Journal of Information and Data Management, 2024, 15:1, doi: 10.5753/jidm.2024.4290

Inter-MOON: Enhanced Middleware for Interoperability between Relational and Blockchain-based Databases

Rafael Avilar Sá [**Universidade Federal do Ceará** | ***rafael.sa@lsbd.ufc.br***]
Leonardo O. Moreira [**Universidade Federal do Ceará** | ***leonardo.moreira@lsbd.ufc.br***]
Javam C. Machado [**Universidade Federal do Ceará** | ***javam.machado@lsbd.ufc.br***]

Laboratório de Sistemas e Banco de Dados, Universidade Federal do Ceará, Campus do Pici - Bloco 952, Fortaleza, CE, 60440-900, Brazil.

Received: 22 March 2024 • **Published:** 18 November 2024

Abstract Multi-model architectures enable the querying of data from different sources through a unified interface, providing interoperability among databases. However, support for blockchain-based databases is still scarce. Inter-MOON is a new approach that aims to promote the interoperability of blockchain-based and relational database systems through the virtualization of blockchain assets in a relational environment, allowing for the execution of all four basic SQL DML commands. Through experimentation, results indicate that Inter-MOON provides near total support for SQL SELECT query syntax and exhibits performance comparable to or better than similar tools. This work is an extension of the original work that introduces Inter-MOON.

Keywords: Databases, Blockchain, Database Interoperability, Data Transformation, Middleware, SQL

1 Introduction

The blockchain, originally conceived as part of the Bitcoin electronic cash system [Nakamoto, 2008], allows storing data without a trustworthy third party to create an immutable, irrefutable, and tamper-proof distributed linked list. However, blockchains are characteristically slow at writing operations [Zheng *et al.*, 2018]. This is partly done on purpose due to the usage of computationally-intensive security algorithms such as *Proof-of-Work* (PoW) [Gervais *et al.*, 2016]. On the other hand, while many relational databases offer great performance, they cannot easily produce the same level of security and integrity as blockchains. Therefore, they have different priorities and divergent data models, each presenting unique challenges.

Given the diverse characteristics of data and the variety of available storage solutions, enhancing the interoperability of heterogeneous data systems has become imperative [Babcock *et al.*, 2002; Stonebraker and Cetintemel, 2018]. Federated databases, multistores, and polystores exemplify this trend. Despite the increasing adoption of blockchain technology [Gadekallu *et al.*, 2022], enhancing the interoperability of blockchain with other systems remains a challenge [Belchior *et al.*, 2021; Meyer and dos Santos Mello, 2022; Maciel *et al.*, 2023].

The *approach to data **M**anagement on relati**O**nal database and bl**O**ckchai**N*** (MOON) [Marinho *et al.*, 2020] is a tool meant to act as a singular entry-point for database queries by applications using both blockchain-based and relational databases. It is a middleware that enables querying both data stores using only SQL by mapping the SQL syntax into appropriate blockchain (BC) operations. However, MOON contains limitations regarding accepted SQL syntax, performance, and interoperability between blockchain and SQL. It does not support all DML operations or consider scenarios in which entities may change over time.

In this work, we present Inter-MOON, a new approach based on MOON and focused on enhancing interoperability between blockchain-based and relational databases via virtualizing blockchain assets in the relational environment. It supports non-distributed queries containing single SQL DML statements and allows for querying any blockchain-based or relational entity defined in the Inter-MOON entity schema. Our approach addresses challenges like how to query, modify, or delete blockchain data using SQL. In a supply chain scenario, for instance, it enables the execution of standard SQL queries across both data stores, facilitating comprehensive querying while maintaining consistency. This versatility extends to other applications where interoperability between blockchain and existing relational databases is desirable, including sectors such as healthcare and the IoT [Guo and Yu, 2022]. In summary, our contributions are:

1. Exploration of interoperability of relational and blockchain databases.
2. Proposal and development of Inter-MOON, a novel approach to interoperability between blockchain-based and relational databases with improved SQL grammar support, blockchain asset mapping, database support, query processing speed, blockchain asset retrieval speed, integrity, and reliability.
3. Testing and comparison of MOON, Inter-MOON, and another available open-source polystore solution.

This work represents an extension of the original work titled ”*Improving Interoperability between Relational and Blockchain-based Database Systems: A Middleware approach*” [Sá *et al.*, 2023]. Extensions include the addition of a background section (Sec. 2), further exploration of related works, updated experimental results, a more in-depth description of the proposed approach, and the publication of a repository containing a prototype of the proposed approach.

2 Background

In this section, we will describe some basic concepts relevant to the body of this work.

2.1 Blockchain

In the Bitcoin paper, [Nakamoto, 2008] describes an electronic payment system based on cryptographic proof instead of trust, supported by a peer-to-peer distributed ledger technology. This ledger, which came to be called Blockchain, essentially takes the form of a linked list of blocks that contain transactions validated by the network members, connected by cryptographic hashes calculated using a Proof-of-Work (PoW) algorithm.

Blockchain has seen continuous research, with many works attempting to apply the technology to areas such as healthcare, supply chain, information systems, Internet-of-things, databases, security, privacy, and voting [Krichen *et al.*, 2022; Javaid *et al.*, 2021; Guo and Yu, 2022; Gamage *et al.*, 2020]. Other research tackles different approaches to consensus mechanisms, hashing functions, smart contracts, and other properties of blockchain as a data model [Guo and Yu, 2022]. Scalability is still an open issue in blockchain research [Gamage *et al.*, 2020; Zhou *et al.*, 2020].

The following is a general description of some technical terms related to blockchain architecture used throughout this work:

- *Assets*: Digital objects representing various forms of data, including digital representations of physical objects or pure data such as text, files, or tokens. They may be stored either on the blockchain (*on-chain)* or outside of it (*off-chain*), in IPFS networks, or similar. If stored off-chain, the blockchain will often store a hyperlink to these assets instead.
- *Transactions*: Contain information regarding the state and ownership of assets. Each transaction will have a unique ID (identifier) in the form of a hash.
- *Metadata*: This represents contextual information regarding the transaction. This information can include details such as the timestamp of the transaction. Usually, metadata is not immutable, unlike assets and transactions.

