ORNL/TM-2019/1253

Extending the Blockchain: Ensuring Transactional Integrity in Relational Data via Blockchain Technology

W. B. Ray

August 14, 2019

OAK RIDGE NATIONAL LABORATORY
MANAGED BY UT-BATTELLE FOR THE US DEPARTMENT OF ENERGY

ORNL/TM-2019/1253

Data System Sciences and Engineering
Center for Infrastructure Security Analysis

Extending the Blockchain: Ensuring Transactional Integrity in Relational Data via Blockchain Technology

William B. Ray

August 14, 2019

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, TN 37831-6283
managed by
UT-BATTELLE, LLC
for the
US DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

CONTENTS

ACRONYMS

API	application programming interface
DML	data manipulation language
ID	identifier
LOB	large object
ORNL	Oak Ridge National Laboratory
RDBMS	relational database management system
SQL	Structured Query Language

ABSTRACT

The idea of combining a relational database with concepts associated with blockchain technology is being examined. The use case outlined herein concurrently utilizes relational and blockchain data structures to facilitate the sharing of complex data among untrusted users without the participation of a trusted third party. By capturing the transactions that result in data being stored in a relational database and placing those transactions in a blockchain structure, users can be assured that the information has not been created, altered, or destroyed without the knowledge of all users of the data.

Capturing relational database transactions via a blockchain structure presents several challenges. Among them are increased storage capacity requirements, transaction ordering prior to placement on the blockchain within a multiuser database, use of existing or creation of new consensus mechanisms, and timely data verification. Addressing these challenges, although not unsurmountable, will introduce trade-offs (e.g., increased system sizing, slower transaction execution) that might or might not be acceptable based upon the envisioned use. If these trade-offs are acceptable, then this concept of extending the blockchain is a powerful method to provide both applications and their users immutable, trusted, well-defined, and consistent data.

1. OVERVIEW

The idea of combining a relational database with concepts associated with blockchain technology resulted from the 2018 ORNL Blockchain Initiative. The Initiative consisted of a core group of researchers who had already been pursuing the technology to investigate potential use cases, establish partnerships, and determine a wise course for the future.[1] The Initiative determined that blockchain technology is not applicable to most of the use cases for which it has been proposed as a possible solution; however, the trust and transparency that it does afford can be leveraged in creative ways that might not be initially obvious. One such application is merging the relational and blockchain data paradigms into a cohesive system that extends blockchain technology past previously envisioned uses.

2. BLOCKCHAIN DEFINITION

Very simply, a blockchain is a growing list of records (or blocks) that are linked together via a cryptographic hash function.[2] In other words, it is an append-only database that is resistant to modification, because the alteration of a single record would require the alteration of all subsequent records.

The structure of the blockchain (see Figure 1) utilizes the record identifier (ID), the information stored in the record (including a timestamp), and the cryptographic hash computed for the previous record to create a new hash that is stored with the current record. If any piece of that information were to be altered in any way, re-computation of the hash would result in a significantly different result. This ensures that the information stored on the blockchain is both transparent and trusted.

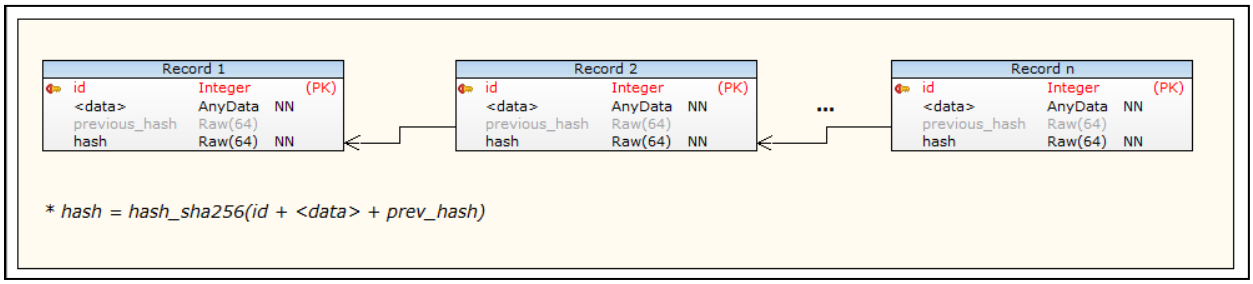

Figure 1. Simplified blockchain structure.

The structure described in Figure 1 is the blockchain at its most basic level. More sophisticated blockchain implementations include structures like Merkle trees within each block to allow for easy verification of the consistency and content of a large body of data (e.g., financial transactions).[3]

[1]Oesch, Sean, et al. *Blockchain at ORNL The What, Why and How*, p. 3. ORNL/SPR-2018/1032, 2018.
[2]A cryptographic hash function can accept a string of arbitrary size and generate a fixed-size value that is practically infeasible to invert.
[3]In order to simplify the concepts presented in this paper, more sophisticated implementations will not be addressed.

