ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 11, Issue 4, April 2024

Decoupling Design Patterns

Iskandarov I.Z , Gulomov A.R.

Urgench branch of Tashkent University of Information Technologies named after Muhammad Al-Khwarizmi
Urgench branch of Tashkent University of Information Technologies named after Muhammad Al-Khwarizmi

ABSTRACT: The article discusses decoupling patterns that are used as the basis for creating ORM (Object Relational Mapper) technology. Three decoupling patterns are considered: description, the problem that the pattern solves, the advantages and disadvantages of each pattern. Comparative characteristics and SWOT analysis are also shown.

KEY WORDS: design patterns, ORM, database

I.INTRODUCTION

Design pattern - an architectural solution to a task or problem that arises when designing software. Design patterns make it easier to reuse successful designs and architectures. Expressing proven techniques as design patterns makes them more accessible to developers of new systems [1]. Design patterns have a number of advantages: scalability, they are time-tested and experienced by software architects and developers, reduction in design time due to the use of ready-made "templates". Design patterns also form an informal standard and terms (names) for solving known problems. This makes it easier to analyze and study the system if you indicate the name of the pattern used in the design of a particular part of the system. The relevance of database design patterns is determined by the fact that in most cases the system needs to store and process data and thereby use the database. Database design patterns allow you to simplify working with databases, build abstractions between application layers, and define the application architecture when working with data. Typically, a design pattern description consists of the following parts:

1. *Name* – defines the meaning of the pattern and the standard name for further use. The names form a dictionary of patterns.
2. *Description of the problem* - indicates the problem or case in which the problem occurs and also defines the statement of the problem.
3. *Solution* – indicates an architectural solution to a problem, usually using diagrams.
4. *Result* – shows the advantages, disadvantages, limitations of using the design pattern.

II. LITERATURE ANALYSIS AND METHODOLOGY

One of the fundamental literature on this theme is the book "Patterns of Enterprise Application Architecture" by Martin Fowler [2]. The book covers the theme in the context of using patterns to create enterprise applications. This literature discusses the theme of the object model and relational databases, various problems of interaction of an application with databases, object-based data mapping (use of OOP), and creating layers between the application and the database. Several database patterns are presented, such as: Active Record, Data Mapper, Gateway, Identity Map and others.
Some literature approaches the theme based on specific programming languages. For example, in [3], in the chapter Database Patterns, there are given some patterns with a description, implementation in the programming language PHP and the consequences of using the pattern. The theme is discussed in detail in [4] given such patterns as: Data Accessor (Data Access Object), Active Domain Object (Active Record), Object/Relational Map and others. For each pattern, an implementation in the Java programming language is provided. Data caching patterns are also discussed.

ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 11, Issue 4, April 2024

A.Object model and relational mapping

Let us look at interacting with data in an application (data layer) in the context of the OOP paradigm and relational databases. Let's define an **object model** as a model in which database entities will represent classes (models) and entity data will represent objects. In the case of a relational database, one object will represent one record in the table, and the object's properties store the values of the record's attributes. This model can be called **object-relational**. Table 1 shows the relationship between the elements of the relational database and the object-relational model.

Table 1. Corresponding elements of the relational data model and the object-relational model

Relational data model	Informal name	Object-relational model
Relation	Table	Class (Model)
Tuple	Record, Row	Object
Attribute	Column	Property (of an object)
Selection[1]	Result table	Collection

Figure 1 shows an example of representing a table through a class.