2.2 Blockchain-based databases

Generally speaking, a blockchain-based database is a database that implements blockchain features, such as tamper-proof mechanisms, consensus, decentralization, or user-owned assets [McConaghy *et al.*, 2016]. Examples include BigchainDB [Bigchain and Gmb, 2018], ChainifyDB [Schuhknecht *et al.*, 2021], ChainSQL [Muzammal *et al.*, 2019], and ProvenDB[1]. They often employ two main components: a blockchain networking service and a decentralized database. The blockchain data will be stored in the database, which can be SQL, NoSQL, or another kind of database. The blockchain networking service will implement a consensus mechanism and replication the database data across the network. Some consider blockchain-based databases preferable alternatives to blockchain in contexts outside of digital currencies due to providing properties like advanced querying mechanisms and high-rate transaction output to blockchain [Tseng *et al.*, 2020].

[1]`https://www.provendb.com/litepaper`. Accessed: Feb 27, 2024

3 Related Work

For multistores, MISO [LeFevre *et al.*, 2014] focuses on the optimal materialization of data in heterogeneous big data environments, using both RDBMS and HDFS (Hadoop). Inter-MOON is not meant for big data workloads and uses virtual tables created on the RDBMS store to minimize processing on the blockchain component.

CloudMdsQL [Bondiombouy *et al.*, 2016] is a cloud-based multistore system with a SQL-like language that enables querying of relational and NoSQL sources while taking advantage of each source's native functions. We apply native SQL instead, focusing on blockchain-based DBs rather than generalized NoSQL.

[Duggan *et al.*, 2015] introduces the BigDAWG polystore, which enables heterogeneous data retrieval using custom markup. It presents the concept of *islands of information* and uses a subset of each query language associated with a data model. On the other hand, Polyphony-DB [Vogt *et al.*, 2018] conceptualizes a self-adaptive system with data replication and partitioning. [Singhal *et al.*, 2019] also presents the building blocks of Polystore++, which envisions a highly performance-oriented polystore solution.

This work utilizes a 'one-size-fits-all' approach to query languages using SQL, while polystores generally strive for mixed query languages. Moreover, none of these systems consider blockchain-based databases, a primary concern in this work. Finally, this work also showcases experimental results and a prototype, while the others are vision papers. To the best of our knowledge, no known federations, polystores, or similar offer explicit support for blockchain solutions.

In querying blockchain using SQL, [Yue *et al.*, 2019] explores three different methods for storing and querying Bitcoin data through relational databases: a local SQL database, through the cloud, and third-party web-based interfaces. [McGinn *et al.*, 2018] and [Spagnuolo *et al.*, 2014] tackle a similar task, but using Neo4j instead of SQL. [Han *et al.*, 2023] enables query processing of SQL SELECT operations for smart contracts in Ethereum-based blockchains by using an SQLite database to store smart contract transactions. These works highlight a desire to explore blockchain data in more detail, and the preference for using databases.

[Zhu *et al.*, 2020] introduces SEBDB, a blockchain database with a SQL-like query language that allows for CREATE, INSERT, and SELECT clauses to create a table, send a new transaction, and get query results, respectively. [Nathan *et al.*, 2019] proposes a *blockchain relational database* by developing a BC layer on top of the PostgreSQL DB. Both works apply the relational model to blockchain objects, similar to ours. However, we propose a tool that enables interoperability between existing technology already in use. In contrast, SEBDB offers an entirely new blockchain-based database solution and [Nathan *et al.*, 2019] an abstraction layer that enables blockchain-like behavior in a relational database.

Finally, [Marinho *et al.*, 2020] introduces MOON, a middleware that enables SQL queries to be executed on blockchain and relational databases. This work is based on MOON, and key changes include (1) support for DELETE and multi-valued INSERT operations, (2) support for mutable blockchain schema, (3) comprehensive handling of subqueries and aggregations in SELECT operations, (4) optimized latency during data fetching, and (5) an updated architecture with improved interoperability, integrity, and resilience. These changes encompass the 3 main qualities defined for interoperability in Section 4.1.

Table 1 compares our approach to similar works. We have categorized each work based on several criteria, such as the type of approach, query language, and blockchain querying support. Full blockchain support means it supports the full range of basic DML commands (SELECT, INSERT, UPDATE, DELETE) on the blockchain. Partial support allows interaction but not the full range of commands. This categorization shows how our approach fills a relatively unexplored niche.

Work	Query language	Blockchain Support	Approach
[LeFevre *et al.*, 2014]	SQL-like	No	Multistore
[Duggan *et al.*, 2015]	Mixed	No	Polystore
[Bondiombouy *et al.*, 2016]	SQL-like	No	Multistore
[Vogt *et al.*, 2018]	Mixed	No	Polystore
[Nathan *et al.*, 2019]	SQL	Partial	Database
[Singhal *et al.*, 2019]	Mixed	No	Polystore
[Zhu *et al.*, 2020]	SQL-like	Partial	Database
[Marinho *et al.*, 2020]	SQL	Partial	Middleware
Inter-MOON	SQL	Full	Middleware

Table 1. Comparison table of related works.

4 The Inter-MOON Approach

Our approach, denominated Inter-MOON, can be summarized as a middleware that allows for the execution of SQL DML queries to blockchain entities. Since blockchain has no standard query language, API is the most general way to execute queries. However, due to its adherence to immutability, most blockchain APIs will not offer any functions that can delete or update preexisting data.

Therefore, our approach is to map blockchain entities to the relational data model using temporary virtual tables by a process we call virtualization (See Sec. 4.4). A relational schema is defined for each blockchain entity so that assets can be easily differentiated and optimize the table-building step. Through the Inter-MOON interface, it is also possible to delete or update data by using lookup tables (called Index Tables. See Sec. 4.3) that reference blockchain hashes. This approach was chosen as it allows us to map all SQL DML operations to blockchain entities.

The Inter-MOON (Fig. 1) approach comprises three major parts: the middleware, the SQL DB, and the Blockchain DB. The middleware is further divided into several modules with separate functions. The Communicator accepts and forwards SQL queries to the Scheduler, which enqueues them for proper processing. The SQL Analyzer, Index Manager, Mapper, and Schema Configuration all function in tandem to extract information from SQL queries, track blockchain assets using Index Tables, and query and virtualize them into the relational environment.

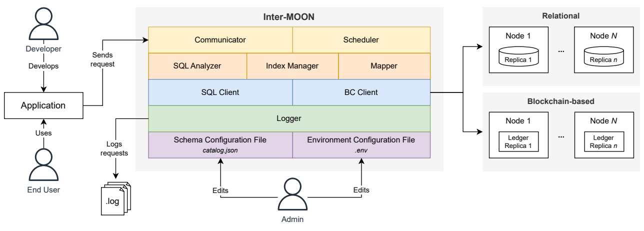


Figure 1. Overview of the Inter-MOON architecture.