From a data storage and management perspective, blockchain technology is extremely limited. The append-only data structure is simple and well-suited to applications like cryptocurrencies. The data needs of most modern software programs, however, are well-served by the relational database and the complex data structures afforded by this mature, widely used, powerful technology. In Section 3, a use case for extending the blockchain concept and its secure digital assets into the realm of the relational will be discussed.

3. USE CASE

The use case for concurrently utilizing relational and blockchain data structures involves two or more users who wish to share more complex information but do not trust each other, whereas if the users were trusted, or if the system could utilize a trusted third party, then the blockchain structures would not be needed. These types of users might be different corporations, universities, national laboratories, or individuals who can benefit by sharing information but need to ensure that the information being shared is verifiable. In other words, the originator and the consumers of the information can be assured that the data is consistent and has not been altered.

Figure 2 illustrates a strategy for tying the two data technologies together. This concept involves several interrelated concepts and structures that will be detailed in this section.

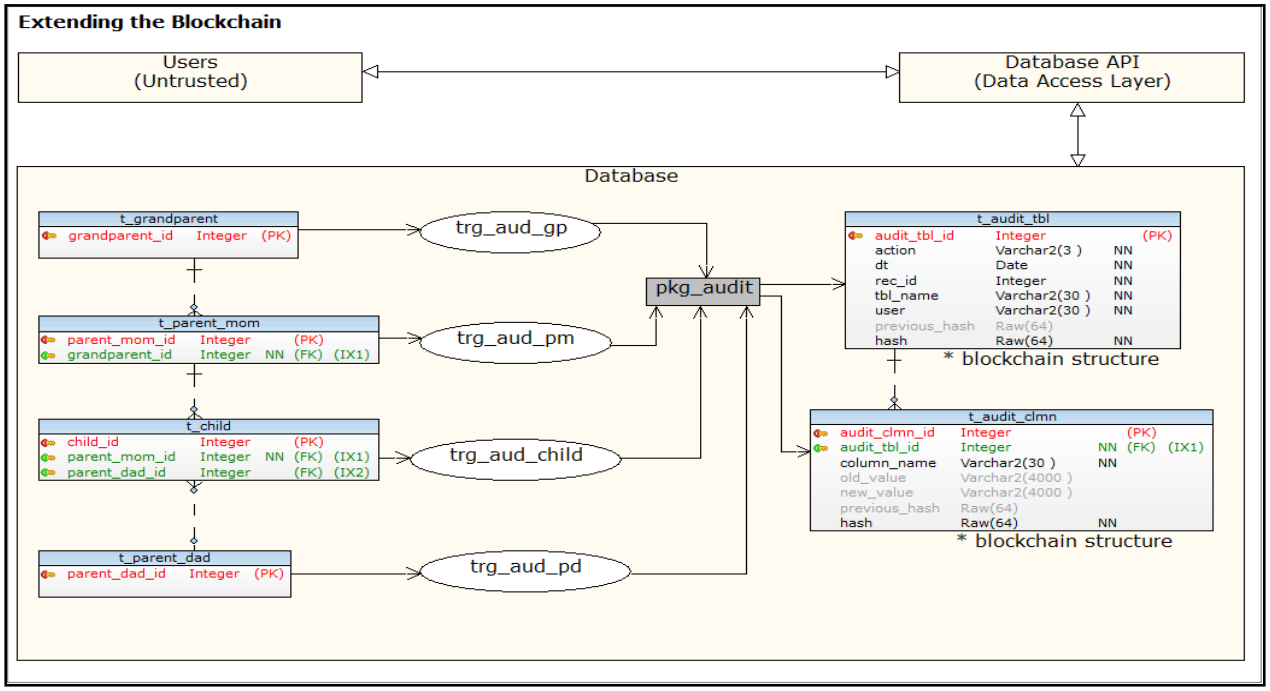

Figure 2. Extending the blockchain.

3.1 USERS

As described in the use case, two or more untrusted users will access the database directly through the database application programming interface (API) or, more likely, through an application that utilizes the API.

3.2 DATABASE API

The database API provides a set of packaged subroutines that exposes the relational data to users via procedures and functions that accept input and produce output in the desired format. The API consists of code that performs all the actions that are needed by an application or user of the database. Examples of this functionality would be retrieving individual or sets of records, executing data manipulation language (DML) statements, transforming data, and implementing business logic. The API provides a single means of accessing and manipulating the data, and, if needed, validates user access. This interface provides this functionality without requiring the user to understand how the data is structured or issue individual Structured Query Language (SQL) statements to manipulate or view the desired information.

Authorized users of the database can utilize the API through one or more applications to fulfill their business requirements. Depending upon the use case, these untrusted users might be anonymous or known. Known, trusted users would not need the blockchain.

