JOURNAL OF
EMERGING INVESTIGATORS

Article

# Blockchain Databases: Encrypted for Efficient and Secure NoSQL Key-Store

**Ayushi Mehrotra[1], David Kim[1]**
[1] Troy High School, Fullerton, California

## SUMMARY

**Databases have revealed many vulnerabilities in which hackers can penetrate and gain access to the data. Blockchains have been proven useful to protect against fraud and data miners, exemplified in cryptocurrencies. In this paper, we propose a blockchain database framework as a non-relational database structure that integrates the properties of blockchains and databases. The independent variables when testing this database were the Append, Delete, Query, and Update functions in the database framework. We predicted that the Append, Query, and Update functions would have a logarithmic runtime and linear throughput; while the Delete function would have a constant runtime and throughput. The database was tested with about 150,000 operations with the server and client locally running. We observed the runtime was slightly positively correlated while the Delete was not positively correlated with the number of operations in the Append, Query, and Update functions. The throughput of the Append, Query, and Update functions were significantly negatively correlated with the number of operations. The blockchain database framework displayed end-to-end encryption and feasibility with this experiment. One possible application of the database framework is in the financial industry, as there is a one-to-one correspondence with user and transaction.**

## INTRODUCTION

A blockchain is a distributed, shared ledger to keep unalterable records across computers. Blockchains have recently gain attention from research and industry through the rise of cryptocurrencies. Since the introduction of cryptocurrencies, the qualities of immutability and persistency attracted the implementation of blockchains in other technologies (1–2). One of these technologies includes non-Structured Query Language (NoSQL) databases, which store data as documents, key-stores, and graphs (3). Comparing to their relational database cousins, which store data based on relationships between data, NoSQL databases are non-relational and store data in different structures other than tables. The main advantages of these types of databases are the improvements in scalability and a data-structure-based database (4). Combining databases and blockchains would retain the security features of blockchains (5).

Many researchers have explored the area of blockchain databases. Peng et al. and El-Hindi et al. implemented blockchain databases with network security in mind and allowed the users to securely collaborate in a blockchain database (6–7). Nathan et al. integrated relational databases, which store data with pre-defined relationships in tables with blockchains (8). BigchainDB is a commercial blockchain database, which uses MongoDB as its data stores (9–10). Adkins et al. derived three constructions of encrypted blockchain databases to implement in blockchains such as Ethereum and Algorand (11).

This work expands on the idea of encrypted blockchain databases, where the user encrypts the sensitive data first, then uses the secret key to query the data (11). The database allows end-to-end encryption, where the data is encrypted in both the server and in transit to the user's device, then decrypted on the user's device (11). This is a significant difference in current commercial NoSQL databases such as Cassandra and MongoDB, where the data is unencrypted at-rest (12).

Adkins et al. addressed this problem with encrypted multi-maps, which are key-stores that support the Get and Put operations on the data (11). While proving its use in other public blockchains is a significant step towards practicability, this work implements its own blockchain and construction to test the feasibility of a database.

First, our work provides an in-depth security analysis on the blockchain framework described. To test the feasibility of the database, we tested the runtime and throughput. The throughput is the number of operations per second. Usually, the latency, throughput, and runtime are tested against common databases (13). As the server and client are both run locally, the latency is not tested. The independent variables of this study are Append, Query, Update, and Delete algorithms of the blockchain database. There are four hypotheses, each focusing on one of the independent variables.

We hypothesized that if the Append method was called, then the runtime would have a positive linear correlation and throughput a negative linear correlation because the number of blocks in the database increases the height logarithmically, therefore the runtime would increase, and throughput decrease.

Additionally, we proposed that if the Query method was called, then the runtime would be logarithmic and throughput

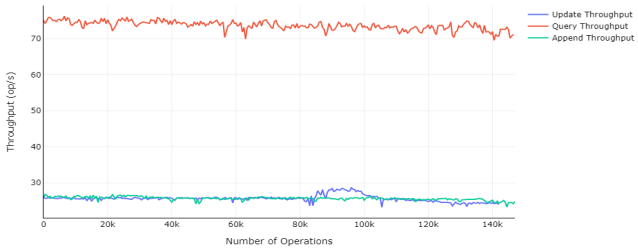


**Figure 1. Throughput Performance Comparison of Database Functions.** Line graph showing throughput of the Update, Query, and Append functions. To reduce sampling variability, the moving average of the past 500 trials was recorded and each data point is the average of 5 trials.


