Data on the Blockchain

This article will cover a high-level overview of Blockchain concepts and tools which facilitate reading and writing Blockchain data.

Writing data

By design, a Blockchain is a distributed database, and blockchain data is optimized to be stored in serialized blocks, writing one block at a time as the chain is being created. These serialized blocks store information about primitive components like transactions, accounts, and contracts.

Blockchain's writing process provides a great way to apply the requested changes to the state in a decentralized manner. However, to observe the changes, you need to actively pull the information from the network. As a result, traversing several blocks and aggregating data is necessary when trying to get data across multiple contracts or a longer time range.

Reading data

Querying the Blockchain for data about a specific block or account is fairly straightforward and can be considered a "narrow" query. However, querying data across many blocks can be cumbersome because you must aggregate results from multiple single-block queries. Therefore, these can be considered "wide" queries.

info The nature of a blockchain is that it must be deterministic. A critical feature of a blockchain is that it knows the state at a given block (or time unit). Think of them as snapshots – a blockchain does snapshots of its state on every block. For example, a user can call a smart contract for a specific block, and the blockchain guarantees that the execution will always produce the same result for the same block any time you call it.

note The deterministic nature of a blockchain isolates it from off-chain (external) variables. It is impossible to call an external API to read data from within a smart contract. Therefore, blockchains and smart contracts are closed-off from the outer (off-chain) world.

Information Availability

As mentioned, the essential information you can acquire from the Blockchain's state is related to the transactions, accounts, and contracts taking place at a given block.

A simple way to get such information from the blockchain is using RPC. NEAR implements a JSON-RPC endpoint for everyone to interact with the blockchain. Through the JSON-RPC API, users can call smart contracts and view the data from the blockchain. Keep in mind that not all the data is available through the JSON-RPC. For example, local Receipts are unavailable through the JSON-RPC because they are not stored in the NEAR node's internal database.

tip The JSON-RPC endpoint approach is the so-called pull model of getting the data. There is nothing wrong with this approach, but it is sometimes not the most comfortable or reliable alternative. As you can imagine, this basic block information might not be enough for a complex project. For example, in an NFT marketplace, basic read operations via RPC calls could help get the status of a purchase or get the NEAR balance of an NFT creator account, but for information like getting the owner of a certain NFT, getting the metadata of an NFT based on its ID, or the total supply for a certain type of NFT, dApp developers would need to do additional parsing and processing after calling the RPC primitive methods.

More advanced real-world queries and operations like aggregation, search, relationships, and non-trivial filtering are simply not possible using RPC calls alone. An example of this is showing a leaderboard of the most popular NFT collections in the last 7 days. It would not be possible to get that information by interacting directly with the contract itself or via RPC. Getting this data would require processing every single transaction that ever happened, aggregating them in terms of collection, filtering them by timestamps calculation, and ordering them in terms of total transacted value.

One option to solve these complex data requirements is to build a server using an indexing framework. A blockchain indexer is an implementation of the push model of getting the data. Instead of actively pulling the data from the source, indexers listen to the stream of data from the blockchain, and the data can be immediately filtered and processed according to defined requirements. Indexers can also be used to simplify the "wide" query execution. For example, a data stream can be written to a permanent database for later data analysis using a convenient query language like SQL.

Data tools

Building a server and indexing blockchain data on your own is not an easy task. The index server will be responsible for processing all the transactions and saving them to a database. Then you'll need to build a set of APIs to get the data for your dApp. This option is resource intensive, requires maintenance, presents a single point of failure, and breaks important security properties required for decentralization. Also, specific Blockchain properties like finality, chain reorganizations, or empty blocks complicate this process further and make it not just time-consuming but conceptually hard to retrieve correct

query results from blockchain data.

As an alternative, you can try one of the many available indexing services on the NEAR blockchain, and here is a suggestive decision tree of which tool to use for your data needs presented by Pagoda:

Data Platform Decision Tree * NEAR Indexer Framework * : a micro-framework providing you with a "live" stream of blocks. Useful to handle on-chain real-time "events". * NEAR Indexer for Explorer * : leverages the indexer micro-framework to watch and store all of the blockchain's events/data into a transactional PostgreSQL database. You can clone theGitHub repository * and customize your own indexer solution. *NEAR Lake Framework * : a companion library to NEAR Lake. It allows you to build your own indexer that watches a stream of blocks from a NEAR Lake data source and allows you to create your own logic to process that data. Keep in mind this is probably the one you want to use for future projects, instead of the Indexer Framework. Readwhy is better * . *NEAR Lake Indexer * : leverages the indexer micro-framework to watch and store all of the blockchain's events/data as JSON files on a user-specified AWS S3 or S3-compatible storage. * Near Query API * : Query API allows you to seamlessly create, manage, and discover indexers on NEAR. Developers can deploy their indexers to aggregate historical data in a matter of minutes, while accessing the data via custom GraphQL queries. * The Graph * : The Graph gives developers tools to process blockchain events and make the resulting data easily available via a GraphQL API, known individually as a subgraph.Graph Node * is now able to process NEAR events, which means that NEAR developers can now build subgraphs to index their smart contracts. * Pagoda API * is a hosted service managed by Pagoda that solves the complex data problem by indexing the blockchain into datamarts and is accessible via standard RESTful APIs for you to directly use in any part of your application. The API enables performant and efficient querying of blockchain data without having to build and manage your own indexer infrastructure. Pagoda also offers the Indexer framework as a decentralized solution to indexing on-chain data, and a centralized Data Lake source that can be used to build your own indexers and servers for such data query needs. * PIKESPEAK API * : an enterprise-grade API where you can fetch blockchain events and aggregated analytics on wallets, validators, delegators, money transfers, dapps activity and more.Documentation * Mintbase Indexer * A highly efficient and adaptable indexing solution developed to capture, process, and store NFT events throughout the NEAR ecosystem. This indexer provides developers with a streamlined and well-structured data stream, making it an optimal choice for constructing data-centric applications in the NFT space. By systematically monitoring and indexing a wide array of NFT-related events across various platforms and projects on NEAR, the Mintbase Indexer offers developers the necessary tools and data access to build innovative and practical applications. Furthermore, the indexed data can be conveniently queried using GraphQL, allowing for flexible and efficient data retrieval tailored to specific developer needs.Documentation * SubQuery * : SubQuery is a fast, flexible, and reliable open-source data indexer that provides you with custom APIs for your web3 project across NEAR and many other chains. NEAR developers will benefit from the superior SubQuery experience, including the open-source SDK, tools,documentation * , and developer support that the SubQuery ecosystem provides. Additionally, NEAR is accommodated bySubQuery’s Managed Service * , which provides enterprise-level infrastructure hosting.Edit this page Last updatedonFeb 7, 2024 bys-n-park Was this page helpful? Yes No

Previous Use cases Next Intro to Indexers