Using a Hybrid Approach to Data Management in Relational Database and Blockchain: a Case Study on The E-health Domain

Carlos S. S. Marinho
Federal University of Ceará
Fortaleza, Brazil
sergio.marinho@lsbd.ufc.br

José S. Costa Filho
Federal University of Ceará
Fortaleza, Brazil
serafim.costa@lsbd.ufc.br

Leonardo O. Moreira
Federal University of Ceará
Fortaleza, Brazil
leonardo.moreira@lsbd.ufc.br

Javam C. Machado
Federal University of Ceará
Fortaleza, Brazil
javam.machado@lsbd.ufc.br

***Abstract*—Relational Databases (RDBs) have been widely used for decades. However, new persistence technologies are emerging, such as Blockchain, which is disruptive and has relevant properties, such as immutability and no third parties. Therefore, applications that use RDB can benefit from these properties by migrating part of their data to Blockchains. This article presents the MOON, a hybrid approach to manage data in RDB and Blockchain, which receives SQL queries. A case study was performed with a real health dataset using three scenarios. The conclusion is that the MOON responds to requests correctly and provides RDB and Blockchain features. Moreover, its response time was intermediate between RDB and Blockchains.**

***Index Terms*—Blockchain, Relational Databases, Distributed Architecture, Data Management**

I. Introduction

A database is a collection of related data, being the relational model widely used in applications of many domains [1]. This model specifies a database as a collection of one or more relations composed of rows and columns. On the other hand, Blockchain technology provides reliable, distributed, and secure support for large-scale peer-to-peer (P2P) network transactions [2]. There is a decentralized trust entity in Blockchain, without the need for reliable centralized third parties, for example, Banks, and makes it a disruptive technology. Moreover, the network participants do not necessarily trust each other to it works correctly [3].

The relational model has been used widely for decades and was substantial for the popularization of the databases [4]. Currently, many applications are to be data-oriented, using data from different types and domains [5], which makes the alternatives to the relational model have been solidifying. One of them is Blockchain, which can be used for applications to benefit from its properties, such as irrefutability and immutability.

Blockchains are advantageous to Relational Databases (RDBs) in trust-building and security aspects. However, RDBs usually have higher throughput in writing operations than Blockchains [6]. It probably occurs because, unlike Blockchains, RDBs do not implement a consensus, such as Proof of Work, used by several Blockchains in the literature. Moreover, Ruan et al. (2019) state that the main purpose of Blockchain is security and integrity, while the databases focus on performance.

An application can use part of their data that are best suited to the relational model and another part that is most suitable to Blockchain. More inherent data of Blockchain are in contexts in which there is no third party, in addition to having no trust between the nodes of the network or immutability of the data. On the other hand, adequate data to the relational model are those that are desired to perform more complex queries, using joins or analytics. Thus, the best for those applications may be to use a hybrid approach.

Accordingly to the actual state-of-the-art, there are a few works that present approaches to hybrid data management, with data storage in RDB and Blockchain. Thus, this article explains the MOON, a hybrid approach to manage data in RDB and Blockchain. Moreover, MOON uses Structured Query Language (SQL) to interact with client applications, making the approach access transparent to the model the data used.

Ross et al. (2016) define e-health as the use of computing, information, or communication technology in health or healthcare aspects. The use of e-health is essential for solving problems faced by healthcare systems, such as the aging population, providing better treatments [8]. According to Zhang et al. (2017) and Kassab et al. (2019), e-health applications are solutions that span multiple contexts in the health area, such as applications related to the patients' physiological data monitoring and drug management in the hospital infrastructure . The data of this domain may require some Blockchain properties, such as irrefutability of a medical report, data auditability, and immutability.

This article conducts a case study using the MOON in the domain of the clinical testing laboratory. The MOON's experiments and validation in this domain used real data. Furthermore, there is an evaluation of general aspects, which are response time and correctness. The main objective of this research is to develop and validate an approach that manages data stored in RDB and Blockchain. The main contributions are: (i) develop MOON, an approach to managing data in Blockchain and RDB; (ii) design a relational data mapping

978-1-7281-4659-1/20/$31.00 ©2020 IEEE
DOI 10.1109/ICSA-C50368.2020.00030

for Blockchain; and (iii) develop a case study to evaluate the proposed approach. The hypothesis is that the MOON provides the following properties to the applications: (a) complex queries, relational model properties; and (b) immutability, auditability, and no third party, inherent to Blockchain. The research questions of this article are:

- **RQ1** - Is it satisfactory to develop an approach that stores data disjointly in Blockchains and RDBs?
- **RQ2** - Given a data collection, what should be considered to decide how best to store it: Blockchain or RDB?
- **RQ3** - How to map relational data to a Blockchain?

II. Background

Blockchain provides secure, reliable, and distributed support for transactions between nodes in a large-scale P2P network [2]. It is a disruptive technology because the network participants do not necessarily have to trust with each other and there is no need for third-party (i.e. bank, notary's office, and government) to mediate transactions between them. Therefore, Blockchain has a decentralized trust entity that eliminates the third parties. Systems from several domains use Blockchain, such as health, Internet of Things (IoT), and finance [3].