**Figure 2. Runtime Performance Comparison of Database Functions.** Line graph showing runtimes of the Update, Query, Append, and Delete functions. To reduce sampling variability, the moving average of the past 500 trials was recorded and each data point is the average of 5 trials.

linearly negative because since the database structure is a hybrid AVL tree, the runtime is reduced to logarithmic trend, so as runtime increases, the number of operations per second decreases.

We theorized that if the Update method was called, then the runtime would be logarithmic and throughput linearly negative, which is due to the construction of the Update method as it calls the Delete method, then Append method, adding to a logarithmic runtime and decreased throughput.

Finally, we hypothesized that if the Delete method was called, then the runtime would be constant and throughput would remain unchanged, as Python is the language of choice, deletion of an element in a dictionary is constant runtime and constant throughput (15).

The purpose of this work is to combine and advance the field of blockchain databases and explore the security advantages. The blockchain database framework has proved its feasibility and security. The throughput and runtime both are aligned with the hypotheses, and this blockchain database framework can be implemented in the financial world, where one-to-one correspondence is needed.

## RESULTS

We tested the throughput and runtime on the blockchain database using a script that models Yahoo! Cloud Serving Benchmark (YCSB). Latency, which is usually coupled with throughput, was not tested for the blockchain database as the server and client were run locally. YCSB is routinely used to benchmark Structured Query Language (SQL) databases, so we tailored a script to perform the same functions for the blockchain database (17). We measured the performance of the database framework to determine if it was feasible. The script inputted test data with the size of 60 bytes into the database framework. It also conducted 143,640 operations of Append, Delete, Update, and Query functions each. The large amount of data and operations ensured that the database framework can be maintained. The script ran five times to ensure the computer and environment did not affect the results of the database.

**Table 1. Throughput Correlation with Number of Operations.** A Pearson correlation test was used to determine the number of operations and throughput of Append, Query, and Update functions (n=143640).

| Function | Correlation | Significance | p-value |
|---|---|---|---|
| Append | negative | significant | <0.0001 |
| Query | negative | significant | <0.0001 |
| Update | negative | significant | <0.0001 |

### Throughput

The throughput is the average of five trials for each database function. The Append, Query, and Update functions showed a negative linear trend; however, the trend was not strong enough to be deemed as a strongly negative correlation (p=0.0; **Figure 1**). Yet, it shows that the Append, Query, and Update functions decrease slightly as the number of operations increase, which is significant (**Table 1**). The Delete function was excluded from the analysis as all 143,640 operations were completed in under a second. The Query function throughput was noticeably higher than the Append and Update function.

**Table 2. Runtime Correlation with Number of Operations.** A Pearson correlation test was performed to establish the number of operations and runtime of Append, Query, Update, and Delete functions (n=143640).

| Function | Correlation | Significance | p-value |
|---|---|---|---|
| Append | positive | significant | <0.0001 |
| Delete | positive | not significant | >0.80 |
| Query | positive | significant | <0.0001 |
| Update | positive | significant | <0.0001 |

