UDC 004.65

Kopp A.M., *Ph.D., Associate Professor of the Department of Software Engineering and Management Information Technologies*
Orlovskyi D.L., *Ph.D., Associate Professor of the Department of Software Engineering and Management Information Technologies*

MAKING DATA STORAGES DECENTRALIZED: AN IDEA OF SMART CONTRACTS GENERATION FROM RELATIONAL DATABASE TABLES

National Technical University «Kharkiv Polytechnic Institute», Ukraine

Introduction. Nowadays blockchain is not only about Bitcoin or other cryptocurrencies, it is a widely-adopted technology that allows to implement and assure forgery resistance, immutability, and decentralized community-backed governance [1]. Blockchain works with transactions that are consolidated into blocks that contain hash values of blocks generated before, which creating a chain of irreversible and immutable blocks that allows data authenticity and consistency, which could be proven by checking conformity of hash values back to the initial block [2]. Thanks to such benefits, "smart contracts" were introduced as computer programs executed using so-called "programmable blockchains", such as Ethereum, Polkadot, Solana, EOS, and Binance Smart Chain [3].

Problem statement. Today in the corporate segment relational database management systems (RDBMS) still dominate. However, emerging trends of data governance assume decentralization in order to achieve timestamping and immutability advantages. Thus, a problem of centralized storages transformation into decentralized solutions becomes relevant and will be addressed in this paper.

Related work. As was already mentioned above, smart contracts are nothing more than computer programs developed using a programming language called Solidity [4]. But, in contrast to other "traditional" programming languages, programs written in a Solidity are stored and executed on the blockchain in order to achieve certain functionality [4]. Thanks to one or another blockchain platform (e.g. Ethereum or Binance Smart Chain) to which smart contracts are deployed, programs do not need any trusted authority to reach consensus, while transactions of smart contracts are always traceable and credible [4]. Moreover, there are other specific features of smart contracts: the storage of each contract instance is at a permanent address on the blockchain. In smart contracts, each instance is a particular execution context and its changes are possible through external calls only [4].

In general, Solidity is a high-level Turing-complete statically typed programming language, which syntax is the most similar to JavaScript. In Solidity contracts are similar to classes in object-oriented programming languages. Therefore, like classes, contracts could contain state variables and functions to read and change their state in the same way, as it could be done by methods in traditional object-oriented programming [5]. Already deployed smart contracts could be considered as instances of classes when compared to the object-oriented programming approach.

Research aims. This study aims at making data storages decentralized using programmable blockchain platforms such as Ethereum or Binance Smart Chain by applying the idea of Solidity smart contracts generation from relational database tables.

Therefore, the following tasks should be solved in this study:

- suggest a method and a corresponding algorithm for the generation of Solidity smart contracts source code from database tables already used in relational databases;
- develop the software prototype to implement the proposed method;
- perform experiments in order to validate the proposed method;
- discuss obtained results, make conclusions, and define research directions for future work.

Materials. Therefore, in order to represent data structures defined by database table columns, we can use the structure type also available in Solidity in order to represent data records stored on the blockchain. Sample mapping is demonstrated in Figure 1.

As it is shown in Figure 1 below, table columns should be translated into structure variables, while the string data type is chosen as default for all variables in order to achieve simplicity and, at the same time, better compatibility (Solidity data types are too specific and can be barely mapped to a variety of modern RDBMS data types).

```
CREATE TABLE [employee] (
    [employee_id] INTEGER NOT NULL,
    [first_name] NVARCHAR(20),
    [last_name] NVARCHAR(20),
    [position_id] INTEGER NOT NULL,
    [birth_date] DATETIME,
    [onboarding_date] DATETIME,
    [department_id] INTEGER NOT NULL,
    CONSTRAINT [employee$employee_id]
        PRIMARY KEY ([employee_id])
);
GO
```

```
contract employee_contract {
    struct employee {
        string employee_id;
        string first_name;
        string last_name;
        string position_id;
        string birth_date;
        string onboarding_date;
        string department_id;
    }
}
```

Fig. 1. Sample mapping between the database table and smart contract structure

In addition to the structure definition mentioned in Figure 1, the smart contract should contain an array of structure instances as the state variable of public scope. Whereas the public getter function for this variable will be generated by the Ethereum Virtual Machine automatically, there still should be provided two more functions: to add new data records that will be appended to the end of an array variable in order to store them on the blockchain; to count the number of records already stored on the blockchain (i.e. the length of the array of structure instances).

Hence, the sequence of steps required to generate the Solidity smart contract using metadata of the RDBMS database table could be shown in Figure 2 below.

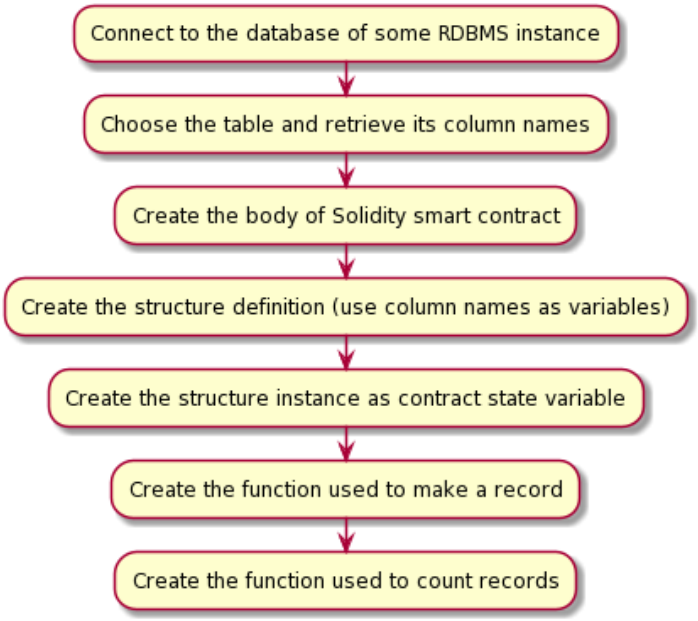

Fig. 2. Algorithm to generate Solidity smart contracts source code from database tables