This architecture is similar to MOON, with two major changes. The first is the removal of the Client Module. In MOON, it is a bridge present in the application used to send the query SQL and DB credentials over to MOON. In Inter-MOON, the Communicator directly receives SQL, and environmental configuration files hosted by the middleware are used to store credentials. This change is meant to eliminate the need to send sensitive information, like access keys, through the network alongside every request, as it is fair to assume that the middleware host will have ownership of said keys and that they will remain the same for every request.

The second is the addition of a Logger module to log write requests received by the middleware. Considering Inter-MOON's use of Index Tables, the Logger can utilize the relational database's own continuous archiving and recovery features to create regular backups and logs of the Index Table data. In the event of failure, data loss is minimized, and restoration of the blockchain indexes is facilitated, empowering data integrity and interoperability. This design also avoids introducing additional logging overhead while harnessing the data protection and recovery features inherent to the underlying relational database system.

Of note is that Inter-MOON is not a federation but a middleware that enables cross-querying blockchain-based and relational entities through SQL. Entities are kept separate in their respective data models, with no data replication. Like in polystores, the autonomy of each DB is preserved, and the granularity of each store is left untouched. For example, BigchainDB is document-based, using a local MongoDB instance to save transaction data.

4.1 Interoperability

In literature, the concept of interoperability is frequently divided into levels [Hasselbring, 2000]. In this work, we adopt a broad definition of interoperability, referring to it as the overall ability of a system to comprehend and engage with others. We consider two levels: Interoperability between Inter-MOON and its clients at the application level and Interoperability among the storage engines at the middleware level. Additionally, we identify three qualities that compose interoperability in the second level, which is the focus of this work:

- **Support** - The middleware can communicate with data storage engines.
- **Generality** - The middleware's capacity to understand and accurately map queries to their correct engine.
- **Efficiency** - The middleware's efficiency in finding and joining data in each storage engine.

In the proposed Inter-MOON architecture (Fig. 1), the Communicator demonstrates interoperability at the first level, allowing Inter-MOON to receive and answer client requests. For the second level, the SQL and Blockchain clients allow the middleware to interact with storage engines (Support), while the Mapper, SQL Analyzer, Index Manager, and Schema Configuration modules all work together as described above (Sec. 4), and in doing so fulfill Generality and Efficiency.

As mentioned, this paper focuses on interoperability at the middleware level. Inter-MOON improves interoperability in this context by increasing the number of supported DBs (Section 4.2), the quality and number of supported SQL commands (Section 4.4), and optimizing data retrieval (Section 4.3). Our approach is generally applicable as long as both the relational DB driver follows the Python DB-API interface and the blockchain-based DB exposes a basic API for storage and querying (See Sec. 4.2).

4.2 Support for Different Data Stores

To improve support, we must increase the number of data storage engines supported by the middleware and the quality of the offered support. For relational DBs, our approach is reminiscent of the *Django* [Holovaty and Kaplan-Moss, 2009] and *Laravel* [Stauffer, 2019] designs for multiple database engine support. In short, it is a generic database driver object, which contains an adapter implementing database access functions (Fig. 2). The generic driver structure is analogous to the popular decorator design pattern for software architecture, while the drivers are to the adapter pattern.

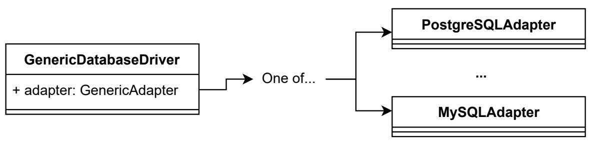


Figure 2. Simple rendition of the generic database driver.

The Inter-MOON middleware was developed using Python. Python's *DB-API*, a standard protocol for designing database access libraries, greatly optimizes the development of the generic driver. Listing 1 shows a basic pseudo-code implementation. As per Fig. 1, connection settings can be obtained from the environmental configuration. This structure promotes maintainability, decoupling, and database support, provided adapters are developed following Python's *DB-API* specification.

```
import psycopg2
class GenericDatabaseDriver:
    def __init__(self, adapter):
        self.adapter = adapter

    def connect(self, *args, **kwargs):
        return self
                .adapter
                .connect(*args, **kwargs)

driver = GenericDatabaseDriver(psycopg2)
with driver.connect("config.cfg") as conn:
    conn.execute("SELECT * FROM users;")
```

Listing 1: The Generic Database Driver basic structure. It holds an adapter object which represents the driver of a database engine.

As for blockchain, the lack of a unified data model is a current research issue [Meyer and dos Santos Mello, 2022; Yuan and Wang, 2018] that introduces a considerable challenge in creating a base interface for querying and accessing blockchain DB data in a similar way to the one specified above for relational DBs. Therefore, Inter-MOON considers blockchain assets to be similar to key-value pairs, where the key is the cryptographic hash and the value is the internal, immutable asset data. With this consideration, a similar approach can theoretically still be used, as long as the blockchain offers an API that exposes the following functions: (1) open a connection to the blockchain, (2) query one asset by hash, (3) query multiple assets using a list of hashes, and (4) send over assets for verification and appending. These comprise the basic blockchain operations that Inter-MOON utilizes.

4.3 Efficiency Optimizations

Inter-MOON utilizes what we call Index Tables (Fig. 3) to track the blockchain hash and the ID of each blockchain entity. These tables must be created by the Admin (Fig. 1) in the RDB, one for each expected blockchain entity. Because relational DBs implement efficient indexing mechanisms, they allow for inherently fast lookups of blockchain IDs. After Inter-MOON receives a request that involves blockchain entities, the middleware will query the Index Tables for the blockchain hashes of each entity. Then, it will fetch the asset data from the blockchain using that list of hashes.

MOON [Marinho *et al.*, 2020] did similarly. However, when fetching asset data from the blockchain, it did so, each asset one at a time. Consequently, the data-fetching process grew slower as the number of assets increased, following a non-linear growth curve. See Algorithm 1 for an overview.

lab_results
Blockchain Entity Index Table

lab_results_index	
bc_hash	**bc_id**
<hash>	758
...	...

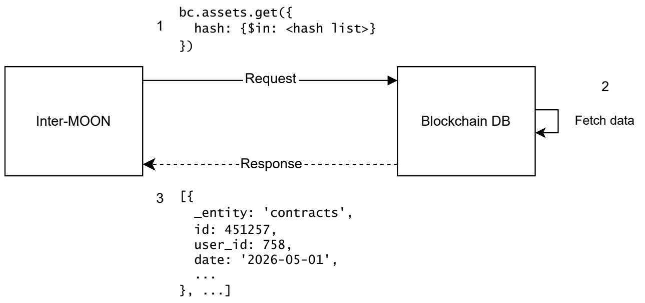
Figure 3. Index Table example for a *Lab Results* entity.

