SQL Database ledger Overview & Glossary of terms

Last updated by | Pooja Kamath | Jun 11, 2021 at 3:41 AM PDT

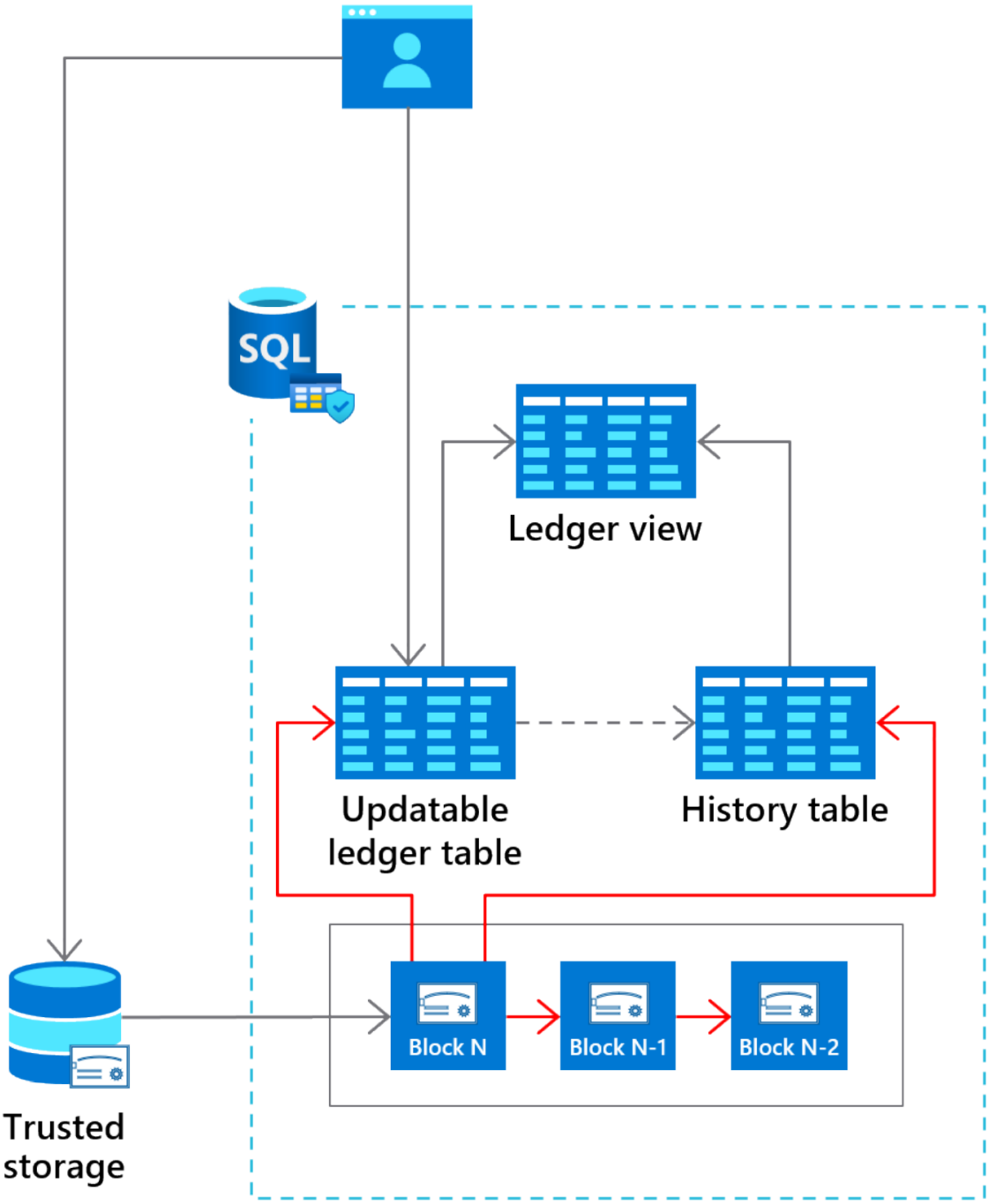
Contents

Azure SQL Database ledger Overview & Glossary of terms

Azure SQL Database ledger gives you the ability to detect any potential tampering of the data in your database. **Using the same cryptographic patterns employed in blockchains,** the ledger feature uses SHA 256 to hash transactions, linked together. The hashes (database digests) produced are then pushed outside of Azure SQL Database to a tamper-proof storage service, such as Azure Storage immutable blobs, or Azure Confidential Ledger. Database digests are used to then verify the integrity of your database by comparing them to the real-time computed hashes inside of your database.

Ledger helps protects data from any attacker or high privileged user, including DBAs and system and cloud administrators. Just like a traditional ledger, historical data is preserved such that if a row is updated in the database, it's previous value is maintained and protected in a history table, providing a chronicle of all changes made to the database over time. The ledger and the historical data are managed transparently, offering protection without any application changes. Historical data is maintained in a relational form to support SQL queries for auditing, forensics and other purposes. Ledger provides cryptographic data integrity guarantees

while maintaining the power, flexibility and performance of Azure SQL Database.



Benefits of using Azure SQL Database Ledger

Here are some benefits of the ledger feature:

- The ability to prove to 3rd parties such as auditors or other business parties that your data has not been maliciously altered.
- Can be used with existing data patterns where UPDATE and DELETE operations are expected, or in an append-only data pattern where only INSERTS are allowed.
- Data lineage capabilities, through GENERATED ALWAYS columns which are added to your ledger tables which capture important information such as the ID of the transaction and the order of transaction

operations.

- An easy to query view of the ledger tables.
- Automatic generation and storage of database digests in tamper-proof storage.
- Automatic generation of the T-SQL script needed to run verification on your database.

Typical Scenarios

Streamlining audits

