



**Making the world's
decentralised data more
accessible.**



Lab Exercise Guide

Table of Contents

Introduction	2
Pre-requisites	2
Exercise 1: Index Staking Rewards	2
High level steps	2
Detailed steps	3
Step 1: Initialize your project	3
Step 2: Update the graphql schema	3
Step 3: Update the manifest file (aka project.yaml)	4
Step 4: handleStakingRewarded	5
Step 5: Build the project	6
Step 4: Query the project	7
Exercise 2: Aggregate Staking Rewards	9
Detailed Steps	9
Step 1: Add an entity called Staking Reward	9
Step 2: Update the manifest file	9
Step 3: handleSumRewarded	11
Step 4: Rebuild the project	11
Step 5: Query the project	11
Exercise 3: Viewing Both Aggregated and Individual Staking Rewards	12
Detailed Steps	12
Step 1: Modify the schema file	12
Step 2: Update handleStakingRewarded	13
Step 3: Rebuild the project	14
Step 4: Query the project	14
Exercise 4: Reward v Rewarded	16
Step 1: Update the manifest file	16
Step 2: Update the mapping file	16
Step 3: Rebuild the project	16
Step 4: Query the project	17

Introduction

In this workbook, we will take the starter project and look at how we can aggregate data. Specifically, we will index staking rewards and then aggregate them over for a particular account. In effect we are determining how much reward an account has accumulated over time.

Pre-requisites

Completion of Module 3.

Exercise 1: Index Staking Rewards

Before we can aggregate all the staked rewards earned by a user or more specifically an DOT account owner, we need to index all the staking rewards.

High level steps

1. Initialise the starter project
2. Update your mappings, manifest file and graphql schema file.
3. Generate, build and deploy your code
4. Deploy your code in Docker
5. Query for address balances in the playground

Detailed steps

Step 1: Initialize your project

The first step in creating a SubQuery project is to create a project with the following command:

```
~/Code/subQuery/workshop$ subql init
Project name [subql-starter]: staking-rewards
? Select a network Polkadot
? Select a template project subql-starter Starter project for
subquery
Cloning project... done
RPC endpoint: [wss://polkadot.api.onfinality.io/public-ws]:
Git repository [https://github.com/subquery/subql-starter]:
Fetching network genesis hash... done
Author [Ian He & Jay Ji]:
Description [This project can be use as a starting po...]:
Version [0.0.4]:
License [MIT]:
Preparing project... done
staking-rewards is ready
```

Step 2: Update the graphql schema

Add an entity called StakingReward. This has fields that allows us to record the account reward along with the balance. The block height will allow us to do a cross check.

```
type StakingReward @entity{
  id: ID! #blockHeight-eventIdx
  account: String!
  balance: BigInt!
  date: Date!
  blockHeight: Int!
}
```

Step 3: Update the manifest file (aka project.yaml)

Update the manifest file to only include a `handleStakingRewarded` handler and update the filter method to `staking/Rewarded`. This is the only event we want to capture for now.

```
- handler: handleStakingRewarded
  kind: substrate/EventHandler
  filter:
    module: staking
    method: Rewarded
```

Note: The `Rewarded` method was only recently introduced from block [6,713,249](#) onwards. It was previously called `Reward`. For this exercise, we will use this newer format and use a `startBlock` of 7,000,000

The full manifest file should look like this:

```
specVersion: 0.2.0
name: staking-rewards
version: 0.0.4
description: >-
  This project can be use as a starting point for developing your
  SubQuery
  project
repository: 'https://github.com/subquery/subql-starter'
schema:
  file: ./schema.graphql
network:
  endpoint: 'wss://polkadot.api.onfinality.io/public-ws'
  genesisHash:
'0x91b171bb158e2d3848fa23a9f1c25182fb8e20313b2c1eb49219da7a70ce90c3'
dataSources:
- kind: substrate/Runtime
  startBlock: 7000000
  mapping:
    file: ./dist/index.js
    handlers:
      - handler: handleStakingRewarded
        kind: substrate/EventHandler
        filter:
          module: staking
          method: Rewarded
```

Note: We are starting at block height 7 million in this example and indenting matters. Otherwise you will get the following error:

```
./node_modules/.bin/subql codegen
=====
-----Subql Codegen-----
=====
bad indentation of a sequence entry (17:5)

 14 |     mapping:
 15 |       file: ./dist/index.js
 16 |       handlers:
 17 |     - handler: handleRewarded
-----^
 18 |         kind: substrate/EventHandler
 19 |         filter:
error Command failed with exit code 1.
```

