S.V.Hrybkov, V.A. Struzik, A.O. Lytvyn, V.V. Chobanu
National university of food technologies

STUDY OF THE METHODS AND APPROACHES OF DATABASES REFACTORING

At different stages of the life cycle of any software the task to refactor its elements and components appears in order to improve the efficiency and quality of its operation, optimize the end product, and ensure compatibility with the approaches of other developers.

At the moment, the approaches to refactoring code are described in most fundamental literary sources, there are many approaches and automated software tools for its implementation. The process of code refactoring in theory does not differ from practical application [1].

But special attention should be paid to the refactoring of databases, aimed at increasing the efficiency of working with it, ensuring the preservation of its functional and information semantics. It should be noted that refactoring the database depends on the stage at which the software is located, because the closer it is to the implementation and operation phase, the more difficult and time consuming it will be. Database refactoring is much more complex than code refactoring, because [1-2]: the database includes structured data stored in accordance with a specific schema; Program constructs, such as stored procedures and triggers, that provide the execution of business logic. Refactoring of the saved program constructions in the database is carried out by standard approaches of code refactoring and is subject to automated testing. But any changes in the database concern not only the formats and storage structures of information in it, but affect the approaches and functionality of the software in general. A particular problem when refactoring a database is creating existing data in it.

The main reasons for database refactoring are the following drawbacks: [2] multipurpose columns used for several purposes and in various applications; multi-purpose tables used to store data about the essence of several different types; data redundancy is a significant problem in the daily operation of the database, since when storing the same data in several places there is a possibility of violation of compliance and consistency; tables with a large number of columns, can indicate that the fields can belong to different entities; multiple-valued columns containing several different pieces of information in different positions; multi-valued columns, within which several different pieces of information are represented in different positions.

When refactoring a database, new functionality should not be added or the existing ones should not be broken, or new data should not be added or the existing data should not be added, but changes may be made both in the database schema and in the business information processing modules, because they are tied to the scheme [1]. It should also be noted that sometimes it is necessary to conduct a sufficiently deep refactoring, which will concern very significant changes in order to optimize and improve, while maintaining the functionality of the software.

When refactoring databases it is expedient to distinguish 6 main categories of operations, they are distributed in the following areas of improvement: structures; data

quality; connections and integrity; architecture; methods; transformations, not included in the refactoring operations. It is advisable to consider each category separately.

Refactoring operations of the structure aimed at changing the structures of one or more tables or views. An example of such an operation can be moving a column from one table to another or splitting the multi-purpose column into several separate columns, each of which performs a separate assignment.

Refactoring operations aimed at data quality should first of all ensure the efficiency of storage and work with such data, achieved by adding validation rules, data entry formats, ensuring the inability to leave empty fields and use default values, etc.

Carrying out a refactoring operation aimed at communication and data integrity ensures the coherence of data in different tables connected by different types of links. In such operations, it is advisable to create triggers for: implementing a cascading communication type; keeping a history of the conducted operations; saving deleted or modified records in additional tables, etc.

The architecture refactoring operations are aimed at breaking down complex business logic operations implemented in client applications into simpler ones and implementing them in stored procedures and functions, allowing unifying their further use in various software modules and applications.

The operation of refactoring of methods consists in modifying stored procedures, functions and triggers aimed at optimizing and improving the quality of their execution. While these operations are performed, there are rules for code refactoring, but all the consequences of such changes must be taken into account. The simplest examples are changing the names of the surviving procedure to simplify understanding of its purpose, and a more complex example might be the replacement of the information processing algorithm.

The last group of transformation operations that are not part of the refactoring operations, aimed at changing its semantics, because they can be justified only if agreed with the customer software.

In order to perform successful database refactoring, the following simple approaches should be followed, which according to the authors' opinion, were identified in the study: create a complete copy of the existing software and its elements; use automated version control tools for software development; make any changes to the initial SQL code for creating the database; use automated testing tools.

It should be noted that the approaches considered for refactoring databases are applied when using relational and object-oriented database management systems. To simplify the process of refactoring at the stage of choosing DBMS, it is expedient to choose object-oriented DBMS, because they support the evolutionary approach for creating software, but relational DBMSs are guided by a consistent development approach.

References

1. Sadalage P. J., Ambler S. W. Refactoring databases: evolutionary database design [Text] (Addison-Wesley Signature Series) – Addison-Wesley Professional, 2006. – 384 p.

2. Fowler M., Sadalage P. J Evolutionary Database Design [Electronic resource] martinfowler.com, May 2016: Proceedings. – Mode of access: http://martinfowler.com/articles/evodb.html - Last access: 2017. – Title from the screen.

3. Fowler M. Refactoring [Electronic resource] Refactoring.com, May 2017: Proceedings. – Mode of access: https://www.refactoring.com/ - Last access: 2017. – Title from the screen.