Algorithm 1 index searching algorithm in MOON.

```
Require: Set $I = \{i_1, i_2, i_3, ..., i_n\}$ of blockchain indexes, given $|I| > 0$.
Ensure: Set $A = \{a_1, a_2, a_3, ..., a_n\}$ of blockchain assets.
1: $A \leftarrow \emptyset$
2: $n \leftarrow 0$
3: $N \leftarrow |I|$
4: while $n < N$ do
5:     $i \leftarrow I(n)$
6:     $B \leftarrow$ get_asset_by_index($i$)
7:     $A \leftarrow A \cup B$
8:     $n \leftarrow n + 1$
9: end while
```

get_asset_by_index(i) represents a request sent to the blockchain network to fetch an asset of index i. As the number of requests increases, the network overhead present in each request accumulates, and response times grow. In a single network trip, there are 3 instances of present latency, L_{req}, L_{in}, and L_{res} for the request, in-network, and response latency respectively, totaling $L_{sum} = L_{req} + L_{in} + L_{res}$ for the total network latency produced by every usage of get_asset_by_index(i).

Inter-MOON, towards improving this behavior, optimizes L_{sum} by executing one trip only for each query. See Fig. 4 for a visual representation, and consider (1) to be L_{req}, (2) to be L_{in}, and (3) to be L_{res}.



```
bc.assets.get({
  hash: {$in: <hash list>}
})
```

```
[{
   _entity: 'contracts',
   id: 451257,
   user_id: 758,
   date: '2026-05-01',
   ...
}, ...]
```

Figure 4. Inter-MOON data fetching example.

This approach requires more computational power from the network and the Inter-MOON middleware host to process and return all necessary assets in one trip. Further optimization can be achieved by using batch-loading to retrieve an X number of indexes per trip, maintaining lower latency while reducing processing load. Suppose a set or range of primary keys is specified in the query. In that case, another optimization is to retrieve only the assets of those identifiers rather than all assets of each involved entity.

4.4 Generality of Query Processing

According to the *ISO/IEC 9075-1* specification, SQL-data statements can perform queries and insert, update, and delete information [Melton, 2016]. Consequently, towards interoperability on a querying level, the Inter-MOON middleware must be able to correctly interpret and reproduce SQL-data statements, or DML statements, in either system (blockchain or relational). We limit our support to queries written using standard SQL syntax and only containing one SQL statement per request. In this work, DDL commands are not considered. Alg. 2 shows our proposed querying algorithm for Inter-MOON.

Algorithm 2 Inter-MOON querying algorithm.

```
Require: SQL query string
Ensure: Processed query or blockchain data
1: $operation, entities \leftarrow$ tokenize(query)
2: $data \leftarrow \emptyset$
3: while $entity \in entities$ do
4:     if $entity$ is a blockchain asset then
5:         $data \leftarrow data \cup$ fetch_blockchain_data($entity$)
6:     end if
7: end while
8: if $data$ is empty then
9:     return Forward query to relational database
10: else
11:     $virtual\_data \leftarrow$ virtualize($data$)
12:     execute_operation($operation, virtual\_data$)
13: end if
```

In summary, we first (Line 1) extract the operation and entities involved in the given SQL query. Assuming at least one of said entities belongs to the blockchain (Line 4), we fetch the necessary data from it, execute the virtualization procedure, and then the operation. Our approach to each type of supported DML operation will be explained further in this section.

The first step is to extract and process information from received SQL queries. The SQL Analyzer module shown in the Inter-MOON architecture (Fig. 1) is responsible for this. It uses a tokenizer mechanism to observe statements as groups of tokens, such as keywords, identifiers, functions, or conditionals, assigned based on the token's semantic meaning inside of its group. This helps prevent ambiguity and allows easier handling of nested subqueries. For example, in the statement depicted in Fig. 5, conditionals can be found by searching for any groups beginning with a WHERE clause, while entities can be found by looking for any *Identifiers* after a FROM or JOIN keyword that is not a subquery. If a subquery is found instead, recursion can be used to go inside the subquery group to find nested Identifiers.

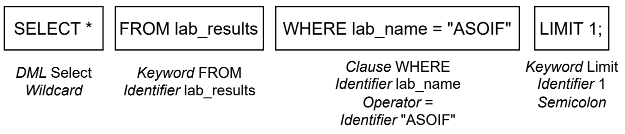
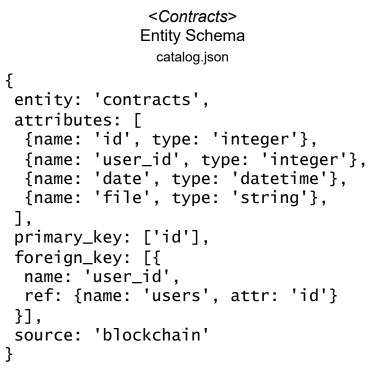


Figure 5. Inter-MOON SQL Analyzer tokenizer example.

In contrast, MOON used a simple lazy search algorithm to find the first matching M_s token (eg. *"SELECT"*) or character literal (eg. brackets, comma) of index i given a query string Q_s where $M_s = Q_s(i)$ or $M_s \subset Q_s$. A match was made when $M_s \neq \emptyset$. This was used, for example, to find the kind of operation being requested (SELECT, INSERT, UPDATE), entities, attributes, or the presence of conditionals. As such, Inter-MOON's technique is less error-prone and enables proper handling of subqueries.

As for virtualization, we define it as the process in which a blockchain asset is queried from the blockchain and instantiated into session-available temporary tables in the relational database. This way, they can interact as if they were relational objects. To support this technique, Inter-MOON expects the Admin to create a schema for each blockchain entity (Fig. 6). It will be used to track each entity and its attributes. Blockchain entities are virtualized in the RDB for blockchain read and write operations, and the blockchain API driver is also activated for write operations.


<Contracts>
Entity Schema
catalog.json

```
{
 entity: 'contracts',
 attributes: [
  {name: 'id', type: 'integer'},
  {name: 'user_id', type: 'integer'},
  {name: 'date', type: 'datetime'},
  {name: 'file', type: 'string'},
 ],
 primary_key: ['id'],
 foreign_key: [{
  name: 'user_id',
  ref: {name: 'users', attr: 'id'}
 }],
 source: 'blockchain'
}
```

Figure 6. Inter-MOON blockchain Schema example.

