where Object-Relational Mapping enters. After some initial setup, ORM takes care of the nitty gritty details of converting between "business logic" and "database queries". For example it converts the statement "add this student to the system" to an appropriate INSERT query that would actually carry out the creation.

This comes at a price: There are often complicated situations that are somewhat slow in an ORM setting but would be faster if we were to write the SQL query directly. The good news is that most ORM systems live in harmony with more "core" systems, so you can write the majority of your application in ORM form, only occasionally resorting to more direct queries.

In an ORM setting you typically *declare* a **mapping** between a certain class in your application and a certain table. You then leave many of the details up to the system to handle. ### ORM In SQL Alchemy

[1] https://docs.sqlalchemy.org/en/latest/orm/tutorial.html