Step 4: handleStakingRewarded

The initialisation command pre-creates a sample mappings file with 3 functions, handleBlock, handleEvent and handleCall. Delete all of them as we will create our own.

```
export async function handleStakingRewarded(event: SubstrateEvent):
Promise<void> {
}
}
```

Next, we declare an event object as follows:

```
const {event: {data: [account, newReward]}} = event;
```

We then declare a new instance of the StakeReward object and pass through the blockheight + hyphen + eventid to create a unique identifier.

```
const entity = new
StakingReward(`${event.block.block.header.number}-${event.idx.toString()
}`);
```

We then obtain the account, newReward and the block timestamp and store it within the relevant fields within our entity object. We then save the entity.

```
entity.account = account.toString();
entity.balance = (newReward as Balance).toBigInt();
entity.date = event.block.timestamp;
entity.blockHeight = event.block.block.header.number.toNumber();
await entity.save();
```

The full mapping file should look like this:

```
import {SubstrateEvent} from "@subql/types";
import {StakingReward} from "../types";
import {Balance} from "@polkadot/types/interfaces";

export async function handleStakingRewarded(event: SubstrateEvent):
Promise<void> {
  const {event: {data: [account, newReward]}} = event;
  const entity = new
StakingReward(`${event.block.block.header.number}-${event.idx.toString()
}`);
  entity.account = account.toString();
  entity.balance = (newReward as Balance).toBigInt();
  entity.date = event.block.timestamp;
  entity.blockHeight = event.block.block.header.number.toNumber();
  await entity.save();
}
```

Step 5: Build the project

Run the standard yarn install, codegen, build and docker-compose pull & docker-compose up commands.

```
yarn install
OR
npm install
```

```
yarn codegen
OR
npm run-script codegen
```

```
yarn build
```

```
OR
npm run-script build
```

```
docker-compose pull && docker-compose up
```

Step 4: Query the project

Once the docker container is up and running, which could take a few minutes, open up your browser and navigate to www.localhost:3000.

This will open up a “playground” where you can create your query. Copy the example below:

```
query{
  stakingRewards(first: 3 orderBy:BLOCK_HEIGHT_ASC){
    nodes{
      blockHeight
      account
      date
      balance
    }
  }
}
```

This should return something similar to below:

```
{
  "data": {
    "stakingRewards": {
      "nodes": [
        {
          "blockHeight": 7000064,
          "account": "16jWQMBXZNxfgXJmVL61gMX4uqtc9WTXV3c8DGx6DUKejm7",
          "date": "2021-09-26T16:50:18.001",
          "balance": "2189068638"
        },
        {
          "blockHeight": 7000064,
          "account": "13MnytvdGdqJLGZbizqd8CDKJUPa9UJyzXcdxRiJEv5g2hq47",
          "date": "2021-09-26T16:50:18.001",
          "balance": "2050030971"
        }
      ]
    }
  }
}
```



```
"blockHeight": 7000064,  
"account": "12L117g377z195J3WaPshEaFC8vsNMyiMi8CTWfWVJdmBAJ4",  
"date": "2021-09-26T16:50:18.001",  
"balance": "2007885451"  
},  
{  
  "blockHeight": 7000064,  
  "account": "13owVsvG3GtmDYfcnDNCVm54z6X6VgYf37QRMFywrVPpkJvv",  
  "date": "2021-09-26T16:50:18.001",  
  "balance": "1987101808"  
},
```

Congratulations. You have now indexed all staking rewards for all accounts from block 7M onwards. Next, let's aggregate or sum up these rewards for each account.

Exercise 2: Aggregate Staking Rewards

To aggregate the staking rewards, we first of all need to create another entity.

Detailed Steps

Step 1: Add an entity called Sum Reward

Add a new entity called SumReward with extra fields as seen below.

```
type SumReward @entity{  
  id: ID! # AccountId  
  totalReward: BigInt!  
  blockheight: Int!  
}
```

The new schema file should now look like this:

```
type StakingReward @entity{  
  id: ID! #blockHeight-eventIdx  
  account: String!  
  balance: BigInt!  
  date: Date!  
}  
  
type SumReward @entity{  
  id: ID! # AccountId
```

```
totalReward: BigInt!  
blockheight: Int!  
}
```

Step 2: Update the manifest file

Add an extra handler called `handleSumRewarded` and filter it by `staking/Rewarded`.

```
- handler: handleSumRewarded  
  kind: substrate/EventHandler  
  filter:  
    module: staking  
    method: Rewarded
```