Any production system's value is based on the ability to trust the data the system is consuming and producing. If data in your database has been tampered with by a malicious user, it can have disastrous results in the business processes relying on the data. Maintaining trust in your data entails a combination of enabling the proper security controls to reduce potential attacks, backup and restore practices, as well as thorough disaster recovery procedures. Ensuring these practices are put in place are often audited by external parties. Audit processes are highly time-intensive activities, requiring on-site inspection of implemented practices such as reviewing audit logs, inspecting authentication and access controls, just to name a few. While these manual processes can expose potential gaps in security, what they cannot provide is attestable proof that data has not been maliciously altered. Ledger provides the cryptographic proof of data integrity to auditors which can help not only streamline the auditing process, but also provides non-repudiation regarding the integrity of the system's data.

Multi-party business processes

Systems where multiple organizations have a business process that must share state with one another, such as supply-chain management systems, struggle with the challenge of how to share and trust data with one another. Many organizations are turning to traditional blockchains, such as Ethereum or Hyperledger Fabric, to digitally transform their multi-party business processes. Blockchain is a great solution for multi-party networks where trust is low between parties that participate on the network. However, many of these networks are fundamentally centralized solutions where trust is important, but a fully decentralized infrastructure is a heavy-weight solution. Ledger provides a solution for these networks where participants can verify the data integrity of the centrally-housed data, rather than having the complexity and performance implications that network consensus introduces in a blockchain network.

Trusted off-chain storage for blockchain

Where a blockchain network is necessary for a multi-party business process, having the ability query the data on the blockchain in a performant manner is a challenge. Typical patterns for solving this problem involves replicating data from the blockchain to an off-chain store, such as a database. However, once the data is replicated to the database from the blockchain, the data integrity guarantees that a blockchain offers is lost. Ledger provides the data integrity needed for off-chain storage of blockchain networks, ensuring complete data trust through the entire system.