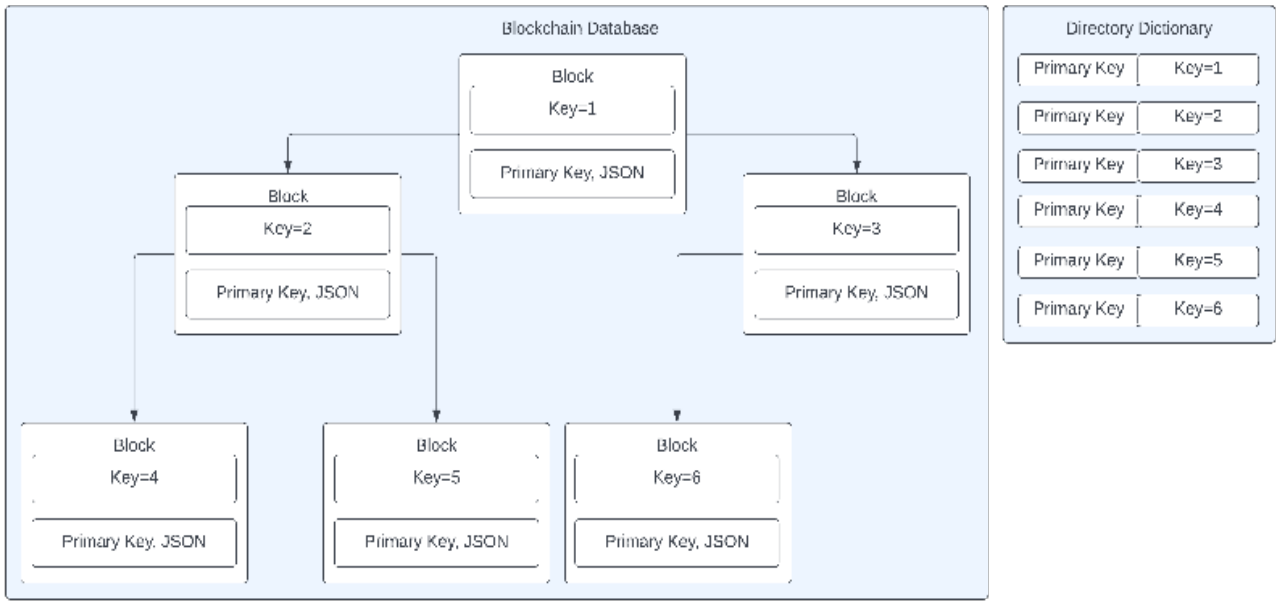


**Figure 3. Structure of blockchain database with directory dictionary.** Diagram of each block pointing to the previous block. To query, the server is directed towards the directory dictionary to get the key to the block. The key acts as a directory when converted to binary, as 0 is towards the right node and 1 is towards the left node.

### Runtime

The runtime, which is the difference between wall time from the start of the function to the end, is defined by the average of five trials of each database function. The Append function runtime was noticeably higher than the Query, Update, and Delete function, while the Query and Update functions had similar runtimes (**Figure 2**). All four functions showed a slightly positive trend, yet the correlation between runtime and number of operations for Append, Query, and Update functions were significant, the Delete function correlation was not significant ($p>0.80$; **Table 2**).

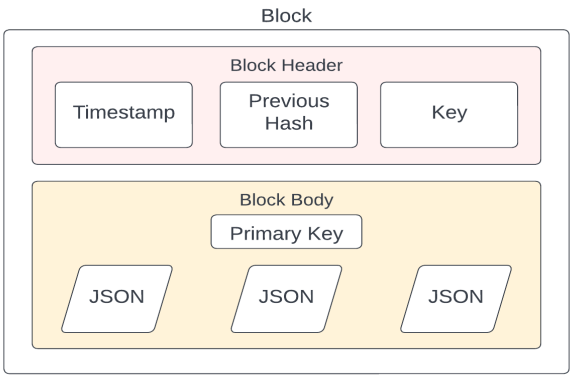


**Figure 4. Components of a block.** Diagram showing each component of a block in the blockchai n database. The primary key is hashed with SHA256() and the JSON files are encrypted with AES-CTR. The key for AES-CTR is transferred between server and client by ECDH.

## DISCUSSION

The runtime of the functions remained relatively stable, with a slight positive increase in the Append, Query, and Update functions. These results support our hypotheses as the Append, Query, and Update functions have linear trends. The Delete function also showed a positive correlation, yet not significant, which supports the hypothesis regarding the Delete function, where the runtime and throughput would be constant.

The throughput of the Append, Query, and Update functions displayed a weak negative correlation, which supports our hypotheses. As the Delete function throughput was constant, it supports our hypothesis too. The throughput was low, yet this could have been influenced by the computer specifications of the Surface 3 Laptop. The server and client were ran locally on a personal computer, whereas databases are usually stored in large data centers. For the scope of this paper, using a personal computer was sufficient to test the blockchain database framework. The negative tendency could have been corrected by adding more threads to the database. Threads allow another instance of the database to run, which could increase the throughput exponentially.