The complete manifest file should look like:

```
specVersion: 0.2.0  
name: staking-rewards  
version: 0.0.4  
description: >-  
  This project can be use as a starting point for developing your  
  SubQuery  
  project  
repository: 'https://github.com/subquery/subql-starter'  
schema:  
  file: ./schema.graphql  
network:  
  endpoint: 'wss://polkadot.api.onfinality.io/public-ws'  
  genesisHash:  
  '0x91b171bb158e2d3848fa23a9f1c25182fb8e20313b2c1eb49219da7a70ce90c3'  
dataSources:  
  - kind: substrate/Runtime  
    startBlock: 7000000  
    mapping:  
      file: ./dist/index.js  
      handlers:  
        - handler: handleSumRewarded  
          kind: substrate/EventHandler  
          filter:  
            module: staking
```

```
    method: Rewarded
  - handler: handleStakingRewarded
    kind: substrate/EventHandler
    filter:
      module: staking
      method: Rewarded
```

Note: This is how more than 1 mapping handler can be added to a project. Also note that the order is important too. Otherwise you may encounter an error such as:

```
ERROR failed to index block at height 7000064 handleStakingRewarded()
SequelizeForeignKeyConstraintError: insert or update on table
"staking_rewards" violates foreign key constraint
"staking_rewards_account_id_fkey"
```

Step 3: handleSumRewarded

Next, create a function called `handleSumRewarded` along with a helper function called `createSumReward`.

```
function createSumReward(accountId: string): SumReward {
  const entity = new SumReward(accountId);
  entity.totalReward = BigInt(0);
  return entity;
}

export async function handleSumRewarded(event: SubstrateEvent):
Promise<void> {
  const {event: {data: [account, newReward]}} = event;
  let entity = await SumReward.get(account.toString());
  if (entity === undefined){
    entity = createSumReward(account.toString());
  }
  entity.totalReward = entity.totalReward + (newReward as
Balance).toBigInt();
  entity.blockheight = event.block.block.header.number.toNumber();
  await entity.save();
}
```

Note: Run `yarn codegen` and import the new entity to remove the errors.

The mappings file should now look like:

```
import {SubstrateEvent} from "@subql/types";
import {StakingReward, SumReward} from "../types";
import {Balance} from "@polkadot/types/interfaces";

export async function handleStakingRewarded(event: SubstrateEvent):
Promise<void> {
  const {event: {data: [account, newReward]}} = event;
  const entity = new
StakingReward(`${event.block.block.header.number}-${event.idx.toString()
}`);
  entity.account = account.toString();
  entity.balance = (newReward as Balance).toBigInt();
  entity.date = event.block.timestamp;
  await entity.save();
}

function createSumReward(accountId: string): SumReward {
  const entity = new SumReward(accountId);
  entity.totalReward = BigInt(0);
  return entity;
}

export async function handleSumRewarded(event: SubstrateEvent):
Promise<void> {
  const {event: {data: [account, newReward]}} = event;
  let entity = await SumReward.get(account.toString());
  if (entity === undefined){
    entity = createSumReward(account.toString());
  }
  entity.totalReward = entity.totalReward + (newReward as
Balance).toBigInt();
  entity.blockheight = event.block.block.header.number.toNumber();
  await entity.save();
}
```

Step 4: Rebuild the project

See building a project in the previous exercise.

Note: Because we have modified the schema, delete your database instance in the .data folder.

Step 5: Query the project

The following query will list out the total rewards for each account.

```
query{
  sumRewards(first:3 orderBy:BLOCKHEIGHT_ASC){
    nodes{
      blockheight
      id
      totalReward
    }
  }
}
```

You should see something similar to below:

```
{
  "data": {
    "sumRewards": {
      "nodes": [
        {
          "blockheight": 7000064,
          "id": "121FXj85TuKfrQM1Pdcjj4ibbJNnfsqCtMsJ24rSvGEdWDdv",
          "totalReward": "10901386603"
        },
        {
          "blockheight": 7000064,
          "id": "123MFw5gAkCjcqEhapJ5zon4Ppyp59Rq2kyNQqEHbfwvM4Ni",
          "totalReward": "1023809925"
        },
        {
          "blockheight": 7000064,
          "id": "129N6sYY5r9LnfaMY2AG9px9yYyUhN6FERPXLfirwBrjkJv",
          "totalReward": "980420660"
        }
      ]
    }
  }
}
```

```
    }  
  ]  
}  
}  
}
```

This is great, but wouldn't it be better if we could not only display the totalReward, but also the individual rewards that made up the totalReward? We'll look at this in the next exercise.

Exercise 3: Viewing Both Aggregated and Individual Staking Rewards

So far in this workbook, we managed to query for all the staking rewards and then aggregate or add them all together for each account. Now we will make another improvement to allow us to view the aggregate amount as well as the individual amounts as a child set.