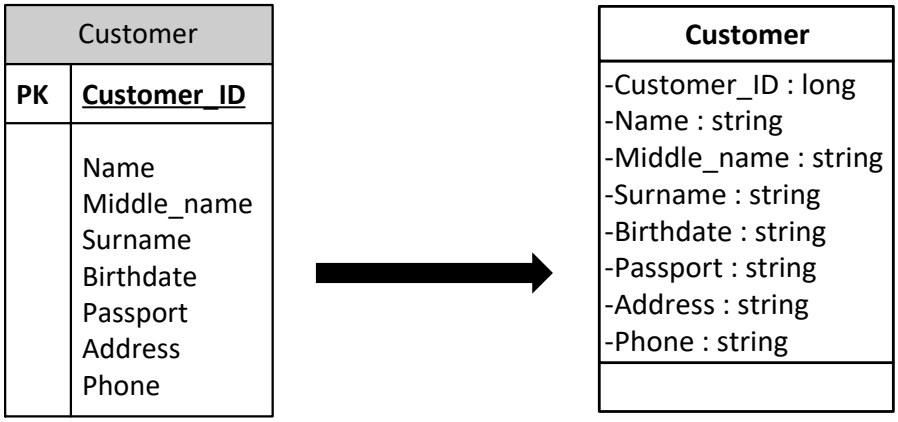

Fig. 1. Representing a database table through a class

Object-relational mapping (ORM) technology is built on the basis of this model. The task of ORM is to ensure work with the database in the application using OOP. Queries to the database are carried out through methods and work with data is carried out through objects. Thus, ORM creates an abstraction and encapsulates working with data and executing queries to the database.

Figure 2 shows the interaction diagram of ORM with the database

[1] This is a relational algebra operation that consists of selecting data

ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 11, Issue 4, April 2024

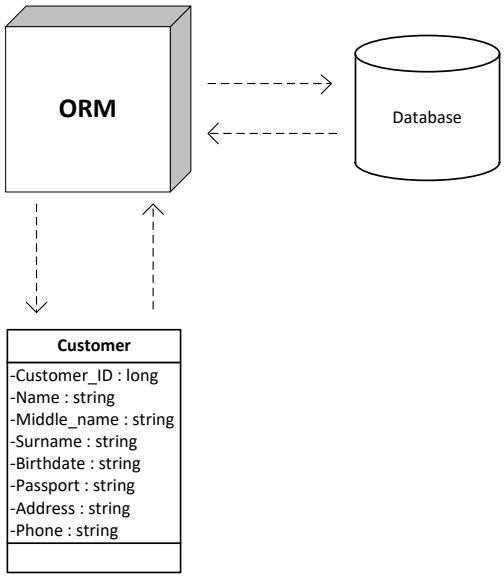

Fig. 2. Interaction between the model and the database

B.Classification of Database Patterns

Based on the problems that the pattern solves and the level of abstraction, database patterns can be divided into categories. In this article, we will define and consider some of them.

One of the main architectural problems when an application interacts with databases is to separate a part that works with data from other parts of the application, that is, create a *data layer* and also provide a way for this layer to interact with the business logic of the application. The most popular scheme is to divide the application into 3 layers: *presentation, business logic, data layer* (Figure 3).

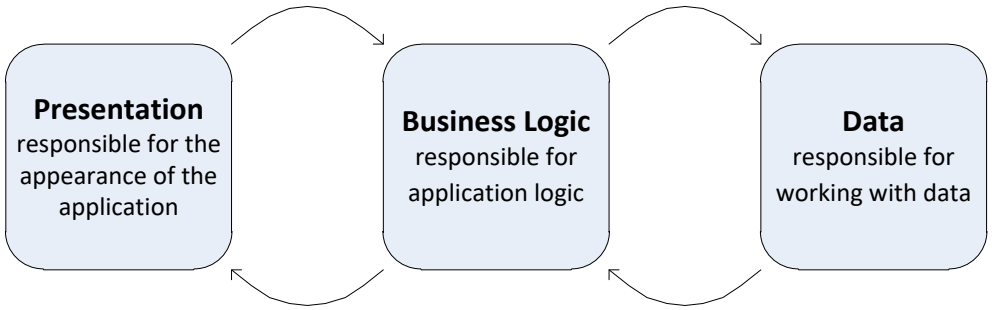

Fig 3. Dividing the application into layers

To solve this problem, we define a category of **decoupling patterns**. Decoupling patterns define how application code relates to its data model and data access code. As you decide on an application architecture, you need to consider how much cohesion you want between orthogonal components based on how much you expect them to vary independently. Decoupling components also makes it easier to build and maintain them concurrently [4]. Examples of decoupling patterns include the following patterns: Active Record, Data Access Object, etc.

The next category of patterns is **behavioral patterns.** Behavioral patterns address issues of object state, identification, and loading. This category includes such patterns as: Unit of Work, Identity Map, etc. **Data modeling patterns** prescribe the structure and method of organizing data for different contexts. This category focuses on solving architectural problems at the database level. This category of patterns includes: EAV, Hierarchy Pattern, etc. Also, other categories are indicated in different sources. For example, in [2] are given object-relational mapping patterns (*Metadata Mapping, Query Object*), object-relational structural patterns (*Identity Field, Foreign Key Mapping*) in [4] resource patterns (*Resource Decorator, Resource Pool*). Figure 4 shows a classification diagram based on the three patterns described above.

ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 11, Issue 4, April 2024

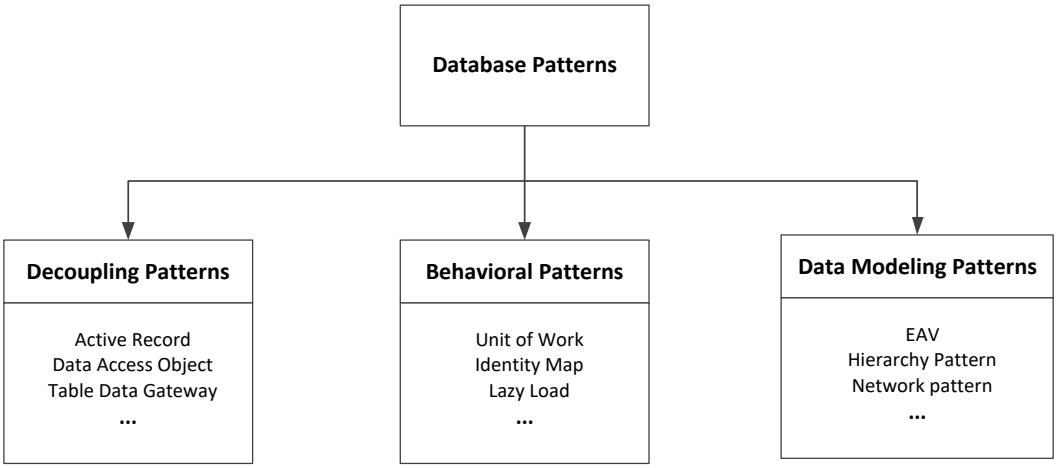

Fig 4. Incomplete classification of database patterns

*C.*Decoupling patterns

Active Record. In this pattern, an object represents the data of a single row of a database table as in the object model, encapsulates access to the database and also includes business logic. Business logic is included as object methods. Figure 5 shows an example of the Active Record pattern model.

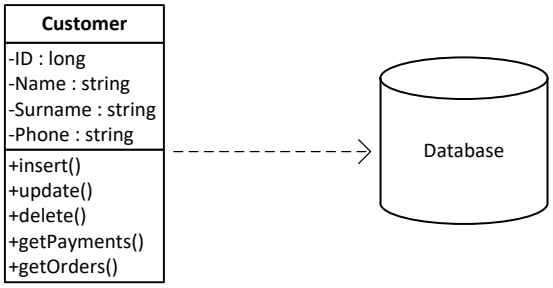

Fig 5. Model of the Active Record pattern

Problem. As noted above, if there is a need to use a database in an application, the most correct thing is to create a layer for working with data so that other parts of the application do not interact directly with the data (this increases the complexity of the application and the connectivity between components) and also add the ability to define business logic in the process interaction with this layer.
Solution. The Active Record pattern solves this problem by using an object model and incorporating business logic into the object. Based on this, the following are usually included in the Active Record:

- fields corresponding to the table schema, access and modification methods (getters/setters) for them.
- various data transformation methods (mutators).
- methods for inserting, deleting, and updating data.
- data retrieval methods.
- business logic operations associated with this model.

It should be noted that Active Record usually interacts with the interface of a physical database adapter that will execute SQL queries to the database.
Purpose. Active Record is a good choice for business logic that isn't too complex, such as creates, reads, updates, and deletes. Derivations and validations based on a single record work well in this structure. Active Record is used by many ORMs and can be said to be an implementation of ORM.
Results. Using this pattern gives the following results:

ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 11, Issue 4, April 2024

Benefits

- Easy to add business logic.
- Simplifies working with data. Creates an abstraction for working with data other parts of the application do not need to know implementation details and can access models.
- Groups related data access code into a single component.

Drawbacks

- Violates the Single Responsibility Principle (SRP). The model will contain both business logic and data processing.
- Active Record is following the Database-First approach. You create a database and then model it in the code. It means that entities do not contain logic.
- When changing the way data is stored, problems with code refactoring may arise.

Data Access Object. An object that encapsulates and abstracts access to data from a persistent store or an external system. The Data Access Object or DAO pattern is used to separate low level data accessing operations from the application or business logic layer. Usually it uses relational databases as the data source, but may use other storage mechanisms and source types.

