Decentralized Database

Learn how to store the application data with a decentralized database on Filecoin.

Store data with Tableland

Tableland is adecentralized database built on the SQLite engine, which offers developers a web3-native, relational database that seamlessly integrates into their EVM-compatible stacks. Under the hood, Tableland records database tables as ERC721 tokens on-chain and enables the execution of SQL statements in a completely decentralized manner through on-chain smart contracts.

To learn more about what is tableland and how to use it, you can visithttps://tableland.xyz/ .

Ingredients

Ensure that you install and import the necessary dependencies in your projects.

- Tableland
- RaaS Starter Kit
- Openzeppelin
-

Instructions

Let's take storage deal aggregator/RaaS as an example to demonstrate how to integrate it with Tableland.

When uploading data via aggregator/RaaS providers to the Filecoin network, you can choose to store its metadata in Tableland tables instead of storing it in the chain state. This metadata can then be easily accessed from the Tableland database and utilized directly within your application.

If you require sample datasets to use, you can use theFilecoin Dataset Explorer .

As an example, let's design the deal aggregator table as follows. However, you can certainly add more columns to this table to include additional information, such as RaaS registration.

column data Type ID int CID bytes/string deal_ID int miner_ID int status string 1. Create aggregator table 2.

To track all the deal aggregation/RaaS requests submitted to the smart contract, we need to create a database table. You can add the following code to create an aggregator table within theconstructor() function of the aggregator contract. This way, when the aggregator/RaaS contract is deployed, an aggregator table will be created to store the metadata of the aggregation requests.

```

Copy uint256privatetableId; stringprivateconstant_TABLE_PREFIX="aggregaor_table";// Custom table prefix

// Constructor that creates a table, sets the controller, and inserts data constructor() { // Create a table tableId=TablelandDeployments.get().create( address(this), SQLHelpers.toCreateFromSchema( "id int, cid string, deal_id string, miner_id string, status string", _TABLE_PREFIX ) ); }C

```

1. We will create aninsert
2. function within the smart contract to add a record whenever an aggregation request is made.
3.

```

Copy functioninsertRecord(uint256id,stringmemoryval,stringmemorystatus)external{

TablelandDeployments.get().mutate( address(this),// Table owner, i.e., this contract _tableId, SQLHelpers.toInsert( _TABLE_PREFIX, _tableId, "id,cid,status", string.concat( Strings.toString(id),// Convert to a string ",", SQLHelpers.quote(val)// Wrap strings in single quotes with the quote method ) ));}

```

Whenever thesubmit orsubmitRaaS function is called, a record will be inserted into the aggregator table instead of being stored in the blockchain's state.

```
```

Copy functionsubmit(bytesmemory_cid)externalreturns(uint256) { // Increment the transaction ID transactionId++;

// Save _cid record to aggregator_table insertRecord(transactionId,_cid,"PROPOSED");

// Emit the event emitSubmitAggregatorRequest(transactionId,_cid); returntransactionId; }

```

1. we create anupdateRecord
2. function to modify an aggregator record once thecomplete
3. function is called after the storage deal has been made on the Filecoin network.
4.

```

Copy // Update aggregation record in the table functionupdateRecord(uint256id,uint256memorydealId,uint256memoryminerId,stringmemorystatus)external{ // Set the values to update stringmemorysetters=string.concat("deal_id=",Strings.toString(dealId), "miner_id=",Strings.toString(minerId), "status=",SQLHelpers.quote(status)); // Specify filters for which row to update stringmemoryfilters=string.concat("id=",Strings.toString(id)); // Mutate a row at id with deal_id, miner_id, status TablelandDeployments.get().mutate(address(this), _tableId, SQLHelpers.toUpdate(_TABLE_PREFIX,_tableId,setters,filters)); }

```

After SP finishes publishing the storage deal on-chain to include an aggregation request, a callback functioncomplete will be called to notify the contract that a CID is packed into a storage deal. Then we can callupdateRecord to update the details for this CID record in the Tableland database.

```

Copy functioncomplete(uint256_id, uint64_dealId, uint64_minerId, InclusionProofmemory_proof, InclusionVerifierDatamemory_verifierData)externalreturns(InclusionAuxDatamemory) { // other code updateTable(_id,_dealId,_minerId,"FINISHED"); }

```

1. Query aggregation records
2.

By using the Tableland SDK, you can easily query the aggregation or RaaS status of all the data stored with the aggregator using SQL statements. For instance, you can retrieve all records associated with a specific CID by executing a SELECT statement.

```

Copy constdb=newDatabase();

const{results}=awaitdb.prepare(SELECT * FROM{tableName}WHERE cid = ?) .bind(cid); console.log(results);

```

To learn how to write different select statements using Tableland SDK, you can refer tohere .

Previous Cross-Chain Bridges

Last updated21 days ago
```