In Blockchain, each block has a set of n transactions, where n differs according to the implementation of the Blockchain used, as this defines the block size. Moreover, each block has its hash and a hash pointer to its previous block, composing a chained structure of blocks. Furthermore, the first block has height 0 and is called genesis. The others have the height of their previous block plus one. To publish a new block with a set of transactions on the network, the nodes have to reach a consensus (e.g., Proof of Work and Proof of Stack).

Blockchain technology has relevant properties. It is intrinsically decentralized and has high availability since it is available even if some of the network nodes are offline. Moreover, Blockchains are auditable because transactions stored in the ledger can be inspected by the network nodes. Also, the transactions are irrefutable, since a node can not contest the authenticity of a transaction sent by itself. Finally, transactions published in Blockchains are immutable since they can not be tampered [3], [6], [11].

III. MOON Architecture

*The approach to data **M**anagement on relati**O**nal database and bl**O**ckchai**N*** (MOON) proposes to manage data in a hybrid form, partitioning it between RDB and Blockchain. Also, the SQL language is used by client applications to communicate to the MOON. Consequently, the client sends insert, update, select, and delete operations to MOON, regardless of whether the data is in a Blockchain or RDB. However, the delete operations on Blockchain data are refused, because the properties of this technology ensure no data removal.

There are significant advantages to using SQL as a communication interface between clients and the MOON. First, SQL is well-documented and widely used by developers. Using this technology avoids developers from learning a new and unconsolidated form to communicate. Also, the use of this language supports transparency, as the client does not communicate differently for each persistence model used, being MOON responsible for providing this abstraction. Consequently, there may be stakeholders interacting with the system and do not know how and where data is stored. Finally, applications that migrate their data from the relational model to the MOON do not need to rewrite their implementations of data access.

The MOON design considers three profiles of users: a final user, a developer, and a configuration user. The first consumes data through applications, such as a doctor that gets exam results on his mobile application. The second is who sends requests through an API programmatically. A developer needs information on the SQL Data Manipulation Language (DML). However, a developer does not need to know where each data item is, as the MOON provides data access using only SQL. The configuration user profile is responsible for configuring and installing the Relational Database replicas and Blockchain networks. Therefore, this user understands how to create and configure the Blockchain and database environments. Also, it is the configuration user's responsibility to set up the schema definition and configuration files that are in Figure 3.

The MOON data partitioning is vertical and its granularity is of entity. Therefore, there is the use of Blockchain or RDB to store each entity. An entity is one part of the total data that represents something to model. Examples of entities in an e-health scenario may be patients and exams. The configuration user determines which model of persistence each entity use. To make this decision, this user must analyze the requirements of the entity. Data inherent of Blockchain require security, integrity, and no third party. Data best suited to RDBs is variable and frequently gives better performance.

Using SQL adds a need to map the requests in SQL to Blockchain because the Blockchains design does not consider storing relational data. Therefore, it is necessary to change the structure of the data received on request to data store it in Blockchains, preserving its semantics. Explained in Figure 1, the proposal consists of transforming each tuple of the relational model to notation an object in JavaScript Object Notation (JSON) and then storing it in Blockchain.

Algorithm shows how MOON handles insertion operations. The received request is sent to the DBMS if the entity mentioned on request is in a RDB. Otherwise, the data is in the Blockchain and passes through two validations: (i) if the request has all the attributes of the entity, and (ii) if its values are appropriate. Next, there is a formation of JSON file using the attributes and their values, plus a special attribute that identifies the entity. Then, it sends the JSON file to the Blockchain network and creates an index entry of this data. Finally, the algorithm returns the number of affected data.

Figure 1 presents how MOON works for selects on Blockchain. Arrow 1 indicates sending a SQL request by the client to the MOON and arrow 2 points the MOON sending a request to the Blockchain. Arrows 3 and 4 indicate, respectively, the returns in JSON and table formats. Accordingly to Figure 1, the response returned by the MOON to clients is in the format of tuples, being the default.

Algorithm 1 SQL request mapping for Blockchain

```
while has SQL insert requests do
  values_received ← get_values(request);
  attributes_names ← get_attributes(request);
  entity ← get_entity_name(request);
  if entity is in database then
    send the request to DBMS;
  else if entity is in Blockchain then
    if attributes_names is in attributes(entity) then
      if values_received are valid then
        json_file ← JSON empty file;
        json_file.insert("entity" : entity_name);
        for each attr in attributes_names do
          json_file.insert(attr : value_received);
        end for
        success ← send_to_blockchain(json_file);
        if success then
          write the new index entry;
          return the number of affected data;
        else
          return error;
        end if
      else
        return error;
      end if
    else
      return error;
    end if
  else
    return error;
  end if
end while
```

