UDC 004.65
Riabenko A.[1], Shyrokorad D.[1] , Boyar E.[2]
[1] assistant professor NU «Zaporizhzhia polytechnic»
[2] student gr. KNT-811 NU «Zaporizhzhia polytechnic»

AUTOMATING DATABASE INTERACTION USING ORM IN PYTHON

Python is one of the most popular programming languages in the world. It is widely used in web development, data analysis, machine learning, and many other fields. One of the key aspects of web development is working with databases. Although raw SQL queries provide full control, they can be difficult to implement and maintain, especially in large projects. That's why ORMs (Object-Relational Mapping) — libraries that allow developers to interact with databases through object-oriented programming — are becoming increasingly popular.

Today, Python is one of the leading languages for backend web development. Popular frameworks such as Django and Flask make extensive use of ORM solutions, which simplify interaction with databases.

ORM enables developers to interact with data through classes and objects rather than SQL queries. This improves security, reduces the amount of code required for database operations, and abstracts away from specific DBMSs (Database Management Systems). Thus, ORM provides:

Simplified data management;

Code portability between different databases;

Reduced errors related to SQL query syntax;

Better integration with object-oriented programming approaches.

The main goal of ORM is to automate database interaction, making it more flexible, secure, and efficient. To achieve this, ORM libraries offer a wide range of features:

Automatic creation of tables and database schemas. ORM allows defining data models as Python classes and then automatically creates corresponding tables in the database. This simplifies the process of developing and modifying the database structure without the need to write SQL code.

Performing CRUD (Create, Read, Update, Delete) operations. CRUD are the basic database operations. With ORM, it's easy to create, retrieve, update, and delete records in tables using Python objects without writing SQL queries. To create records in the database, ORM provides methods that allow adding new

entries using Python dictionaries. In our implementation, the insert_entries method of the BaseModel class allows adding multiple entries at once.

Migration support. Migrations are a vital component of modern ORMs because they allow automatic updates to the database structure in accordance with changes in the models. This is especially useful in large projects, where the database schema can change frequently.

Query optimization. One of the key benefits of ORM is the ability to optimize queries to the database. This helps reduce the number of SQL queries, improve application performance, and avoid unnecessary load on the database. Our ORM can implement several optimization mechanisms that are commonly used in modern ORM libraries.

Modern ORM libraries allow developers to add custom logic for query processing, extend functionality using plugins and middleware, and implement additional methods for specific tasks. Below are several aspects of extensibility and examples of their implementation:

Conclusion. Developing a custom ORM library can be a challenging yet rewarding endeavor. It provides a deeper understanding of how databases work, how queries can be optimized, and how object-oriented principles can be applied effectively. ORM gives programmers a powerful tool to interact with databases, simplifying development, increasing code security, and reducing the risk of errors.

Considering the growing popularity of Python and the need for effective data management, the use of ORM is becoming an essential aspect of modern development. Understanding the basic principles of ORM allows developers to create scalable, efficient, and maintainable applications.