The throughput of the Append and Update function was significantly less than the Query function, which could have been due to the creation of a block in the database. All three of the functions have the same time complexity, which is

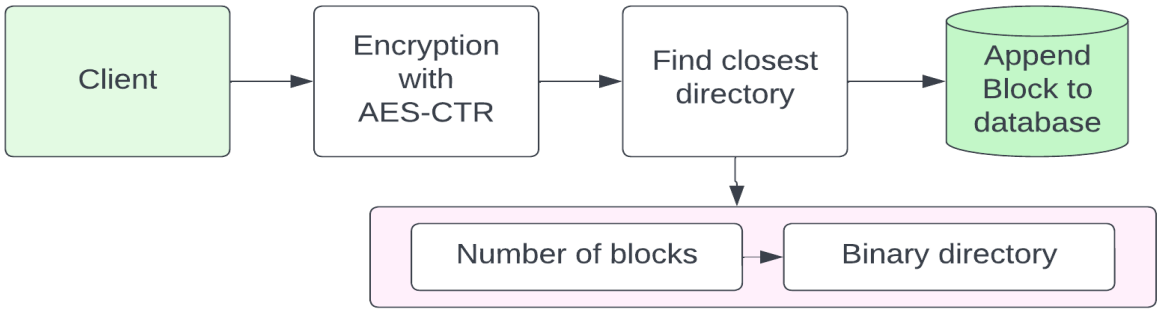


**Figure 5. Flowchart of an Append function.** The JSON that the client inputs is encrypted with AES-CTR, which is transferred by ECDH to decrypt while querying. The primary key is hashed with SHA256(SHA256()) and the EMM is stored in the block. The location of the block in the blockchain database is obtained by the closet directory, which is essentially converting the number of blocks in the blockchain to a binary directory. The server uses the binary directory to get to the closest directory and appends the block with the EMM and hashed primary key.

outlined below, so the lower throughput was most likely due to the new block that is appended to the database framework.

Qualitatively, the time complexity (the amount of time taken by an algorithm to run as a function of the input) of the four functions are listed below.

**Equation 1.1.** Append function Time Complexity: $O(\log_2 n+k)$.

Where $n$ is the number of blocks and $k$ is the overhead in creating a block. Let $V$ be the length of characters of $n_2$ – 1, where $n_2$ is n in base 2, and the Equation (Eq.) 1.1 be $O(V+k)$. $V$ can be approximated as $\log_2 n+1$ empirically, so Eq. 1.1 would result to $O(\log_2 n+1+k)$. After disregarding the constants and converting to conventional standards, the time complexity of the Append function will be $O(\log_2 n+k)$.

**Equation 1.2.** Delete function Time Complexity: $O(1)$

Eq. 1.2 has constant time complexity since it removes the directory of a block in the directory dictionary.

**Equation 1.3.** Query function Time Complexity: $O(\log_2 n)$

Where n is the number of blocks and k is the overhead in creating a block. Let V be the length of characters of $n_2$ – 1, where $n_2$ is $n$ in base 2, and the Eq. 1.3 be $O(V)$. $V$ can be approximated as $\log_2 n+1$ empirically, so Eq. 1.3 would result as $O(\log_2 n+1)$. After disregarding the constants and converting to conventional standards, the time complexity of the Query function will be $O(\log_2 n)$.

**Equation 1.4.** Update function Time Complexity: $O(\log_2 n+k)$

Where n is the number of blocks and k is the overhead in creating a block. Let V be the length of characters of $n_2$ – 1, where $n_2$ is $n$ in base 2, and the Eq. 1.4 be $O(V+k)$. The extra constant is due to the deletion of a block to replace it with an updated one. $V$ can be approximated as $\log_2 n+1$ empirically, so Eq. 1.4 would result as $O(\log_2 n+2+k)$. After disregarding the constants and converting to conventional standards, the time complexity of the Update function will be $O(\log_2 n+k)$.

In the future, we would like to continue to scale this blockchain database framework to a fully working database with an Application Programming Interface (API). We would also like to test out the database with more than one user to implement the decentralized aspect of a blockchain.

Our paper presents a novel encrypted blockchain database framework and implements its own blockchain to test feasibility. The blockchain database framework was tested for runtime and throughput with about 150,000 operations. Overall, the runtime has no significant positive correlation with the number of operations and the throughput has a slight negative correlation with the number of operations. With the security of an encrypted database and the results from this paper that supports scalability, the architecture presented in this paper is feasible in the real world. Both metrics show promising future for a larger database, which includes load balancing, an API, and consistent data partitioning.