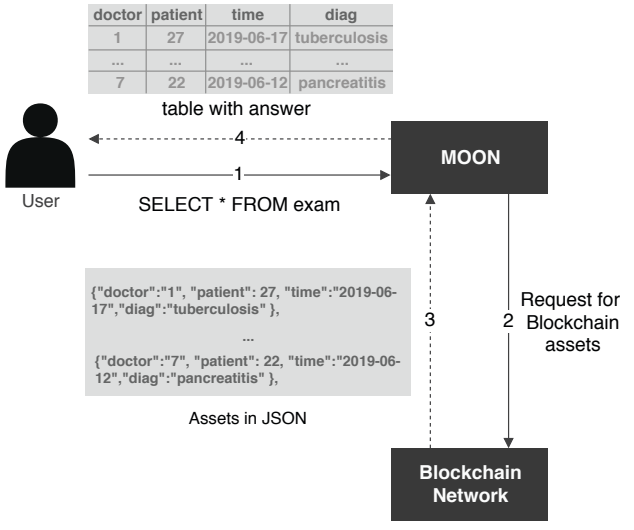

Fig. 1. Mapping from Blockchain to the relational model

When the MOON receives a join between relational and Blockchain data, a temporary table is used. It is important to note that these tables are virtual, not materialized, to minimize the overhead caused by this process. First, the Moon makes a Blockchain data retrieval, which is similar to that shown in Figure 1. Later, the MOON uses this data to create a temporary table in the database and uses it to make joins. Next, the database returns a response in tuple format to the MOON, which sends it to the client. Figure 2 illustrates this process. This strategy was adopted because, although the delay in the table creation, the DBMS uses efficient join techniques already implemented.

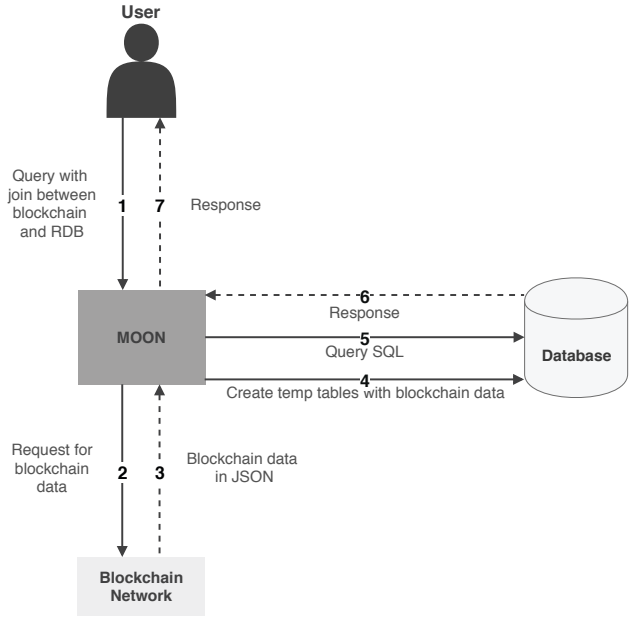

Fig. 2. Data retrieval with join between Blockchain data and database data

In the architecture presented in Figure 3, there are seven proposed modules. The user sends an SQL request to the Client Module, which interacts with the Communication Module. Next, It sends the request to the Scheduler Module, which is responsible for ordering the incoming transactions and forwarding them to the SQL Client, Blockchain Client, or both. If the request requires Blockchain data, the Mapping Module makes the needed changes to transform Blockchain data into relational data, preserving semantics. SQL Client contains drivers from many DBMSs, such as Oracle and PostgreSQL, and sends received requests to the RDB. The Index Manager stores and retrieves Blockchain index entries and the SQL Analyser examines received requests to identify pertinent information, for example, the involved entities and the type of request: select, insert, update or delete.

The MOON uses two files: the configuration file and the schema definition file. The first has the necessary information to execute the modules correctly: (i) the Blockchain infrastructure and DBMS used; and (ii) the addresses of the nodes. The schema definition file is a file in JSON format that has the following information for each entity: (i) its name; (ii) its data and types; (iii) where it is stored; and (iv) what is its