The suggested algorithm was implemented using the Python programming language, whereas as the sample database table we used one mentioned in Figure 1 and being the part of the Microsoft SQL Server "staff" database. Source code of the Python program, database scripts, and obtained Solidity code are available in the experimental GitHub repository [6].

Results. Generated smart contract is available at [6] in the "staff.sol" file, as well as the Python code used to connect the Microsoft SQL Server instance and retrieve the "employee" table columns list. In order to validate generated smart contract, the following steps were made:

– it was deployed to the Ropsten network of Ethereum blockchain for testing purposes;
– five data records were added to the blockchain using the generated function (see Figure 3);
– automatically generated public get function was used to retrieve records (see Figure 3);
– generated function was used to count records stored on the blockchain (see Figure 3).

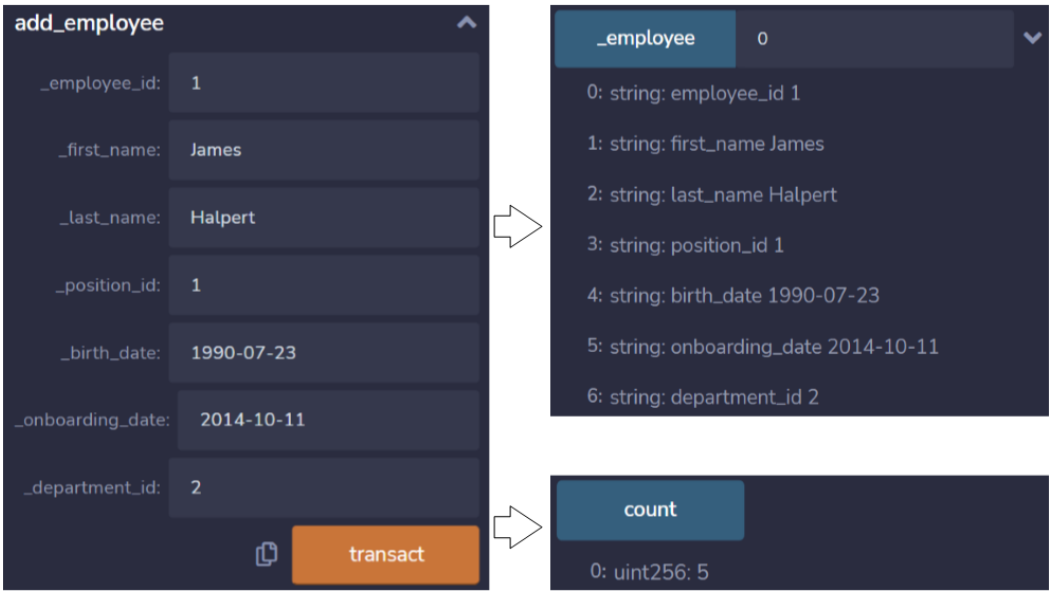

Fig. 3. Using smart contract functions to create, retrieve, and count records

Generate smart contract was verified, so its read and write functions are available at [7].

Conclusion and future work. This paper has presented an idea of a "smooth" transition from centralized data storage systems to the blockchain-based decentralized traceable and immutable data storages, suitable for industries where data security and integrity properties are crucial, such as supply chains and logistics, insurance, and real estate, social security and personal data processing, voting and governance, healthcare and pharmaceutics. Future research directions include improvement of the proposed method, as well as the elimination of current limitations related to the usage of string data type for all variables by default, restricted set of functions generated automatically, unsupported data integrity and consistency assurance tools, as well as software tool elaboration for usage with other RDBMS and possibly NoSQL systems.

References.

1. Macdonald M., Liu-Thorrold L., Julien R. The blockchain: a comparison of platforms and their uses beyond bitcoin // Work. Pap. – 2017. – P. 1–18.

2. Sato M. Fundamentals of Blockchains // Blockchain Gaps. – Springer, Singapore, 2021. – P. 1–8.

3. The Best Smart Contract Platforms. [Electronic resource]. Access mode : https://academy.shrimpy.io/post/the-best-smartcontract-platforms

4. Jiao J. et al. Semantic understanding of smart contracts: Executable operational semantics of solidity // 2020 IEEE Symposium on Security and Privacy (SP). – IEEE, 2020. – C. 1695–1712.

5. Wohrer M., Zdun U. Smart contracts: security patterns in the Ethereum ecosystem and solidity // 2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE). – IEEE, 2018. – P. 2–8.

6. GitHub. [Electronic resource]. Access mode : https://github.com/andriikopp/new-research-calculations/tree/main/sql_server_to_smart_contract

7. Deployed smart contract "employee_contract". [Electronic resource]. Access mode : https://ropsten.etherscan.io/address/0x47a1c8d7742f22826e89e56c98184eeb17d41540