One contribution of Inter-MOON is that this schema can be mutable. While blockchain must offer immutability, it only concerns stored information, not necessarily the structure that any piece of information should have. To achieve this, the aforementioned virtual tables are built using both the attributes present in the queried assets and the schema. When an asset of a given blockchain entity contains an attribute not found in its schema, the attribute is ignored. When the opposite is true, the value of the missing attribute is set to NULL. This allows for the virtualization of assets using either a new or old version of the same schema, provided each attribute has a unique name. It also helps promote further interoperability by bringing the blockchain schema applied by Inter-MOON closer to the relational data model.

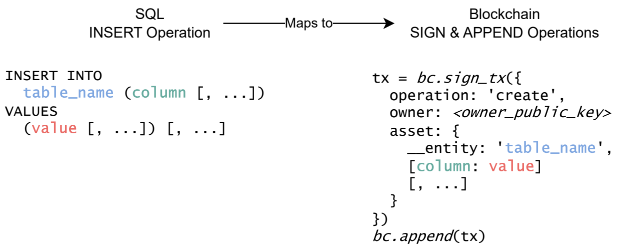
Upon receiving a query, Inter-MOON expects it to fall into one of the following scenarios: (1) SELECT, INSERT, UPDATE, or DELETE with only relational entities, (2) SELECT, INSERT, UPDATE, or DELETE with only blockchain entities and (3) SELECT with both blockchain and relational entities. For (1), Inter-MOON simply forwards the query to the RDB and sends back the response. For (2), there are separate approaches depending on the type of SQL statement, explained further below. The approach for (3) is similar to the one used in (2) for SELECT.

For SELECT in (2) and (3), Inter-MOON supports all common SQL tokens applicable to SELECT statements by always virtualizing needed blockchain entities in the RDB. This includes joins, aggregations, and subqueries. The query is then executed and results are returned as tuples. To see how Inter-MOON handles optimization, see Section 4.3.

For INSERT, while MOON only supports simple INSERT statements without subqueries, multi-valued INSERTs are also supported by Inter-MOON (See Lst. 2). We consider INSERT to be categorically equivalent to a blockchain APPEND, with multi-valued equivalent to several APPENDs. Attributes are extracted from SQL and used as the asset data, and inside the blockchain, data is stored in its given type, defined in the schema, to preserve integrity. See Fig. 7 for a visual rendition of this mapping technique.

```
INSERT INTO table_name [ AS alias ] [ (
    column_name [, ...] ) ]
    { VALUES ( { expression | DEFAULT } [, ...] )
     [, ...] }
```

Listing 2: Supported syntax for SQL INSERT.



```
INSERT INTO
  table_name (column [, ...])
VALUES
  (value [, ...]) [, ...]
```

```
tx = bc.sign_tx({
  operation: 'create',
  owner: <owner_public_key>
  asset: {
    __entity: 'table_name',
    [column: value]
    [, ...]
  }
})
bc.append(tx)
```

Figure 7. Inter-MOON blockchain mapping of an SQL INSERT operation.

Blockchains have inherent limitations when performing DELETE or UPDATE operations due to their append-only nature. For UPDATE operations, a new transaction is created with updated data and a pointer to the old asset. The expected syntax is similar to standard UPDATE (See Lst. 3).

```
UPDATE table_name [ * ] [ [ AS ] alias ]
    SET { column_name = { expression | DEFAULT }
    } [, ...]
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
```

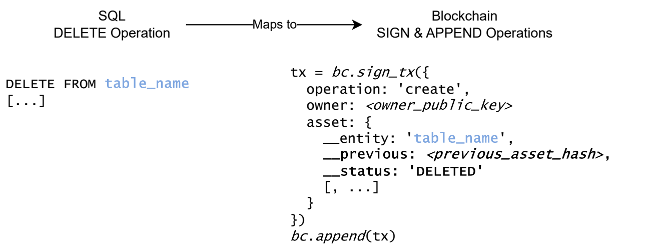
Listing 3: Supported syntax for SQL UPDATE.

As for DELETE, our proposal is a soft-delete mechanism in which the deleted blockchain asset's index is removed from the Index Tables, preventing retrieval through Inter-MOON. Conditionals may be used to refine the deletion criteria (See Lst. 4). To record this deletion within the blockchain context, we can append a new asset (like in UPDATE described above) with a *status: DELETED* label. This approach offers a pseudo-DELETE functionality while maintaining blockchain consensus and immutability (See Fig. 8).

```
DELETE FROM table_name [ [ AS ] alias ]
    [ WHERE condition ]
```

Listing 4: Supported syntax for SQL DELETE.

While these approaches preserve consensus mechanisms, scalability becomes a concern when dealing with many assets. The scalability issue is an ongoing research topic in blockchains [Zhou *et al.*, 2020]. Some studies aim to explore mutability in blockchains, which would further align them



```
DELETE FROM table_name
[...]
```

```
tx = bc.sign_tx({
  operation: 'create',
  owner: <owner_public_key>
  asset: {
    __entity: 'table_name',
    __previous: <previous_asset_hash>,
    __status: 'DELETED'
    [, ...]
  }
})
bc.append(tx)
```

Figure 8. Inter-MOON blockchain mapping of an SQL DELETE operation.

with relational systems [Politou *et al.*, 2019]. However, developing a new blockchain or blockchain-based technology is beyond the scope of this work.

5 Experiments

A prototype of the Inter-MOON middleware was developed using Python 3.6.9. Three experiments were prepared to test Inter-MOON regarding performance and support for the SQL querying syntax.

5.1 Comparing the performance of MOON & Inter-MOON

The first experiment aimed to compare the performance of MOON and Inter-MOON, focusing on response speed, measured by the round-trip time from client request to response. A synthetic dataset was generated with a *Patients* entity stored on an RDB and Lab Results on the blockchain DB. (Fig. 9). The testing environment consisted of Ubuntu 18.04.6 VMs on a local network (Fig. 10). Table 2 describes each VM in detail.. VM-1 hosted instances of both MOON and Inter-MOON, running one at a time. VM-2 utilized Postgres 10.23 for SQL, and BigchainDB 2.2 for blockchain nodes. A separate machine with Ubuntu 22.04, 4 GB RAM, and an Intel i5-4300 2.60 GHz CPU simulated the client.

Lab Results (100 rows)	
`varchar`	uid
`integer`	patient_id
`varchar`	content_base64
`date`	datetime
`varchar`	lab_name
`integer`	lab_site
`integer`	expired

Patients (100 rows)	
`integer`	id
`varchar`	name
`varchar`	email
`varchar`	phone
`date`	birth_date

Figure 9. Entity schema used for the first experiment.