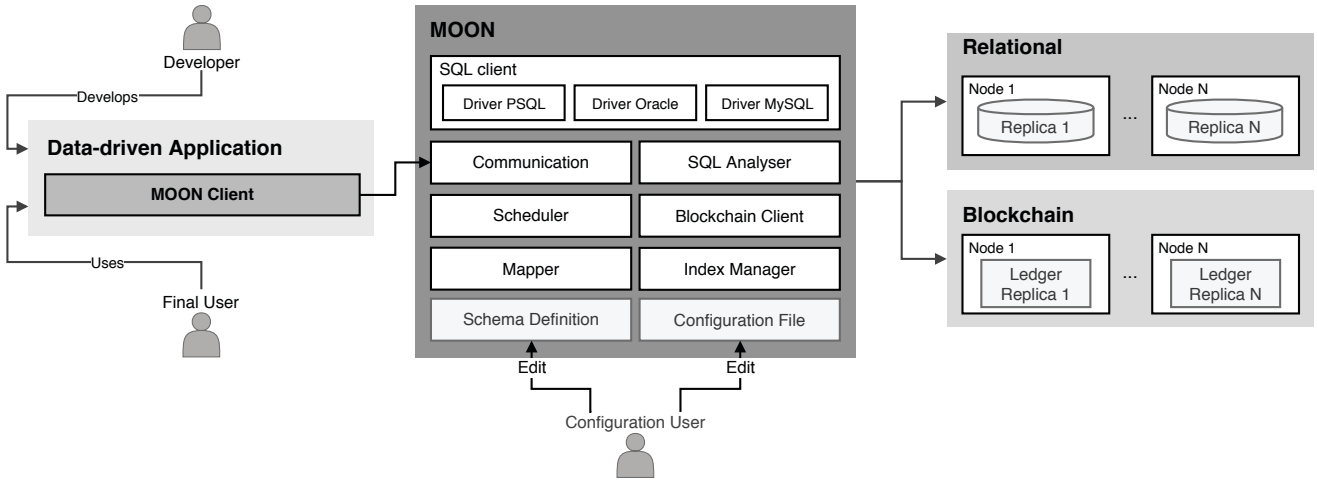

Fig. 3. Moon Architecture

primary keys and foreign keys.

As previously mentioned, the MOON architecture includes a Client Module. It is responsible for maintaining the keys used in Blockchain and the credentials of the database. Moreover, it receives SQL queries from the client and forwards them to the Communication Module, along with database credentials and Blockchain keys. Therefore, communication aspects are transparent to the user, without the need for entering the credentials and keys whenever accessing data.

Indexes are needed to solve a difference between Blockchain and RDB. Generally, Blockchains provide only sequential searches, accessing each block one by one to find any specific data. However, the identifier of a Blockchain data is its hash, and it is possible to get data directly by that identifier. In RDB, on the other hand, there are other forms of data identification, such as numeric or text used as a primary key. Hence, it is necessary to use indexes to match the identifier defined by the user and the hash identifier used by the Blockchain.

There is another reason for using indexes. Unlike databases, Blockchains do not divide data into logical entities, such as doctors, hospitals, and patient records. It probably slows down the data recovery, because it is needs checking all registries on Blockchain or index entries to find any specific data, including those in entities unrelated to desired data. To address this issue, MOON creates lists of index entries that each have only entries for a specific entity. Therefore, retrieving specific entity data becomes faster as it is not necessary to get all records and select related ones.

The Blockchain index structure uses the relational model, so each index entry is a tuple in the RDB. Each Blockchain entity matches one index relation in the RDB, which has the following information: (i) the id of the data, defined as the primary key; and (ii) the data hash on Blockchain. The management of indexes in a RDB uses the efficient features implemented by DBMS, such as concurrency control and optimized data retrieval.

The use of Blockchain indexes on databases introduces a relevant point. Someone with database access can tamper with index entries, so queries response have some omitted data. Although the data is in the Blockchain, it may not be returned in a query if its index entry is deleted. Therefore, the MOON has the function to inspect for discrepancies among index entries and Blockchain. This procedure does not use index entries, querying all Blockchain records and checking that all are present in the index entries. If not, the function returns the inconsistencies between index and Blockchain.

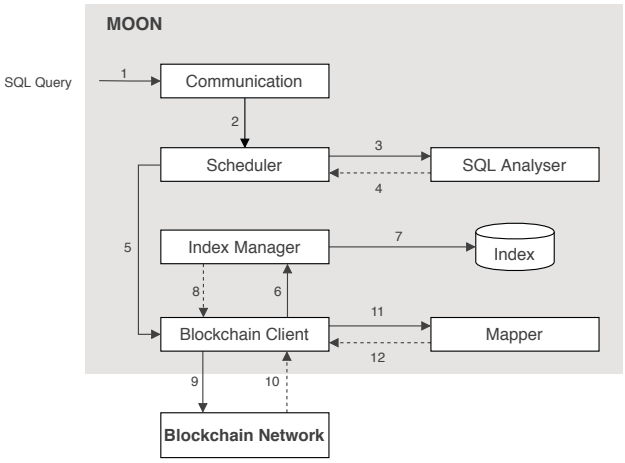

Fig. 4. Step by step of a select request in MOON for Blockchain data

Figure 4 shows the step-by-step of Blockchain data retrieval using indexes on the MOON, in which the dashed arrows correspond to returns of calls. That figure considers only the simplest case, without conditionals. First, a request is sent to the Communication Module (1). The request is sent to the Scheduler (2), and this module interacts with SQL Analyser (3). Then, It identifies where are the requested data

and the request type, and it is returned (4). After that, the request is sent to the Blockchain Client Module (5), which requests the hashes of the requested entity data to Index Manager (6), so it uses the database (7) and returns the list of hashes (8). Then, the Client Blockchain requests the data from the Blockchain network (9) and receives it (10). Next, the Blockchain Client sends the response to the Mapper (11), which changes the data to tuples format and returns it (12). Finally, the response is returned to the client, through the Scheduling and Communication Modules, respectively. These last steps have been omitted from the figure to simplify it.

IV. Case Study

The case study is an application for the domain of clinical laboratory tests. It was used a real dataset that contains data collected from blood samples from patients of a Portuguese hospital: (i) glucose; (ii) insulin; (iii) leptin; (iv) adiponectin; (v) resistin; and (vi) MCP-1. According to Figure 5, there are the following entities in the application: Patient, Doctor, Laboratory worker, and Exam.