## MATERIALS AND METHODS

To start the procedure of this research, the specifications of the computer was reviewed to ensure the following requirements were meet: Windows 11 (Version 21H2), PyCharm (Version 1.2), and Python Interpreter (Version 3.9). The framework was implemented on a Surface Laptop 3 with an Intel i7 CPU at 1.30 GHz and 16 GB RAM. The blockchain database was built in the Integrated Development Environment (IDE) and exported to GitHub. Using the Append, Query, Delete, and Update functions presented in the code, the throughput and runtime of the functions were measured. To test the database, a script was created to model the Yahoo! Cloud Serving Benchmark (YCSB) program and transferred all the output metrics to a CSV file. To test the significance of the data as the number of operations grow, a Pearson correlation test was used.

### Encryption Scheme

The encryption used in this database framework was the

Elliptic Curve Diffie-Hellman Key Exchange (ECDH) with Advanced Encryption Standard in counter mode (AES-CTR). Elliptic curve cryptography was chosen for the key exchange between the server and the client because it is not as space consuming as Rivest-Shamir-Adleman (RSA). Diffie-Hellman Key Exchange is an algorithm that facilitates the communication between the server and the client. ECDH uses the points on the elliptic curve to designate the private and public key of both client and server. To encrypt the JSON files, AES-CTR was used to provide secure symmetric encryption (11, 14). The AES key was transferred as the secret between the server and client using ECDH.

### Blockchain Database Structure

A blockchain database is a sequence of connected blocks, which holds the data as an EMM (**Figure 3**). The block header stores the previous block hash, allowing the block to only have one parent block. The starting block of the blockchain is named the genesis block, as it has no parent block. Blockchains display immutable blocks, decentralization, and data persistence (5).

The blockchain database was created in a tree format, more specifically, an AVL tree. Instead of a linear blockchain, which would display the runtime of $O(n)$, an AVL tree was used to ensure the height of the blockchain tree does not exceed log $n$, where $n$ is the number of blocks in the blockchain and therefore runtime of $O(\log n)$. AVL trees are a type of self-balancing tree, which means that they rearrange the nodes on the tree to make it the constant height. However, this is not possible in a blockchain, where each block is connected to another block through its hash and address. Therefore, it is not possible to rearrange the blocks on the tree to create an even height. To mitigate this, when appending a block into the chain, the block would be inserted at the closest directory. This directory was based on the next spot in the tree available to append. Each block was given a key, and according to the binary version of the key, the block will go to that directory, where 0 is left branch and 1 is the right branch. The closest directory of the block went to a separate dictionary, where the Query function references to find the block. This dictionary is called the directory dictionary.

### Block

The two parts of the block are the block header and the block body (**Figure 4**). A block header comprises of:

**Key:** a number that references block number in the blockchain database and, when converted into binary, its directory to reach the block.

**Parent block Hash:** A hash of the previous block.

**Timestamp:** The block time is in Unix epoch time, the current time in seconds since January 1, 1970.

The block body is formed with EMMs and primary key. The EMMs store the data that is directed by the user, as detailed above. The primary key is a string that references the EMM, where the user enters a primary key to find the corresponding data.

### Algorithms

A blockchain database (BCD) consists of four algorithms. The first algorithm in BCD is the append algorithm.

**Equation 2.1.** (BCD') ← Append($k$, $d$, $p$)

Where $k$ is the user-inputted primary key to search the database, d is the data corresponding to the primary key in the form of a JSON file, and $p$ is the private key of the user to decrypt $d$. To append into the blockchain tree and retain the height of log n, the block is assigned a key, which is the current block number. The key is then converted into binary and used to navigate the tree to insert at a certain location, where 0 is the left branch and 1 is the right branch. A directory dictionary is used to store k and the key that it is given. (**Figure 5**)

The JSON file is converted into a Python dictionary then encrypted with the AES key, which was the secret message transmitted between the server and client using ECDH.

The next algorithm in the BCD is the query function.

**Equation 2.2.** $d$ ← Query($k$, $p$)

Where $k$ is the primary key given to search for d and p is the client's private key. The query method uses the key that corresponds to the primary key in the directory dictionary to find the block needed. The data d comes in the form of an EMM and gets decrypted on the client's end.

Furthermore, the next algorithm in the BCD is the delete function.

**Equation 2.3.** (BCD') ← Delete($p$)

Where $p$ is the primary key that corresponds to the block. Blockchains are immutable, which makes deletions not possible. To create a fix to this problem, Eq. 3 uses the primary to delete the directory from the directory dictionary, and therefore cannot find the block when Eq. 2 is called.