Problem. We can access source APIs directly to work with data, but this creates a strict dependency on that source or storage mechanism. That is, when changing the source, you will have to make changes to the code base in those places where access to the API is used. Using the Active Record example, it was indicated that it interacts with the interface of a physical database adapter that will execute SQL queries to the database.

Solution. Use a Data Access Object (DAO) to abstract and encapsulate all access to the data source. The DAO manages the connection with the data source to obtain and store data. The DAO provides an interface that ensures the necessary methods for working with a data source. Since the interface does not change, we can change the source quite easily. The DAO creates an abstraction for working with a data source and hides the implementation details. The data source can be relational DBMS, external data storage, files, etc. Following are the participants in Data Access Object Pattern:

- DAO interface - This interface defines the standard operations to be performed on a model object(s).
- DAO concrete class - This class implements above interface. This class is responsible to get data from a data source which can be database / xml or any other storage mechanism.
- Transfer object – This object contains data retrieved from source which are stored in object fields and access methods for this fields (getters/setters). The Data Access Object may use a Transfer Object to return data to the client.
- DataSource - This represents a data source implementation. A data source could be a database such as an RDBMS, OODBMS, XML repository, flat file system, and so forth.
- BusinessObject - The BusinessObject represents the data client. It is the object that requires access to the data source to obtain and store data.

Figure 6 shows the class diagram representing the relationships for the DAO pattern.

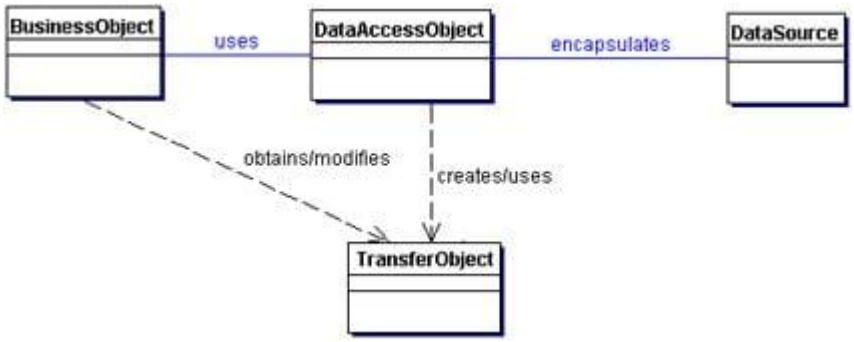

Fig. 6. The Data Access Object structure diagram

ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 11, Issue 4, April 2024

Figure 7 shows an example of the DAO class diagram.

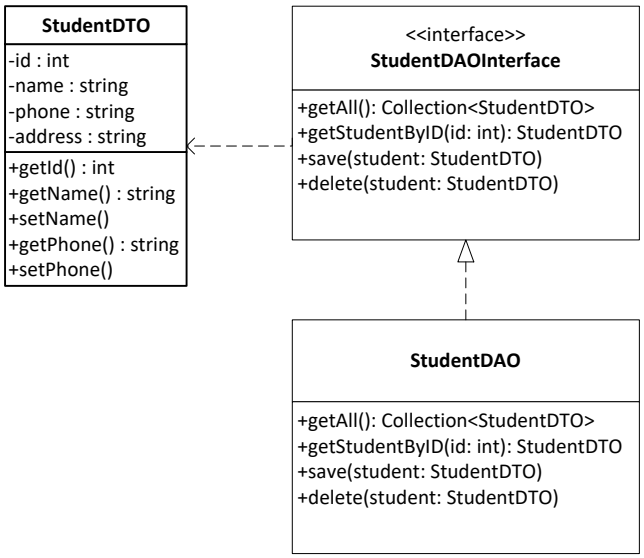

Fig. 7. Example of the DAO

Results. Using this pattern gives the following results:

Benefits

- Enables Transparency. Business objects can use the data source without knowing the specific details of the data source's implementation. Access is transparent because the implementation details are hidden inside the DAO.
- Enables Easier Migration. A layer of DAOs makes it easier for an application to migrate to a different database implementation.
- Reduces Code Complexity in Business Objects. Because the DAOs manage all the data access complexities, it simplifies the code in the business objects and other data clients that use the DAOs.

Drawbacks

- Adds Extra Layer. The DAOs create an additional layer of objects between the data client and the data source that need to be designed and implemented to leverage the benefits of this pattern. But the benefit realized by choosing this approach pays off for the additional effort.
- Needs Class Hierarchy Design. When using a factory strategy, the hierarchy of concrete factories and the hierarchy of concrete products produced by the factories need to be designed and implemented.