The patient entity has all the patient's data, where some data can be changed. The doctor entity has the personal and professional data of a doctor. Also, the doctor can request exams and view the results of his patients. Laboratory workers can be biologists and biomedical, for example. The laboratory workers are responsible for quality control and release exams, signing for the exams. A laboratory worker cannot refute that he issued the result of an exam and cannot modify the result of an exam after its publication. Table I presents the requirements for the case study.

TABLE I
Case Study - Requirements

ID	Description	Details
R1	Manage patients	Add, select, update, and delete patients
R2	Manage doctors	Add, select, update, and delete doctors
R3	Manage laboratory workers	Add, select, update, and delete laboratory workers
R4	Request exam	A doctor requests an exam for a patient
R5	View the exam result	The doctor who requested the exam see the results
R6	Issue an exam result	A laboratory worker issues the result of an exam

To use MOON, one must identify how to partition entities. Patient, doctor and laboratory worker entities must be in RDB because they can receive insert, update, and delete operations, according to requirements R1, R2, and R3 in Table I. However, the exam entity must be in Blockchain, according to the requirements R4, R5, and R6.

Three scenarios using the same data schema were designed. The first, the four entities were in the RDB, without Blockchain and MOON. In the second, there is a data partitioned between RDB and Blockchain, using MOON. That is, the entities Doctor, Laboratory Worker, and Patient were in the RDB while the Exam entity was in the Blockchain. In the third, Blockchain stored all data from the four entities, without using the MOON and RDB. In this case, the sequential

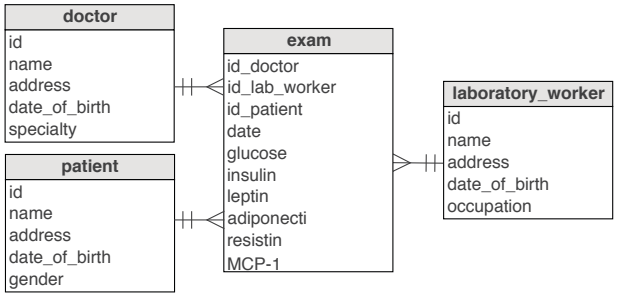

Fig. 5. Data schema used in the case study

Blockchain search was used, inspecting each block to find the desired data. The update gets the data sequentially, selects the relevant ones, changes it, and sends it to the Blockchain. The insert receives the data in JSON format and inserts it into the Blockchain network.

In total, the workload consisted of using six queries, shown in Table II. Each operation was executed 100 times in the three scenarios. The used metrics were the mean of response times and the correctness, which is the number of responses without errors and inconsistencies. Eight Virtual Machines (VMs) were used in this case study. These VMs were virtualized from a server with processor Intel(R) Xeon(R) E5645 @2.40GHz. Six of them compose the network of data nodes, each of which had 1GB RAM, 20GB of storage, and Ubuntu 18.04 LTS. Also, these nodes had a DBMS and a Blockchain node installed. The seventh had the same previous settings and was used to send requests, working as a client. Moreover, a machine with 4GB RAM and Ubuntu 18.04 LTS hosted MOON. Finally, the experiments used a local network, which does not introduce a significant network delay. Finally, there was the use of the local network.

There is the use of some technologies to support the case study. The PostgreSQL DBMS was used to manage relational data, as it is a consolidated and well-documented technology. BigchainDB[1] was the Blockchain infrastructure used because: (i) it is a robust, general-purpose and open-source infrastructure; and (ii) has a well-documented API. Moreover, for the case study's data schema to be as closer to real possible, the Mimesis framework was used to create the personal data of doctors, patients, and laboratory workers, such as address and name. Finally, the implementation of the MOON was performed in the Python programming language, allowing integration with PostgreSQL and BigchainDB.

V. Results

The results of queries Q1 to Q4 are in Figure 6, while Figure 7 shows the results of Q5 and Q6. Each query has response times for executions in the three scenarios: (i) data stored only in the RDB (Scenario 1); (ii) data managed using MOON (Scenario 2); and (iii) data stored only in the Blockchain (Scenario 3).

[1]BigchainDB. Available at: https://www.bigchaindb.com/. Accessed: January 4, 2020.

TABLE II
CASE STUDY - USED QUERIES

ID	Query
Q1	SELECT * FROM exam
Q2	SELECT * FROM exam as e, doctor as d WHERE e.id_doctor = d.id
Q3	UPDATE laboratory_worker SET name = '444 Alhambra Park' WHERE id=1
Q4	UPDATE exam SET glucose = 87, insulin = 71, leptin = 41, adiponectin = 34, resistin = 24, MCP-1 = 1 WHERE id_doctor = 3 AND id_lab_worker = 2 AND id_patient = 99 AND date = '2019-02-12'
Q5	INSERT INTO doctor VALUES (0, 'Chassidy Fischer', '384 Whitney Garden', '1971-01-18', 'obstetrician')
Q6	INSERT INTO exam VALUES (1, 2, 86, '2020-01-08', 70, 2.7, 8.8, 9.7, 7.9, 417.1)