The final algorithm is the update algorithm,

**Equation 2.4.** (BCD') ← Update($k$, $d$, $p$)

Where $k$ is the user-inputted primary key to search the database, $d$ is the data to replace the current data, which is corresponding to the primary key in the form of a JSON file, and $p$ is the private key of the user to decrypt $d$. Eq. 4 is a combination of Eq. 1 and Eq. 3, where the function first deletes the existing entry with the primary key in the directory dictionary and then uses the append function to create another block with the same primary key and the newly inputted data.

## ACKNOWLEDGMENTS

We would like to thank the Ardent Research Team for the tools to conduct this research. We would also like to thank L. Klein for advice and comments on the paper.

**Received:** April 08, 2022

**Accepted:** June 16, 2022
**Published:** March 18, 2023

## Appendix

GitHub Repository: https://github.com/ayushimehrotra/Blockchain_Database.git

## REFERENCES

1. Wright, Craig S. "Bitcoin: A Peer-to-Peer Electronic Cash System." SSRN Electronic Journal, 2008, doi.org/10.2139/ssrn.3440802.
2. Zheng, Zibin, *et al.* "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends." 2017 IEEE International Congress on Big Data (BigData Congress), 2017, doi.org/10.1109/bigdatacongress.2017.85.
3. Jing Han, *et al.* "Survey on NoSQL Database." 2011 6th International Conference on Pervasive Computing and Applications, 2011, https://doi.org/10.1109/icpca.2011.6106531.
4. Nisa, Behjat U. "A Comparison between Relational Databases and NoSQL Databases." International Journal of Trend in Scientific Research and Development, Volume-2, no. Issue-3, 2018, pp. 845–848., doi.org/10.31142/ijtsrd11214.
5. Dasgupta, Dipankar, *et al.* "A Survey of Blockchain from Security Perspective." Journal of Banking and Financial Technology, vol. 3, no. 1, 2019, pp. 1–17., doi.org/10.1007/s42786-018-00002-6.
6. Peng, Yanqing, *et al.* "FalconDB: Blockchain-Based Collaborative Database." Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, 2020, https://doi.org/10.1145/3318464.3380594.
7. El-Hindi, Muhammad, *et al.* "BlockchainDB - towards a Shared Database on Blockchains." Proceedings of the 2019 International Conference on Management of Data, 2019, doi.org/10.1145/3299869.3320237.
8. Nathan, Senthil, *et al.* "Blockchain Meets Database." Proceedings of the VLDB Endowment, vol. 12, no. 11, 2019, pp. 1539–1552., doi.org/10.14778/3342263.3342632.
9. BigchainDB 2.0 the Blockchain Database. www.bigchaindb.com/whitepaper/bigchaindb-whitepaper.pdf.
10. MongoDB, https://www.mongodb.com/.
11. Adkins, Daniel, *et al.* "Encrypted Blockchain Databases." Proceedings of the 2nd ACM Conference on Advances in Financial Technologies, 2020, doi.org/10.1145/3419614.3423266.
12. Okman, Lior, *et al.* "Security Issues in Nosql Databases." 2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications, 2011, doi.org/10.1109/trustcom.2011.70.
13. Cooper, Brian F., *et al.* "Benchmarking Cloud Serving Systems with YCSB." Proceedings of the 1st ACM Symposium on Cloud Computing - SoCC '10, 2010, doi.org/10.1145/1807128.1807152.
14. Curtmola, Reza, *et al.* "Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions." Journal of Computer Security, vol. 19, no. 5, 2011, pp. 895–934., doi.org/10.3233/jcs-2011-0426.
15. "Python 3.10.4 Documentation." 3.10.4 Documentation, docs.python.org/3.10/.
16. Gauravaram, Praveen. "Security Analysis of Salt||Password Hashes." 2012 International Conference on Advanced Computer Science Applications and Technologies (ACSAT), 2012, https://doi.org/10.1109/acsat.2012.49.
17. Yassien, Amal W., and Amr F. Desouky. "RDBMS, NoSQL, Hadoop: A Performance-Based Empirical Analysis." Proceedings of the 2nd Africa and Middle East Conference on Software Engineering - AMECSE '16, 2016, https://doi.org/10.1145/2944165.2944174.