Detailed Steps

Step 1: Modify the schema file

Update the graphql schema field called account to be type SumReward. We are creating a one-many entity relationship where one sumReward will comprise of many individual staking rewards.

```
type StakingReward @entity{  
  id: ID! #blockHeight-eventIdx  
  account: SumReward!  
  balance: BigInt!  
  date: Date!  
}
```

Step 2: Check the manifest file

Check that the manifest file looks like the below:

```
specVersion: 0.2.0  
name: staking-rewards  
version: 1.0.0  
description: ''  
repository: ''
```

```
schema:
  file: ./schema.graphql
network:
  genesisHash:
    '0x91b171bb158e2d3848fa23a9f1c25182fb8e20313b2c1eb49219da7a70ce90c3'
  endpoint: wss://polkadot.api.onfinality.io/public-ws
dataSources:
  - kind: substrate/Runtime
    startBlock: 7000000
    mapping:
      file: ./dist/index.js
      handlers:
        - handler: handleSumRewarded
          kind: substrate/EventHandler
          filter:
            module: staking
            method: Rewarded
        - handler: handleStakingRewarded
          kind: substrate/EventHandler
          filter:
            module: staking
            method: Rewarded
```

Step 3: Update handleStakingRewarded

In handleStakingRewarded, modify:

```
entity.account = account.toString();
```

to:

```
entity.accountId = account.toString();
```

Because we are effectively creating a relationship or link between our two entities or tables, the StakingReward entity needs to have a column that is the same value as the primary key column in the SumReward entity.

Because the SumReward entity has been assigned the account value (account.toString()), we must do the same here.

The updated mappings file should look like this:

```
import {SubstrateEvent} from "@subql/types";
import {StakingReward, SumReward} from "../types";
import {Balance} from "@polkadot/types/interfaces";

export async function handleStakingRewarded(event: SubstrateEvent):
Promise<void> {
  const {event: {data: [account, newReward]}} = event;
  const entity = new
StakingReward(`${event.block.block.header.number}-${event.idx.toString()
}`);
  entity.accountId = account.toString();
  entity.balance = (newReward as Balance).toBigInt();
  entity.date = event.block.timestamp;
  await entity.save();
}

function createSumReward(accountId: string): SumReward {
  const entity = new SumReward(accountId);
  entity.totalReward = BigInt(0);
  return entity;
}

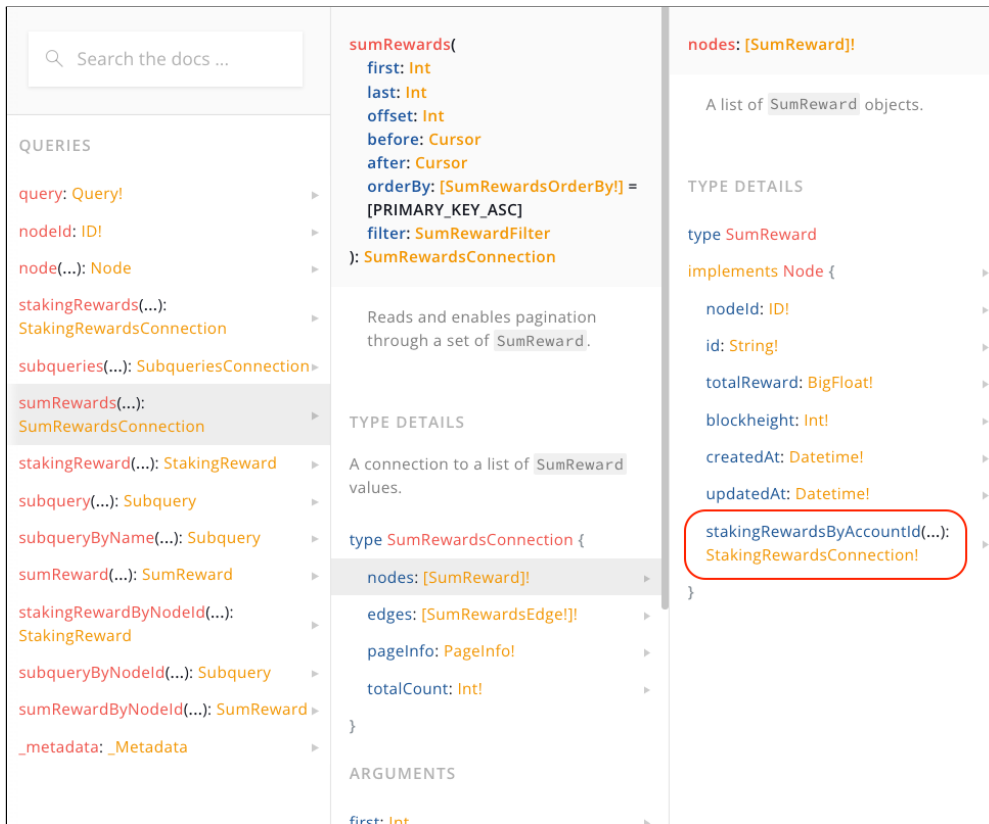
export async function handleSumRewarded(event: SubstrateEvent):
Promise<void> {
  const {event: {data: [accountId, newReward]}} = event;
  let entity = await SumReward.get(accountId.toString());
  if (entity === undefined){
    entity = createSumReward(accountId.toString());
  }
  entity.totalReward = entity.totalReward + (newReward as
Balance).toBigInt();
  entity.blockheight = event.block.block.header.number.toNumber();
  await entity.save();
}
```