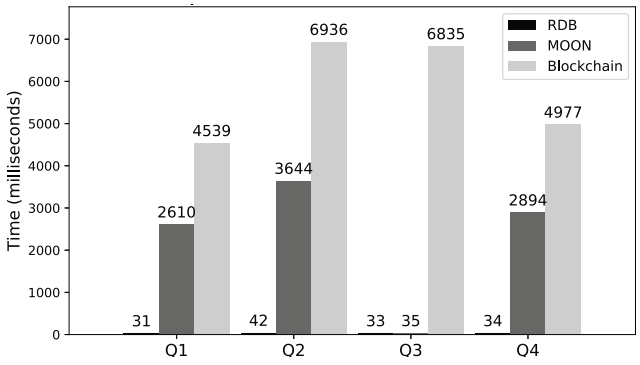

Fig. 6. Response times from Q1 to Q4

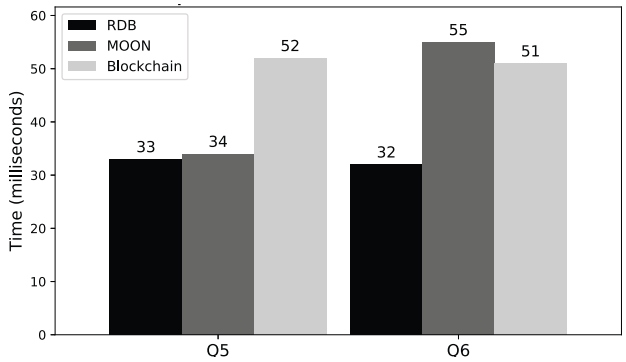

Fig. 7. Response times for Q5 and Q6

The responses of the MOON confirm their correctness because it returned the same data instance obtained if all data were stored in the traditional RDB, as expected. The comparison with the RDB responses is appropriate as it is a consolidated and validated technology. Hence, the proposed architecture works correctly, including the models of indexing and mapping.

Q1 results indicate that indexing MOON significantly reduces search time on Blockchain. Scenario 3 checked all Blockchain records sequentially, including those of Doctor, Laboratory Worker, and Patient, to collect data from the Exam entity. MOON did it faster by retrieving the Exam index entries in the database and then requesting only it from Blockchain. Consequently, the response time for Scenario 2 was approximately 42.5% less than Scenario 3. However, RDB was substantially faster than the MOON, as it did not manipulate data in Blockchain, as expected.

In Q2, the response times for Scenario 2 were approximately 46.5% faster than Scenario 3. The DBMS data retrieval is more efficient than getting the same data from the Blockchain. Therefore, the MOON was faster than Scenario 3 as only Exam data was recovered in the Blockchain, while the recovering of Doctor data was in the RDB. Moreover, MOON used indexing to obtain Exam data, while Scenario 3 used sequential search, as in Q1. It is important to notice that the use of a temporary table in RDB to execute the WHERE clause increases the MOON's response time. However, it can faster than checking record by record because it uses the optimized implementation of the DBMS.

The use of the MOON was quite beneficial in Q3. After receiving the request, the MOON verified in its scheme that the Laboratory Worker entity was in the relational model. Next, it forwarded the request to the RDB, which returned efficiently. Since only these two steps distinguished Scenario 2 from Scenario 1, the difference between their response times was just 2 milliseconds. On the other hand, Scenario 3 was slower because it searched for data sequentially in the Blockchain, selected the desired data, changed it and then sent it to the Blockchain. Therefore, if there is no need for all entities to be in the Blockchain, the use of MOON is advantageous. It provides good performance to data that does not need to be in the Blockchain and integrates it with those that require to be there.

The results of Q4 confirm those of Q2. The use of indexes to obtain Exam data and their migration to a temporary table to run WHERE explain the shorter time for Scenario 2 compared to Scenario 3. On the other hand, Scenario 1 was substantially faster than the other two approaches, however, it does not provide Blockchain characteristics to the data, which may be necessary.

The results of Q5 and Q6 attest that MOON does not add significant overhead while managing inserts. In Q5, MOON checked where the Doctor entity was and sent the received request to the RDB, being 1 millisecond slower than Scenario 1. In Q6, after verifying where the Exam entity was, MOON performed the mapping of the request to JSON and sent it to the Blockchain, being 4 milliseconds slower than Scenario 3. This was expected because the API call to send data to the Blockchain is the same in Scenarios 2 and 3. However, Scenario 3 was faster because it does not map data and there is no verification of where the data is located. Moreover, Blockchain inserts were fast due to: (i) it has used a network with a small number of nodes; and (ii) the BigchainDB consensus is based on votes, which is not so expensive.

VI. Related Work

The following search criteria was used to collect related works: (i) works that use Blockchain or its concepts; and (ii) works that use RDBs or their properties. The search for papers from 2015 to 2020 used the following repositories: ACM Digital Library, IEEE Xplore Digital Library, ScienceDirect, Proceedings of the Brazilian Symposium on Databases (SBBD) and Journal of Information and Data Management (JIDM).

Table III compares the works presented with the MOON. For each work, there is a check if it uses: (i) data partitioning between Blockchain and RDB; (ii) languages like SQL; (iii) simultaneous use of Blockchain and RDB; and (iv) Blockchain data indexing.