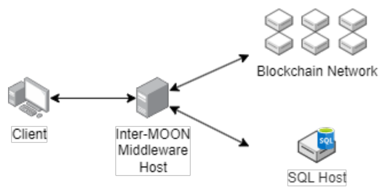


Figure 10. Testing environment.

Four queries (Table 3) were executed on MOON first and then on Inter-MOON. Inter-MOON was expected to provide significantly improved response speeds in queries involving many entities (Q2 and Q3) while maintaining similar but slightly faster speeds in other kinds.

Name	Role	RAM	Disk Read/Write Speed (GB p/ second)
VM-1	Middleware host	4 GB	7.5/0.8
VM-2	SQL database host	2 GB	6/0.8
VM-3 $\sim$ 8	BC network nodes	1 GB/each	5/0.4

Table 2. Summary of the virtual machines used in the first experiment.

Query	SQL
Q1	INSERT INTO lab_results (`<...columns>`) VALUES (`<...values>`);
Q2	SELECT * FROM lab_results;
Q3	SELECT * FROM lab_results JOIN patients ON lab_results.patient_id = patients.id;
Q4	UPDATE lab_results SET expired = 1 WHERE uid = `<uid>`;

Table 3. Set of queries used in the first experiment.

Results (Fig. 11) show that Inter-MOON was generally much faster. In Q1, the results were in the same ballpark. In Q2 and Q3, they were about 10 times higher. In Q4, there was an improvement of about 5.5 times instead. UPDATE-type transactions, which is the case for Q4, are more computationally expensive and latency-inducing, as they involve several trips to both database systems in order to read, update, and write the updated information.

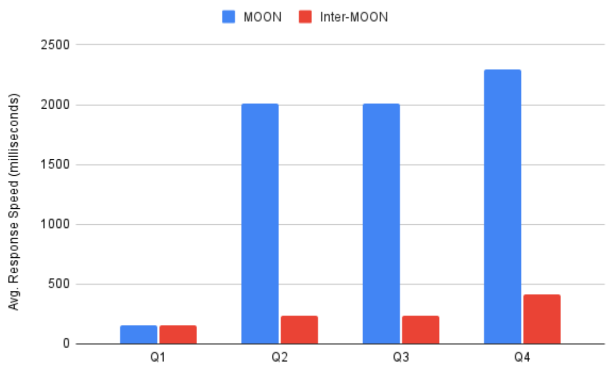


Figure 11. Graphical comparison of the Avg. Response Speed of 100 query executions between MOON and Inter-MOON.

5.2 SQL Syntax Support

In the second experiment, the goal was to evaluate Inter-MOON regarding SQL syntax support in read operations to illustrate the viability of our approach. The TPC-H[2] decision support benchmark was used for this experiment, as it is an industry-tested standard with various queries that showcase critical business needs. We also compared Inter-MOON against the BigDAWG polystore to show how a similar tool fares in this regard. BigDAWG was chosen as it is one of the few available open-source tools that show similar characteristics to our work: a unified interface with support for SQL and NoSQL data stores that can be simultaneously queried using a single request. Both systems were populated with a 1 GB scale factor workload of the benchmark data and then 20

[2]`https://www.tpc.org/tpch/`. Accessed: Feb 27, 2024.

of its 22 queries were executed using a simple Python script. Execution results were compared against the expected output given by the benchmark. Two were ignored as they contained no unique keywords or operations not found in other queries and took too much time to run, which obstructed testing, considering execution speed was not a metric in this experiment. The metric was simply the factor of successful queries over the total number tested: $SupportScore(S) = Q_{success}/20$. Successful, in this case, means the query was properly executed, and the results matched the expected output.

Inter-MOON was capable of running 19 queries, attaining a score of 0.95, an improvement over the result presented in the original work of 0.9 with 18 queries [Sá *et al.*, 2023]. BigDAWG successfully ran 6, scoring 0.3 (Table 4). Both systems failed to run a query that created and then utilized a view. BigDAWG, additionally, demonstrated issues when executing queries containing a mix of nested subqueries, aggregation, and sorting. Results indicate that BigDAWG supports only a small subset of SQL, while Inter-MOON could understand much of the standard syntax.

System	Successful queries	S Score
Inter-MOON	19	0.95
BigDAWG	6	0.3

Table 4. Support score for Inter-MOON and BigDAWG.

5.3 Cross-model Query Performance

The last experiment evaluated Inter-MOON's cross-model querying performance in a realistic scenario where requests were continuously sent to the middleware for processing. Once again, the BigDAWG polystore was chosen for comparison. BigDAWG polystore was used for comparison; however, since BigDAWG does not support blockchain, blockchain entities were stored in Accumulo, a key-value NoSQL data store, while Inter-MOON stored them in BigchainDB. The environment used Docker container setups provided by the BigDAWG project[3] and a custom-built setup for Inter-MOON, with separate containers for Inter-MOON, PostgreSQL 9.6, and BigchainDB 2.2. Both setups were run on the same machine used in the first experiment (See Subsection 5.1), although only one setup was active at a time.

A supermarket sales history dataset[4] was inserted into both BigchainDB (Inter-MOON) and Accumulo (BigDAWG), with synthetic customer data inserted into the SQL databases. JMeter 5.5[5], running on a separate machine, monitored and executed the test plan. This plan involved executing a simple JOIN query (Table 5) using data from both data models, with 1000 threads and a ramp-up time of 1000 seconds, simulating a constant stream of transactions. The evaluated metrics were average latency, standard deviation, and failure count.

Table 6 shows the mean, median, and standard deviation of the recorded latency of both Inter-MOON and BigDAWG

Customers (1000 rows)	
`integer`	c_id
`varchar`	c_name
`varchar`	c_email
`varchar`	c_gender
`varchar`	c_phone
`date`	c_birth_date
`varchar`	c_type

Sales (1000 rows)	
`integer`	s_id
`decimal`	s_unit_price
`integer`	c_id
...	

Figure 12. Entity schema used for the third experiment. Only the relevant attributes from the *Sales* entity are being shown here.

```
SELECT c_name, s_unit_price
FROM customers c JOIN sales s ON c.c_id = s.c_id
ORDER BY s.s_unit_price DESC LIMIT 10;
```

Table 5. Query used in the third experiment.

considering the whole workload, while Fig. 13 shows a combined histogram. We can see that neither system provides results that seem to follow a normal distribution. For BigDAWG in particular, we note a very unbalanced distribution with high deviation.

System	Mean Latency	Median Latency	Std. Deviation
Inter-MOON	892.48	913.0	94.25
BigDAWG	892.45	574.0	1187.87

Table 6. Mean, median, and std. deviation for latency in both systems. Values are rounded down to two decimal places.