Step 4: Rebuild the project

See building a project in the previous exercise.

Step 5: Query the project

Now we can run a query and utilise a `stakingRewardsByAccountId` field that is automatically created in order to find the individual staking rewards.



The screenshot shows the GraphQL IDE interface. On the left is a search bar and a list of queries. The main area displays the schema for the `sumRewards` query, including its arguments and the structure of the returned `SumRewardsConnection` type. The `nodes` field is expanded to show a list of `SumReward` objects, and the `stakingRewardsByAccountId` field is highlighted with a red box.

Below is an example query of one specific account.

```
query{
  sumRewards(filter:
  {id:{equalTo:"16jwQMBXZNxfgXJmVL61gMX4uqtc9WTXV3c8DGx6DUKejm7"}}){
    nodes{
      blockheight
      id
      totalReward
      stakingRewardsByAccountId{
        nodes{
          balance
        }
      }
    }
  }
}
```

The result returned shows that a total reward of 4049635655 is made up of two balances.

```
{
  "data": {
    "sumRewards": {
      "nodes": [
        {
          "blockheight": 7013941,
          "id": "16jWQMBXZNxfgXJmVL61gMX4uqtc9WTXV3c8DGx6DUKejm7",
          "totalReward": "4049635655",
          "stakingRewardsByAccountId": {
            "nodes": [
              {
                "balance": "2189068638"
              },
              {
                "balance": "1860567017"
              }
            ]
          }
        }
      ]
    }
  }
}
```

Exercise 4: Reward v Rewarded

Thus far, we have been using the Rewarded method in the manifest file which was only recently introduced from block [6713249](#) onwards as mentioned earlier. It was previously called Reward so to capture all the staking rewards prior to this change, we need to update our code.

Step 1: Update the manifest file

Add the following mapping filters to the manifest file. Essentially we have removed the “ed” from the handler name and the method.

```
- handler: handleSumReward
  kind: substrate/EventHandler
  filter:
    module: staking
```

```
      method: Reward
    - handler: handleStakingReward
      kind: substrate/EventHandler
      filter:
        module: staking
        method: Reward
```

Also changing the start block to 6,000,000 should result in staking reward data being returned.

When you change the starting block, don't forget to delete the database and reindex.

Step 2: Update the mapping file

Here we can create a redirect function from the old method to utilise the same code as we have already captured the event.

```
export async function handleSumReward(event: SubstrateEvent):
Promise<void> {
  await handleSumRewarded(event)
}

export async function handleStakingReward(event: SubstrateEvent):
Promise<void> {
  await handleStakingRewarded(event)
}
```

Step 3: Rebuild the project

See building a project in the previous exercise.

Step 4: Query the project

Re-run the previous queries and data should appear for blocks starting from 6M. Note, you may have to wait until the relevant blocks have been indexed.

```
query{
  sumRewards(first:3 orderBy:BLOCKHEIGHT_ASC){
    nodes{
      blockheight
      id
      totalReward
    }
  }
}
```

You should see:

```
{
  "data": {
    "sumRewards": {
      "nodes": [
        {
          "blockheight": 6001338,
          "id": "11283CvjWwXesEPQxryZYxjBwTqFV7NMRw8reNGJfzQF4GvS",
          "totalReward": "5068047768"
        },
        {
          "blockheight": 6001338,
          "id": "112EHZp2Dn8jqW9iqpAUFW3ChmiiT6cMnN1arsqJtatnthfz",
          "totalReward": "503936239"
        },
        {
          "blockheight": 6001338,
          "id": "11agCcnJ8cYvKby6p27CiLxBaS1G1hnbRmwtUBAQ3beygUA",
          "totalReward": "1874696285"
        }
      ]
    }
  }
}
```