How it Works

Each transaction that is received by the database is cryptographically hashed (SHA 256). The hash function uses the value of the transaction, along with the hash of the previous transaction, as input to the hash function. This cryptographically links all transactions together, similar to a blockchain. Cryptographic hashes (database digests), which represent the state of the database, are periodically generated and stored outside of Azure SQL

Database in a tamper-proof storage location such as Azure Storage immutable blobs or Azure Confidential Ledger. Database digests are then later used to verify the integrity of the database by comparing the value of the hash in the digest against the calculated hashes in database.

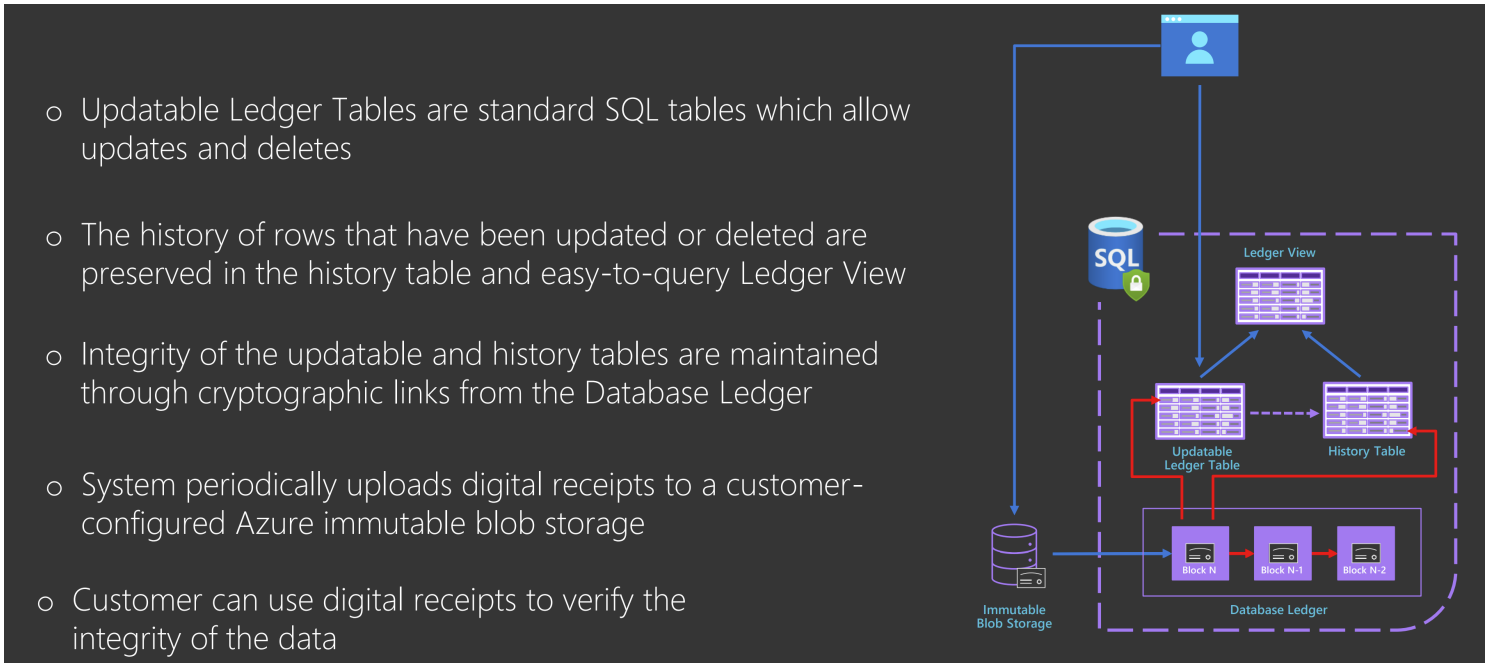
Ledger introduces ledger functionality to tables in Azure SQL Database in 2 forms.

- **Updatable ledger tables**, which allow you to update and delete rows in your tables.
- **Append-only ledger tables**, which only allow inserts to your tables.