Both Inter-MOON and BigDAWG demonstrated difficulty in reaching higher throughput. Inter-MOON (Fig. 14) provided an overall consistent latency of 800-890 ms, with zero errors. BigDAWG (Fig. 15) provided lower average latency but much higher deviation and a total of 45% failure rate, slightly below half of all queries. As the flood continues, BigDAWG accumulates failures (Yellow line) in cascade, which heavily impacts latency. After some additional testing with lower ramp-up time, both tools show much worse performance, with BigDAWG increasing the failure rate even more and Inter-MOON showing exponentially higher latency. This indicates the existence of concurrency issues when obtaining data from separate data models simultaneously. However, more testing needs to be done to confirm this issue. In conclusion, while Inter-MOON showed better reliability than BigDAWG and stable performance, we cannot fully claim it is better or worse than BigDAWG regarding performance in this scenario.

6 Conclusion

In this work, we detailed our approach to providing interoperability between relational databases and blockchains by developing Inter-MOON, an extension of MOON. Experimental results showed that Inter-MOON provided average response times of 5 to 10 times faster than MOON in most tested queries, representing common SQL DML operations. Along with increased performance, DELETE operations and nested queries and aggregations are now fully supported. Data integrity is also enhanced, alongside database support with the generic database driver, which allows Inter-MOON

[3]`https://github.com/bigdawg-istc/bigdawg`. Accessed: Feb 27, 2024.
[4]`https://www.kaggle.com/datasets/aungpyaeap/supermarket-sales`. Accessed: Feb 27, 2024.
[5]`https://jmeter.apache.org/`. Accessed: Feb 27, 2024.

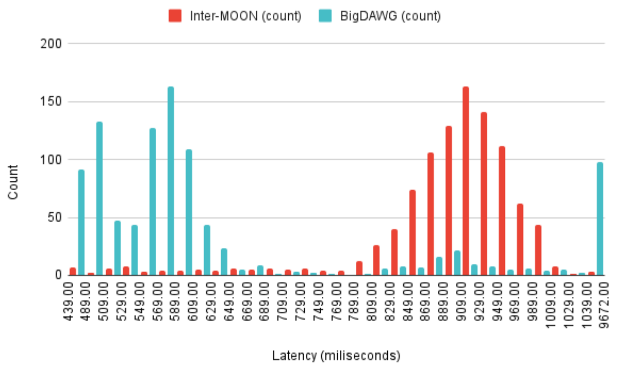


Figure 13. Histogram comparison of query latency from Inter-MOON and BigDAWG. This graph considers a bucket size of 20 and a 5% outlier rate.

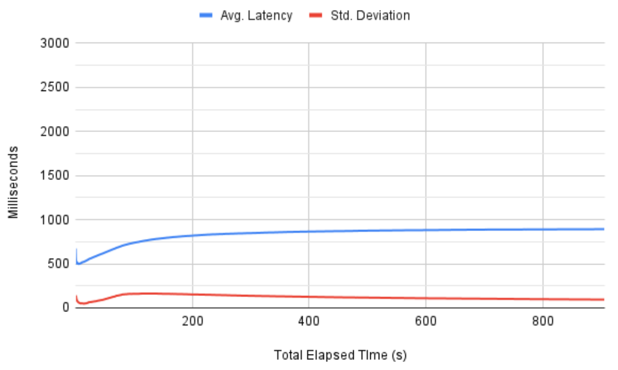


Figure 14. Inter-MOON Avg. Latency and Std. Deviation over time.

to use many different RDBs. Finally, Inter-MOON surpassed BigDAWG in SQL syntax support and provided more consistent performance with fewer errors, although due to the large number of errors BigDAWG showed during testing, it cannot be said that Inter-MOON is either better or worse than BigDAWG concerning performance.

For future works regarding the interoperability of blockchain and relational models, blockchain support for SQL DDL statements and other remaining commands not covered by Inter-MOON, such as DTL (BEGIN, COMMIT, and ROLLBACK), functions, and stored procedures, could prove fruitful. Another viable approach is to explore NoSQL alongside relational and blockchain, as blockchain can be considered similar to NoSQL as a data model. Finally, the scalability issues plaguing blockchain, discussed briefly in this work, still present a great obstacle to interoperability and integration.

Acknowledgements

I am grateful to LSBD (Laboratório de Sistemas e Bancos de Dados) for offering funding and crucial resources to enable this research, as well as to my colleagues and professors for their invaluable guidance and support.

Availability of data and materials

A prototype of the proposed approach for Inter-MOON was developed and published to a GitHub repository[6]. The datasets generated via scripts can also be found in the linked repository. Any external datasets are also linked via footnotes when mentioned.

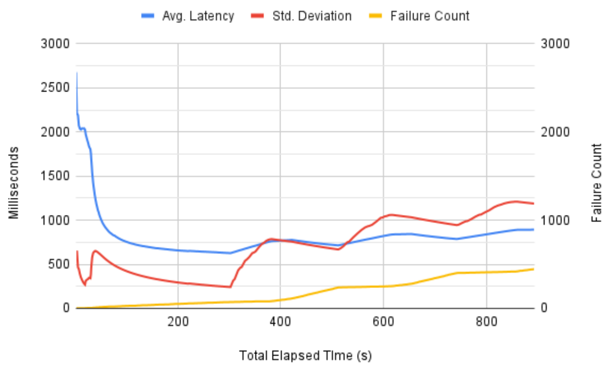


Figure 15. BigDAWG Avg. Latency, Std. Deviation and Failure Count over time.

References

Babcock, B., Babu, S., Datar, M., Motwani, R., and Widom, J. (2002). Models and issues in data stream systems. In *Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, page 1–16, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/543613.543615.

Belchior, R., Vasconcelos, A., Guerreiro, S., and Correia, M. (2021). A survey on blockchain interoperability: Past, present, and future trends. *Acm Computing Surveys (CSUR)*, 54(8):1–41.

Bigchain, D. and Gmb, H. (2018). Bigchaindb 2.0: The blockchain database. white paper.

Bondiombouy, C., Kolev, B., Levchenko, O., and Valduriez, P. (2016). Multistore big data integration with cloudmd-sql. *Transactions on Large-Scale Data-and Knowledge-Centered Systems XXVIII: Special Issue on Database-and Expert-Systems Applications*, pages 48–74.

Duggan, J., Elmore, A. J., Stonebraker, M., Balazinska, M., Howe, B., Kepner, J., Madden, S., Maier, D., Mattson, T., and Zdonik, S. (2015). The bigdawg polystore system. *ACM Sigmod Record*, 44(2):11–16.

