Theory: Object-Relational Mapping(ORM)

12 minutes 0 / 4 problems solved

A programming language is a tool to process data from one state to another. If we want to save the state of data permanently, we need some storage. Usually, for this purpose, we use a relational database, but then a problem occurs: programming languages and databases work with data differently. Moreover, relational databases have their language named **SQL** (Structured Query Language).

Luckily we are not the first people on Earth to suffer from this trouble. There is an approach called **Object-Relational Mapping** or **ORM** that is able to "translate" the database-way of interacting with information in an object-oriented way or to translate it backward.

Current topic:

Object-Relational Mapping(ORM)

Topic depends on:

✕ Data and object mapping

✕ What is SQL

Topic is required for:

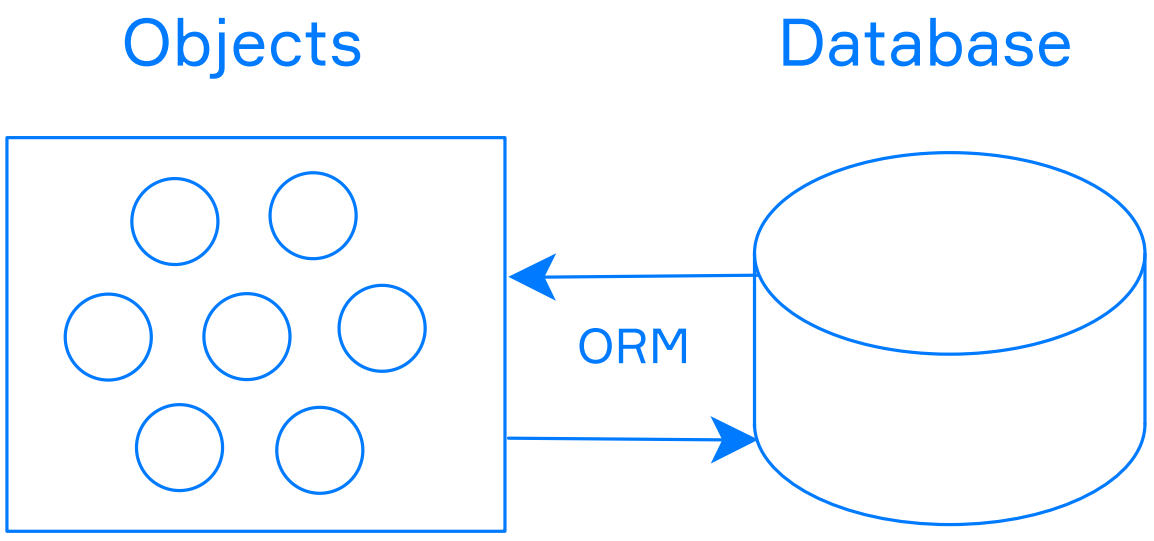
Introduction to JPA

Django ORM

§1. ORM concept

Almost all programming languages have ORM libraries, but before you start using them, let's find out what ORM is all about.

Object-Relational Mapping is a concept of converting data from an object-oriented programming language to relational database representation and vice versa. It solves the problem of matching two different type systems together and synchronize the data flow between them.



The main parts of ORM are:

- Virtual tables
- Relations
- Operations with objects

So what are these parts, and why do we need them?

Table of contents:

↑ Object-Relational Mapping(ORM)