Both updatable ledger tables and append-only ledger tables provide tamper-evidence as well as digital forensics capabilities. Understanding which transactions submitted by which users that resulted in changes to the database are important in the event of both remediating potential tampering events, or proving to 3rd parties that transactions submitted to the system were by authorized users. This enables users, their partners or auditors to analyze all historical operations and detect potential tampering. Each row operation is accompanied by the ID of the transaction that performed it, allowing users to retrieve more information about the time the transaction was executed, the identity of the user who executed it and correlate it to other operations performed by this transaction.

Updatable ledger tables

Updatable ledger tables are ideal for application patterns that expect to issue updates and deletes to tables in your database, such as System of Record (SOR) applications. This means that existing data patterns for your application do not need to change to enable ledger functionality. You can create updatable ledger tables by either enabling it at the database-level, which ensures all future tables created in your database are updatable ledger tables, or when creating specific tables when you do not want all tables to be updatable ledger tables by default.

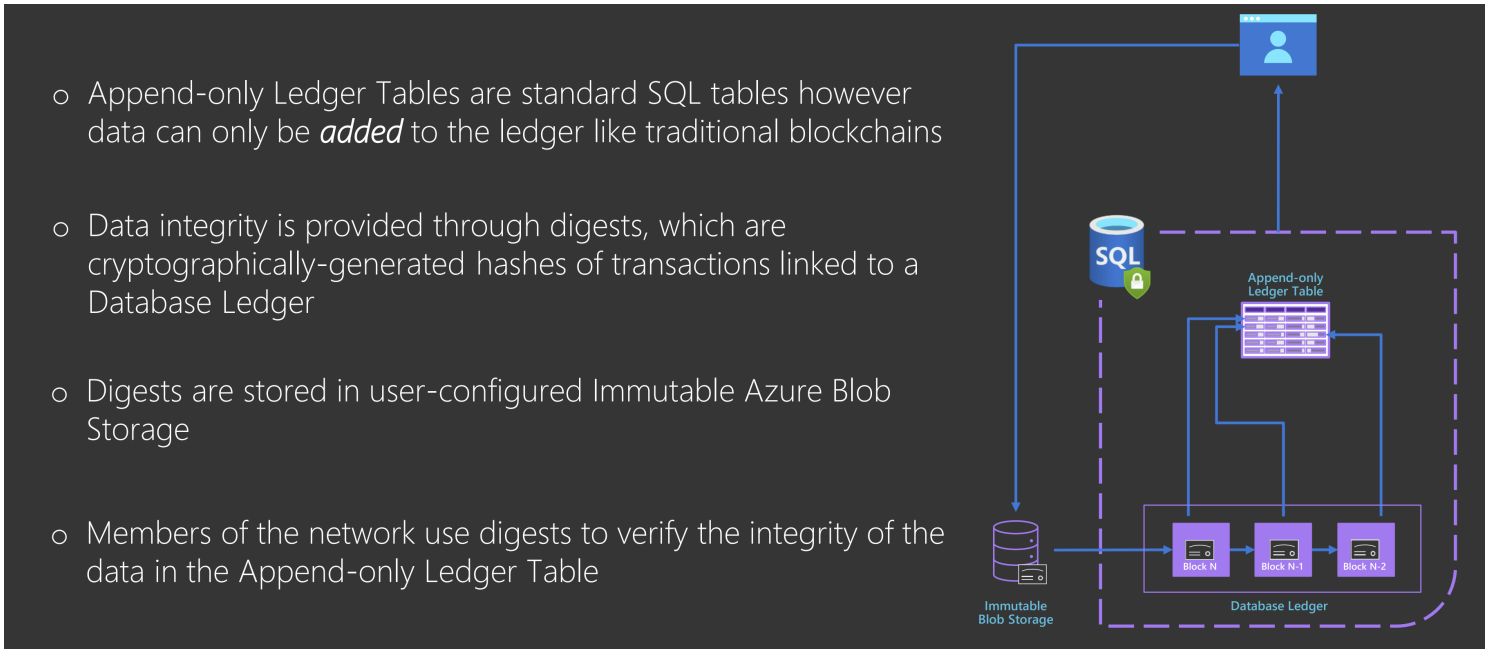


Updatable ledger tables track the history of changes to any rows in your database when transactions that perform updates or deletes occur. An updatable ledger table is a system-versioned table that contains a reference to another table with a mirrored schema. The system uses this table to automatically store the previous version of the row each time a row in the temporal table gets updated or deleted. This additional table is referred to as the history table. The history table is automatically created when you create an updatable ledger table. The values contained in the updatable ledger table and it's corresponding history table

subsequently provide a chronicle of the values of your database over time. In order to easily query this chronicle of your database, a system-generated ledger view is created which joins the updatable ledger table and the history table.

Append-only ledger tables

Append-only ledger tables are ideal for application patterns that are insert-only, such as Security Information and Event Management (SEIM) applications. Append-only ledger tables block updates and deletes at the API, providing further tampering protection from privileged users such as systems administrators and DBAs. Since only inserts are allowed into the system, append-only ledger tables do not have a corresponding history table as there is no history to capture. Like updatable ledger tables, a ledger view is created providing insights into the transaction that inserted rows into the append-only table, as well as the user that performed the insert.



Database ledger