Gadekallu, T. R., Huynh-The, T., Wang, W., Yenduri, G., Ranaweera, P., Pham, Q.-V., da Costa, D. B., and Liyanage, M. (2022). Blockchain for the metaverse: A review. *arXiv preprint arXiv:2203.09738*.

Gamage, H., Weerasinghe, H., and Dias, N. (2020). A survey on blockchain technology concepts, applications, and issues. *SN Computer Science*, 1:1–15.

Gervais, A., Karame, G. O., Wüst, K., Glykantzis, V., Ritzdorf, H., and Capkun, S. (2016). On the security and performance of proof of work blockchains. CCS '16, page 3–16, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/2976749.2978341.

Guo, H. and Yu, X. (2022). A survey on blockchain technology and its security. *Blockchain: Re-*

[6]`https://github.com/rafero1/inter-moon`. Accessed: Feb 27, 2024

search and Applications, 3(2):100067. DOI: https://doi.org/10.1016/j.bcra.2022.100067.

Han, J., Seo, Y., Lee, S., Kim, S., and Son, Y. (2023). Design and implementation of enabling sql–query processing for ethereum-based blockchain systems. *Electronics*, 12(20):4317.

Hasselbring, W. (2000). Information system integration. *Communications of the ACM*, 43(6):32–38.

Holovaty, A. and Kaplan-Moss, J. (2009). *The definitive guide to Django: Web development done right*. Apress.

Javaid, M., Haleem, A., Pratap Singh, R., Khan, S., and Suman, R. (2021). Blockchain technology applications for industry 4.0: A literature-based review. *Blockchain: Research and Applications*, 2(4):100027. DOI: https://doi.org/10.1016/j.bcra.2021.100027.

Krichen, M., Ammi, M., Mihoub, A., and Almutiq, M. (2022). Blockchain for modern applications: A survey. *Sensors*, 22(14). DOI: 10.3390/s22145274.

LeFevre, J., Sankaranarayanan, J., Hacigumus, H., Tatemura, J., Polyzotis, N., and Carey, M. J. (2014). Miso: Souping up big data query processing with a multistore system. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD ’14, page 1591–1602, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/2588555.2588568.

Maciel, R. S., Valle, P. H., Santos, K. S., and Nakagawa, E. Y. (2023). Systems interoperability types: A tertiary study. *arXiv preprint arXiv:2310.19999*.

Marinho, S. C., Costa Filho, J. S., Moreira, L. O., and Machado, J. C. (2020). Using a hybrid approach to data management in relational database and blockchain: A case study on the e-health domain. In *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pages 114–121. IEEE.

McConaghy, T., Marques, R., Müller, A., De Jonghe, D., McConaghy, T., McMullen, G., Henderson, R., Bellemare, S., and Granzotto, A. (2016). Bigchaindb: a scalable blockchain database. *white paper, BigChainDB*, pages 53–72.

McGinn, D., McIlwraith, D., and Guo, Y. (2018). Towards open data blockchain analytics: a bitcoin perspective. *Royal Society open science*, 5(8):180298.

Melton, J. (2016). Iso/iec 9075-1 information technology-database languages-sql-part 1: Framework (sql/framework). *ISO/IEC*, 2016(E):9075–1.

Meyer, J. V. and dos Santos Mello, R. (2022). An analysis of data modelling for blockchain. In *Information Integration and Web Intelligence: 24th International Conference, iiWAS 2022, Virtual Event, November 28–30, 2022, Proceedings*, pages 31–44. Springer.

Muzammal, M., Qu, Q., and Nasrulin, B. (2019). Renovating blockchain with distributed databases: An open source system. *Future generation computer systems*, 90:105–117.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260.

Nathan, S., Govindarajan, C., Saraf, A., Sethi, M., and Jayachandran, P. (2019). Blockchain meets database: Design and implementation of a blockchain relational database. *arXiv preprint arXiv:1903.01919*.

Politou, E., Casino, F., Alepis, E., and Patsakis, C. (2019). Blockchain mutability: Challenges and proposed solutions. *IEEE Transactions on Emerging Topics in Computing*, 9(4):1972–1986.

Sá, R. A., Moreira, L. O., and Machado, J. C. (2023). Improving interoperability between relational and blockchain-based database systems: A middleware approach. In *Anais do XXXVIII Simpósio Brasileiro de Bancos de Dados*, pages 115–127. SBC.

Schuhknecht, F. M., Sharma, A., Dittrich, J., and Agrawal, D. (2021). chainifydb: How to get rid of your blockchain and use your dbms instead. In *CIDR*.

Singhal, R., Zhang, N., Nardi, L., Shahbaz, M., and Olukotun, K. (2019). Polystore++: accelerated polystore system for heterogeneous workloads. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 1641–1651. IEEE.

Spagnuolo, M., Maggi, F., and Zanero, S. (2014). Bitiodine: Extracting intelligence from the bitcoin network. In *Financial Cryptography and Data Security: 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers 18*, pages 457–468. Springer.

Stauffer, M. (2019). *Laravel: Up & running: A framework for building modern PHP apps*. O’Reilly Media.

Stonebraker, M. and Cetintemel, U. (2018). *“One Size Fits All”: An Idea Whose Time Has Come and Gone*, page 441–462. Association for Computing Machinery and Morgan & Claypool.

Tseng, L., Yao, X., Otoum, S., Aloqaily, M., and Jararweh, Y. (2020). Blockchain-based database in an iot environment: challenges, opportunities, and analysis. *Cluster Computing*, 23:2151–2165.

Vogt, M., Stiemer, A., and Schuldt, H. (2018). Polypheny-db: towards a distributed and self-adaptive polystore. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 3364–3373. IEEE.

Yuan, Y. and Wang, F.-Y. (2018). Blockchain and cryptocurrencies: Model, techniques, and applications. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 48(9):1421–1428. DOI: 10.1109/TSMC.2018.2854904.

Yue, K.-B., Chandrasekar, K., and Gullapalli, H. (2019). Storing and querying blockchain using sql databases. *Information Systems Education Journal*, 17(4):24.

Zheng, Z., Xie, S., Dai, H.-N., Chen, X., and Wang, H. (2018). Blockchain challenges and opportunities: A survey. *International Journal of Web and Grid Services*, 14(4):352–375.

Zhou, Q., Huang, H., Zheng, Z., and Bian, J. (2020). Solutions to scalability of blockchain: A survey. *Ieee Access*, 8:16440–16455.

Zhu, Y., Zhang, Z., Jin, C., Zhou, A., Qin, G., and Yang, Y. (2020). Towards rich qery blockchain database. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 3497–3500.