§1. ORM concept

§2. Virtual tables

§3. Relations

§4. Operations on objects

§5. Pros and Cons

Feedback & Comments

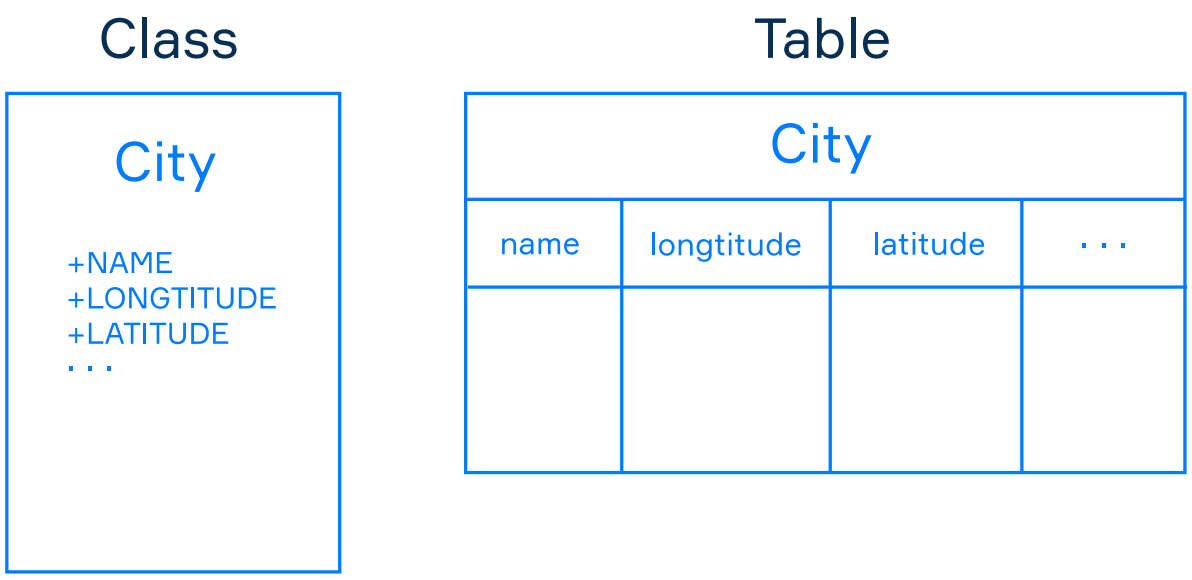
§2. Virtual tables

Relational databases use tuples and tables to store data. Most of the programming languages have tuples but don't have tables. How can we represent them in a programming language?

The main idea is to use classes as tables descriptions. We create a class as a virtual table for the given table in the database. We use or define methods for this class to retrieve, change and delete data

To represent one row from the table, we can define another class (for some libraries it can be the same class), and match its attributes to columns of the table. The instance of this class can manipulate not only the values of the row but also relations this row has, we will discuss it below.

Let's look at the example with the class `City`. We match values in tables with scalar attributes in the class. The name can be a string. Longitude and latitude can be real numbers.