The database ledger consists of system tables which store the cryptographic hashes of transactions processed in the system. Since transactions are the unit of atomicity for the database engine, this is the unit of work being captured in the database ledger. Specifically, when a transaction commits, the SHA-256 hash of any rows modified by the transaction in the ledger table, together with some metadata for the transaction, such as the identity of the user that executed it and its commit timestamp, are appended as a “transaction entry” in the database ledger. Every 30 seconds, the transactions processed by the database are SHA-256 hashed together using a Merkle tree data structure, producing a root hash. This forms a block, which is subsequently SHA-256 hashed using the root hash of the block along with the root hash of the previous block as input to the hash function, forming a blockchain.

Database digests

The hash of the latest block in the database ledger is known as the database digest and represents the state of all ledger tables in the database at the time the block was generated. When a block is formed, it's associated database digest is then published and stored outside of Azure SQL Database in tamper-proof storage. Since database digests represent the state of the database at the point in time they were generated, protecting the digests from tampering is paramount. An attacker that has access to modify the digests would be able to

tamper with the data in the database, generate the hashes representing the database with the tampered changes, and subsequently modify the digests to represent the updated hash of the transactions in the block. Ledger provides the ability to automatically generate, and store, the database digests in Azure immutable blob storage, or Azure Confidential Ledger, to prevent tampering. Alternatively, users can manually generate database digests, storing them in the location of their choice. Database digests are used for later verifying that the data stored in ledger tables has not been tampered.

Ledger verification

The ledger feature does not allow users to modify the content of the ledger, however, an attacker or system administrator who has control of the machine can bypass all system checks and directly tamper with the data, for example by editing the database files in storage. Ledger cannot prevent such attacks but guarantees that any tampering will be detected when the ledger data is verified. The ledger verification process takes as input one or more previously generated database digests and recomputes the hashes stored in the database ledger based on the current state of the ledger tables. If the computed hashes do not match the input digests, the verification fails, indicating that the data has been tampered with, and reports all inconsistencies detected.

Since the ledger verification recomputes all of the hashes for transactions in the database, it can be a resource intensive process for databases with large amounts of data. Running the ledger verification should be done only when users need to verify the integrity of their database rather than in a continuous manner. Ideally, ledger verification should be run only when the organization hosting the data goes through an audit and needs to provide cryptographic evidence regarding the integrity of their data to another party. To reduce the cost of verification, ledger exposes options to verify individual ledger tables or only a subset of the ledger.

How good have you found this content?