Data Mapper. A Data Mapper is a pattern that performs bidirectional transfer of data between a persistent data store (often a relational database) and an in-memory data representation.

Problem. The data we receive from a data source (usually a relational database) can be in different formats. Relational databases store data in the form of tables that consist of rows and columns. The format may differ for other storage systems. As with the object model, we need to map this data as an object and push changes back to the database if necessary. In addition, we may only need part of the data or a different storage structure.

Solution. The approach to solving this pattern is similar to the Data Access Object pattern. In practice, these two patterns provide a similar solution. A layer for working with data is created and a data transfer object is used/generated. But the Data Mapper pattern pays special attention to displaying data in an object and uses an additional EntityManager class.

ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 11, Issue 4, April 2024

Figure 8 shows the class diagram of the DataMapper pattern.

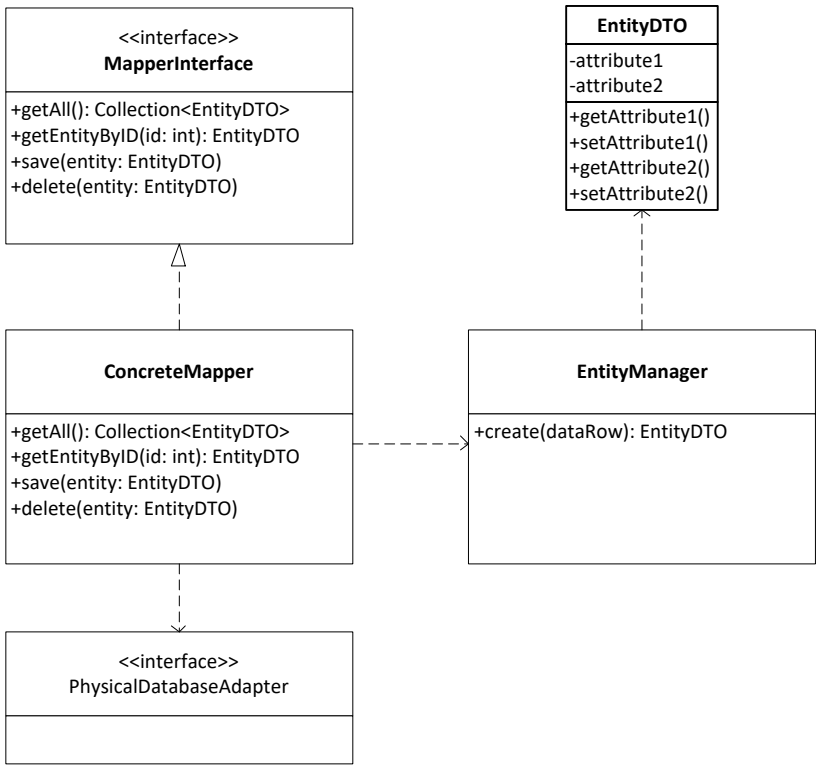

Fig. 8. The DataMapper structure diagram

Results. Using this pattern gives the following results:

Benefits

- Allows flexible customization of the creation of data objects that can come from different sources and in different formats.
- Separates the business logic of the application and the data layer. Separates other parts of the application from working with data and creates a single area of responsibility in the data layer.
- Good testability. The Data Mapper pattern promotes testability by decoupling the domain objects from the database. Since the domain objects are not tightly coupled to the database operations, you can easily write unit tests for the business logic without the need for an actual database connection.

Drawbacks

- Increased complexity. The Data Mapper adds an extra layer of abstraction, which can make the overall architecture more complex and harder to understand.
- Performance overhead. The data mapper must translate between the domain and the persistence layer, adding a performance overhead.

III. RESULTS AND DISCUSSION

Table 2 provides a comparison of the three patterns above. Numerical characteristics are given on a scale from 0 to 5.

ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 11, Issue 4, April 2024

Table 2. Comparison table of the three patterns

Pattern name	Antipattern	Which ORMs use the pattern	Popularity (0-5)	Complexity (0-5)	Related patterns	Flexibility (0-5)	Testability (0-5)
Active Record	Yes	Laravel, Yii2, Django, ActiveJDBC	5	2	Data Accessor, Object/Relational Map, Selection Factory	3	3
Data Access Object	No	Persist, Entity Framework, Hibernate	3	3	DTO, Singleton, Abstract Factory, Command, Table Data Gateway	5	4
Data Mapper	No	Doctrine2, SQLAlchemy, Cycle ORM, Hibernate	4	4	DTO, Domain Object Assembler, Table Data Gateway	5	5

Figure 9 shows a SWOT (Strengths, Weaknesses, Opportunities, Threats) analysis of the Active Record pattern.