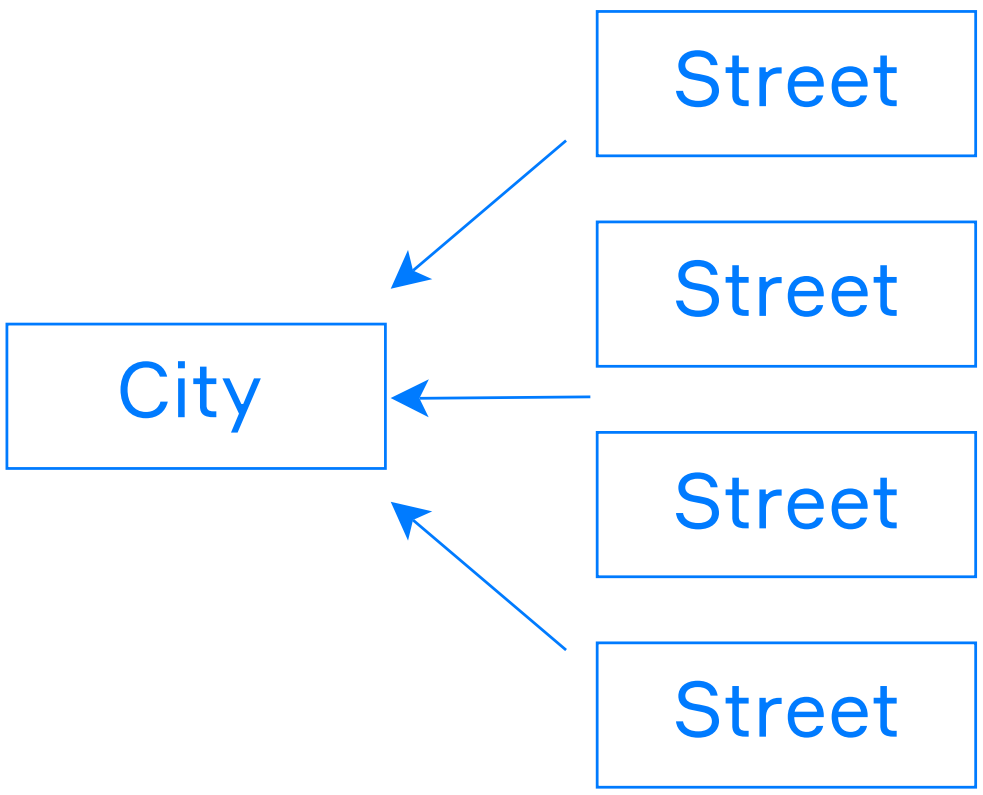


As you know, a class can have attributes that represent a list of other objects. For example, a city may have a lot of streets. It contradicts attribute-to-column mapping, but the ORM provides instruments for these cases too. In terms of relational databases, the link between one object and several others is called one-to-many relation. It's quite similar to a list attribute of an object.

§3. Relations

The relation is a link that connects a value from one table to the row in another. The database can store such links as keys. You can think about it as an object containing another object as an attribute.

Relations in databases are more than simple links. When you delete the root row from one table, it can imply cascade deletions of all related rows from other tables. It's not the same logic as for a programming language. If you have a link in an object, you expect that when you delete this link, only the link will disappear, not the connected object itself. On the other hand, if your database has a table "*City*" and London city in it, you can suppose when you delete London city from the table, all related rows from the table "*Street*" will disappear too. A street belongs to the city; without the city, there are no streets.



We can represent the example as the class `City` with a list of streets or as the many instances of class `Street` with an attribute `City`.

You may expect that if you delete the city, you will remove it with all the appropriate streets. However, if you delete the street and it has an attribute city, the city will stay in the database. It means that relations have directions. The cascade deletions imply the deletion of dependent objects, not dependent on ones.

§4. Operations on objects

Four common-used operations with rows in the database are known as CRUD (Create, Read, Update, Delete) operations. It's similar to what we can do with objects in the programming language.

After you've read about tables and relations, you can try to control them via ORM library. Libraries usually provide high-level commands that look like working with any other objects in the programming language. Knowing something about relations and tables helps you not to corrupt data in the database.

> It's highly recommended to read the documentation for the ORM library before using it. It helps you to get acquainted with the effects and consequences of applied operations.

§5. Pros and Cons

If you still do not know, should you use ORM for your project or not, you can consider some pros and cons that this concept has.

Pros:

- You work with a database the same manner you write another code
- The library isolates your code from the specificity of SQL language of the database you're using
- The code using ORM is usually easier to read, write and maintain
- You don't have to know SQL to work with a database

Cons:

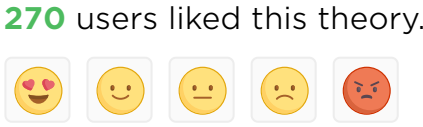
- Sometimes ORM libraries generate inefficient queries to a database
- It's hard to control the generation of queries
- You cannot use all the features and strengths of the specific database you have
- You should learn how to work with the library still

ORM is a good starting point for a project working with a database. Give it a chance and only then start interacting with a database directly. ORM will likely serve all your needs.

> To start working with an ORM library, you may consider SQLAlchemy for Python, Hibernate or JPA for Java/Kotlin, Sequalize or TypeORM for JavaScript.

Report a typo

270 users liked this theory. **6** didn't like it. **What about you?**



Start practicing

Comments (7) Hints (0) Useful links (2) Show discussion