TABLE III
Comparison of related works

Work	Data Partition	Use of SQL-like	Blockchain and RDB simultaneously	Blockchain data indexes
[12]		✓	✓	
[13]		✓		
[14]		✓		✓
[15]		✓	✓	
[16]	✓	✓	✓	✓
MOON	✓	✓	✓	✓

A. R3 Corda

Hearn (2016) described R3 Corda, a permissioned Blockchain specific to the financial sector that stores data in RDB. The Corda's objective is to provide the Blockchain properties, such as immutability, and RDB aspects, such as complex SQL queries. Moreover, Corda restricts access to data in a transaction using need-to-know protocols, which only share data with those directly included in the transactions, not with all nodes on the network. Furthermore, this approach uses the Unspent Transaction Output (UXTO) model, avoiding double-spent. Finally, Corda supports smart contracts, running it on the Java Virtual Machine, and is restricted to validate transactions by accepting or refusing them.

Corda has some significant correlations to the MOON, but some points differ. First, the mapping done by Corda is Object-Relational, by annotations defined in the Java Persistence API (JPA), which makes it dependent on that technology. Also, there is no data partitioning between databases and Blockchains, but the merging of the two technologies to create a new approach. Finally, as it is specific to the financial domain, Corda does not support applications of other contexts. MOON is designed to be of general use, supporting e-health and data, for example.

B. A general framework for blockchain analytics

Bartolleti et. al (2017) presented a framework to perform analytics on Blockchain data in the cryptocurrency context, supporting Bitcoin and Ethereum. The approach is a Scala library that can be used to construct a Blockchain view and store it in SQL or NoSQL database. The main idea is to represent Blockchain primitive entities in Scala classes, such as block and transactions, and then store them in a database. Therefore, executing queries is available using the database communication interface, such as SQL language if the database is relational.

Another feature of the presented framework is that it enables the integration of Blockchain data with data from external sources. An example is integrating data from a Blockchain with exchange rate data between the traded cryptocurrency and the dollar, to facilitate conversions and analysis. Thus, the research does not discuss how to manage data stored in different infrastructures. However, there is a focus on the need to provide ways to simplify Blockchain access using consolidated and widely used technologies. Hence, this framework provides complex queries and analytics on data from Blockchains.

C. Ethereum Query Language (EQL)

Bragagnolo et. al (2018) developed the Ethereum Query Language (EQL), supporting to get data from an Ethereum Blockchain through analogous to SQL queries. Without this, to find specific data in Ethereum, it is necessary to access the blocks using a unique identifier or search multiple blocks sequentially. Moreover, the EQL provides rich syntax that allows specifying data elements to search for information spread across multiple records. The approach uses index files structured in a Binary Search Tree (BST) to manipulate data in Blockchain internally. Furthermore, the research exposes some advantages of using this language, such as describing filters to get information, ordering query results, and limiting the number of results returned.

A relevant aspect of the relational model leads research: the use of a powerful data query language similar to SQL. However, the work does not present characteristics of hybridism between the relational model and the Blockchain. Nevertheless, the article indicates future work that has a strong correlation with MOON. It is proposed to use the language to query data from different sources, not only Blockchains.

D. ChainSQL

Muzammal, Qu, and Nasrulin (2019) presented the ChainSQL, a Blockchain-based log database system. The ChainSQL domain is financial institutions distributed geographically, where there is a large volume of business transactions. In this approach, the Blockchain stores transactions, while the database stores current data. The architecture of ChainSQL receives a transaction and sends it to a Blockchain node. After consensus, the transaction is forwarded to the database. Hence, there is a synchronization between database and Blockchain that occurs in two ways: synchronize-at-each-transaction and synchronize-at-each-interval. Moreover, ChainSQL provides an API that supports queries on SQL and JSON with a specific format.

Although it has several points similar to ChainSQL, MOON has some significant differences. First, MOON does not propose to perform database-blockchain synchronization, since there is no data replication because the purpose is to partition

them. Keeping the same current data in the two models is likely to be computationally costly. Second, the article that presents ChainSQL does not discuss Blockchain data indexing or relational model mapping to the Blockchain, which are points of the present article.

E. SEBDB: Semantics Empowered BlockChain DataBase

Zhu et. al. (2019) proposed SEBDB, an approach that includes relational data semantics to Blockchains, where each transaction is associated with a tuple with multiple attributes from a specific table. SEBDB uses an SQL-like language as its communication interface to support application development. Moreover, it uses RDBs, although there is no trust in them as there may be changes in the data stored in it. RDBs are used to store off-chain data, which is data that does not be in the Blockchain network. SEBDB joins data from the Blockchain and the RDB, which is named on-off join. Furthermore, SEBDB uses index files for Blockchain data because it is inefficient to process a request scanning blocks one by one since tuples in a block can belong to multiple tables.

There are some differences between MOON and SEBDB. The first point is that, in SEBDB, there is no data replication in RDB, each network node has a different data schema and is not necessarily well defined. Consequently, the RDB schema is not part of the SEBDB, as it does not know about it. Another aspect is that there is no discussion about data mapping between the Blockchain and RDB. Finally, the way to index data is different between them, because the MOON uses it in database relations and the SEBDB uses index files.