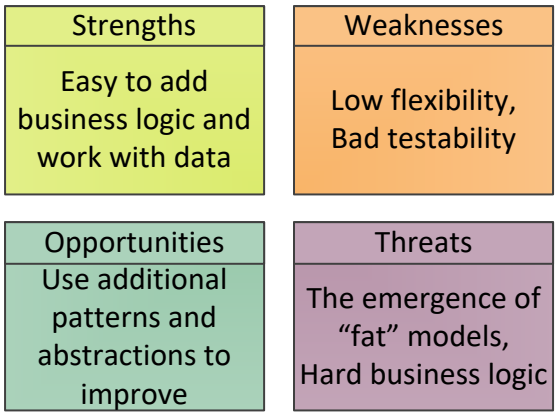

Fig. 9. SWOT analysis the Active Record

Figure 10 shows a SWOT analysis of the Data Access Object pattern.

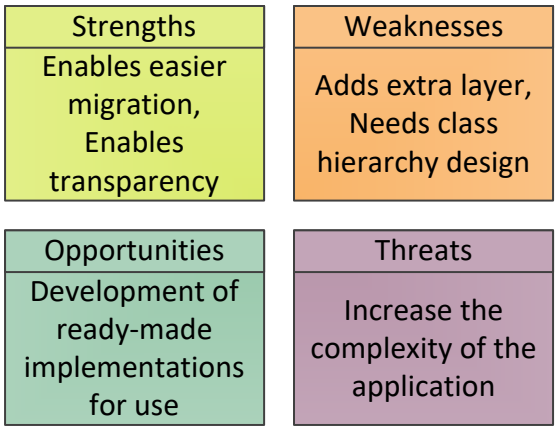

Fig. 10. SWOT analysis the Data Access Object

ISSN: 2350-0328
International Journal of AdvancedResearch in Science, Engineering and Technology
Vol. 11, Issue 4, April 2024

Figure 11 shows a SWOT analysis of the Data Access Object pattern.

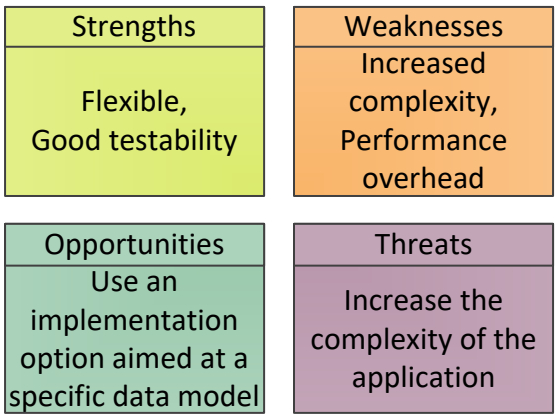

Fig. 11. SWOT analysis the Data Access Object

IV. CONCLUSION

The patterns considered are used by most ORMs and are the basis for their creation. However, decoupling patterns do not solve all the problems that arise when creating an ORM. Thus, it is necessary to use additional patterns, architectural solutions, algorithms, and models to build a full-fledged ORM. Which decoupling pattern to choose depends largely on the tasks being solved and the scale of the application. The Active Record pattern is the most popular and easy to use, and we can choose this pattern if the business logic of the application is not too complex. On the other hand, for large applications, it is recommended to use the Data Access Object and Data Mapper patterns. However, they increase the complexity of the application, thereby complicating development. In general, we can say that these patterns are almost equivalent and perform similar tasks.

REFERENCES

1. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, “Design Patterns: elements of reusable object-oriented software”, Addison-Wesley, 1995
2. Martin Fowler, David Rice, Matthew Foemmel, Edward Hieatt, Robert Mee, Randy Stafford, “Patterns of Enterprise Application Architecture”, Addison-Wesley, November 05, 2002.
3. Matt Zandstra, “PHP Objects, Patterns, and Practice, Second Edition”, Apress, 2008.
4. Clifton Nock, “Data Access Patterns: Database Interactions in Object-Oriented Applications”, Addison-Wesley, September 15, 2003.