3.3 DATABASE

The database contains many different types of objects to support this use case. These include application tables, audit tables, triggers, and packages.

3.3.1 Application Tables[4]

Data is stored in a typical relational structure, which is two or more tables identified by primary keys related to each other via foreign keys. Each table column can be of any data type supported by the relational database management system (RDBMS), including string, date, numeric, and large object (LOB).

3.3.2 Triggers and Audit Package

Each application table in the database has an associated trigger with a naming convention of 'trg_aud_<tbl>' where '<tbl>' is the associated table name without a prefix. These triggers are constructed automatically via a code generator that creates the trigger based upon the database metadata. These triggers fire (i.e., execute code) whenever an insert, update, or delete DML action occurs. The triggers utilize the audit package, pkg_audit, to insert records into audit tables that record the action performed, affected table, record ID, date and time, user, columns affected, and old and new values.

3.3.3 Audit Tables

The audit tables, t_audit_tbl and t_audit_clmn, are populated as simple append-only data structures that record every DML action (i.e., transaction) performed against the application tables.[5] The records must be added in order of execution to ensure that the application table data can be reproduced accurately. By adding a hash of the data associated with each transaction (which includes the hash of the previous transaction), an immutable, trusted blockchain is created.

[4]Tables that store data to support the applications utilized by the users are herein referred to as "application tables" to differentiate from them from "audit tables" that are used to store the transactions in a blockchain structure.
[5]The audit tables are still technically relational tables; however, the only action that is ever performed on them is an insert. No updates or deletes are executed against these tables.

3.4 DATA VERIFICATION

The blockchain structure of the audit tables allows verification of the application data, as well as the blockchain itself.

3.4.1 Application Data Verification

Since all transactions have been captured in the blockchain, the current state of the database is easily verifiable. By creating a copy of each application table, the transactions stored in the audit tables can be re-executed against these temporary table copies. Once all the transaction DML statements have been processed, a dynamic query can be created and executed to compare the current state of the application table against its temporary copy. If any differences are encountered, then it can be deduced that data was changed in the application tables without the corresponding transaction being recorded on the blockchain. For example, a bad actor might have disabled the triggers that record transactions on the blockchain and surreptitiously modified data in the application tables. Utilizing the blockchain for verification allows the other database users to catch this activity and decide how to proceed.

Additionally, the database state at any point in time can be reconstructed and used to settle any conflict or disagreements among the untrusted users. For example, if other users detect that a record is unexpectedly missing in the application data, the blockchain can be queried to determine who deleted the record, when it was deleted, and the contents of the record prior to deletion.

3.4.2 Audit Table Data Verification

The audit tables' blockchains must be validated as well as the application data. The blockchain structure lends itself to being easily queried and to having the cryptographic hashes recalculated based upon both the current block and the hash of the previous block. If this recalculated hash does not match the value stored within the block, then it is certain that the blockchain has been modified. Additionally, once a recalculated hash has been found to be incorrect, then all subsequent hashes also will be incorrect due to the linking of the records.

A broken blockchain indicates that a bad actor has attempted to modify the information related to a transaction that is associated with building the current state of the data in the application tables. At this point, the application data can be rebuilt based upon the transactions stored on the blockchain prior to the block with the corrupted hash.

4. CHALLENGES

The idea of combining a relational database with blockchain technology must surmount several challenges in order to become a truly viable computing solution.

1. Storing a transaction that creates, modifies, or deletes each record in the database can generate large volumes of data. The database administrator must ensure that there is enough space to store the data and to perform the application data verification.
2. Ordering of the blockchain is critical to ensuring that the hashes are correct and that the transactions are stored in the proper order. A multiuser database environment with rollback mechanisms and delayed commits complicates this ordering. A mechanism must be developed to ensure that only committed transactions are stored on the blockchain and that they are written in an ordered, timely manner.

3. Blockchains must use a consensus mechanism to ensure that the blockchain can be trusted. Bitcoin uses proof-of-work. This mechanism mandates that the calculated hash for each block must meet certain criteria. Meeting these criteria ensures that the hash is computationally expensive to produce and not easily hacked. Capturing database transactions and storing them on a blockchain also requires an acceptable consensus mechanism that ensures users have confidence in the resulting blockchain.
4. Each block on the blockchain represents a single transaction in the database. As mentioned earlier in this paper, most blockchain implementations utilize a Merkle tree to store multiple transactions on a single block. This method can be used here; however, the data verification process becomes more complicated and possibly more time consuming.

5. CONCLUSION

This simple concept expands the usefulness of the blockchain data model to more complex relational data structures that are better aligned with current data manipulation needs. There are, however, challenges that complicate the actual implementation of this idea. This is an area ripe for research and exploration.

REFERENCES

Oesch, Sean, et al. 2018. *Blockchain at ORNL*: *The What, Why and How*, ORNL/SPR-2018/1032, p. 3.