VII. Conclusion

This article presents the MOON, an approach to managing partitioned data between RDB and Blockchain that accepts SQL requests. The partitioning gives benefits: (i) allows data to transparently be stored in the blockchain, taking advantage of important properties such as immutability and no third parties; and (ii) data mutability and complex queries, properties of RDB. As part of the MOON, there is a description of mapping data between RDB and Blockchain and a general architectural model to support applications from different domains.

There is a case study using the MOON on the e-health domain with real data. It demonstrates that the MOON returns correct responses to clients, that is, the same data instance obtained if all data were stored in the traditional RDB. Moreover, considering the response times obtained, the MOON was slower than data stored only in RDB, which may be acceptable in some data domains, given the benefits provided. As a limitation of the experiments conducted, the data volume sent to the approach was not large.

Future work opportunities arise from this work. One is studying how MOON performs in networks with a large number of nodes, such as hundreds, to evaluate whether this adds any substantial communication or processing overhead. Also, the use of other workloads introduces higher reliability and attests to the robustness of the approach presented. There are plans to evaluate the solution under different workloads: transactional order processing (TPCC); (ii) web-oriented social network (Twitter); and (iii) key-value (YCSB) [17]. Also, the next experiments using MOON should use large volumes of data to evaluate its scalability.

Acknowledgment

This study was financed by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. This research was partially supported by Laboratory of Systems and Databases (LSBD).

References

[1] R. Elmasri and S. Navathe, *Fundamentals of database systems*. Pearson, 2015.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[3] F. Greve, L. Sampaio, J. Abijaude, A. Coutinho, Ítalo Valcy, and S. Queiroz, *Blockchain e a Revolução do Consenso sob Demanda*. SBC, Maio 2018, ch. 5, pp. 1–52. [Online]. Available: http://portaldeconteudo.sbc.org.br/index.php/minicursossbrc

[4] A. Pavlo and M. Aslett, "What's really new with newsql?" *ACM Sigmod Record*, vol. 45, no. 2, pp. 45–55, 2016.

[5] P. Maenhaut, H. Moens, V. Ongenae, and F. D. Turck, "Design and evaluation of a hierarchical multi-tenant data management framework for cloud applications," in *2015 IFIP/IEEE IM*, May 2015, pp. 1208–1213.

[6] M. J. M. Chowdhury, A. Colman, M. A. Kabir, J. Han, and P. Sarda, "Blockchain versus database: A critical analysis," in *2018 17th TrustCom / 12th BigDataSE*. IEEE, 2018, pp. 1348–1353.

[7] P. Ruan, G. Chen, T. T. A. Dinh, Q. Lin, D. Loghin, B. C. Ooi, and M. Zhang, "Blockchains and distributed databases: a twin study," 2019.

[8] J. Ross, F. Stevenson, R. Lau, and E. Murray, "Factors that influence the implementation of e-health: a systematic review of systematic reviews (an update)," *Implementation Science*, vol. 11, no. 1, p. 146, Oct 2016. [Online]. Available: https://doi.org/10.1186/s13012-016-0510-7

[9] P. Zhang, M. A. Walker, J. White, D. C. Schmidt, and G. Lenz, "Metrics for assessing blockchain-based healthcare decentralized apps," in *2017 IEEE 19th International Conference on e-Health Networking, Applications and Services (Healthcom)*, Oct 2017, pp. 1–4.

[10] M. Kassab, J. DeFranco, T. Malas, G. Destefanis, and V. Graciano Neto, "Exploring research in blockchain for healthcare and a roadmap for the future," *IEEE Transactions on Emerging Topics in Computing*, vol. PP, 08 2019.

[11] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *2017 IEEE International Congress on Big Data (BigData Congress)*, June 2017, pp. 557–564.

[12] M. Hearn, "Corda: A distributed ledger," *Corda Technical White Paper*, 2016.

[13] M. Bartoletti, S. Lande, L. Pompianu, and A. Bracciali, "A general framework for blockchain analytics," in *Proceedings of the 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*, ser. SERIAL '17. NY, USA: ACM, 2017, pp. 7:1–7:6. [Online]. Available: http://doi.acm.org/10.1145/3152824.3152831

[14] S. Bragagnolo, H. Rocha, M. Denker, and S. Ducasse, "Ethereum query language," in *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*, ser. WETSEB '18. NY, USA: ACM, 2018, pp. 1–8. [Online]. Available: http://doi.acm.org/10.1145/3194113.3194114

[15] M. Muzammal, Q. Qu, and B. Nasrulin, "Renovating blockchain with distributed databases: An open source system," *Future Generation Computer Systems*, vol. 90, pp. 105 – 117, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X18308732

[16] Y. Zhu, Z. Zhang, C. Jin, A. Zhou, and Y. Yan, "Sebdb: Semantics empowered blockchain database," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, April 2019, pp. 1820–1831.

[17] D. E. Difallah, A. Pavlo, C. Curino, and P. Cudre-Mauroux, "Oltp-bench: An extensible testbed for benchmarking relational databases," *Proceedings of the VLDB Endowment*, vol. 7, no. 4, pp. 277–288, 2013.