SubQuery

Making the world's decentralised data more accessible.

Lab Exercise Guide

Table of Contents

Introduction

Here we will take the starter project and focus on understanding how many-to-many relationships work. We will create a project that allows us to query for the number of votes that councillors have made and how many votes a given proposal has received.

To learn more about the Polkadot governance structure, please refer to: https://polkadot.network/blog/a-walkthrough-of-polkadots-governance/

Pre-requisites

Completion of Module 2

Exercise 1: Council Proposals (many-to-many)

High-level steps

1. Initialise the starter project
2. Update your mappings, manifest file and graphql schema file by removing all the default code except for the handleEvent function.
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$ subql init
Project name [subql-starter]: council-proposals
Git repository:
RPC endpoint [wss://polkadot.api.onfinality.io/public-ws]:
Authors: sa
Description:
Version: [1.0.0]:
License: [Apache-2.0]:
Init the starter package... council-proposals is ready
```

Step 2: Update the graphql schema

Let's first create an entity called "Proposals". This proposal is an event of type council. In other words, we are interested in extracting data from the council event. Visit https://polkadot.js.org/docs/substrate/events#council for more information.

Within the council event, we are going to focus on the "proposed" method. The proposed method is defined as:

" A motion (given hash) has been proposed (by given account) with a threshold (given MemberCount). [account, proposal_index, proposal_hash, threshold]" - source

We can therefore add the following fields: id, index, hash, voteThreshold and block to our entity.

id => *account*
index => *proposal_index*
hash => *proposal_hash*
voteThreshold => *threshold*
block => *Not part of proposed method but useful to extract*

Next, let's create an entity object called Councillor. This object will simply hold the number of votes each councillor has made. This can be thought of as a simple table like below:

	id [PK] text	number_of_votes integer
1	128qRiVjxU3TuT3...	2
2	12NLgzqfhuJkc9...	22
3	12RYJb5gG4hfo...	1

Finally, let's create a VoteHistory entity. This will be another council event using the voted method defined as:

"A motion (given hash) has been voted on by a given account, leaving a tally (yes votes and no votes given respectively as MemberCount). [account, proposal_hash, voted, yes, no]"

We can therefore add the following fields: id, proposalHash, approvedVote, councillor, votedYes, votedNo, and block to our entity.

id => *account*
proposalHash => Proposal
approvedVote => *voted*
Councillor => Councillor
votedYes => *yes*
votedNo => *no*
block => *Not part of proposed method but useful to extract*

Note that for proposalHash, we are specifying the type as the proposal entity. We also introduced a new field called Councillor and gave that a type of Councillor. What this has effectively done is created a table where these two columns are references to their respective tables.

This means that the VoteHistory entity or VoteHistory database table can link the Councillor entity to the Proposal entity thereby creating what can be considered as a many to many relationship.

A councillor can vote for many proposals and a proposal will have many votes is effectively what this all means.

The schema file should look like this:

```
type Proposal @entity {
  id: ID!
  index: String!
  account: String
  hash: String
  voteThreshold: String
  block: BigInt
}

type VoteHistory @entity {
  id: ID!
  proposalHash: Proposal
  approvedVote: Boolean!
  councillor: Councillor
  votedYes: Int
  votedNo: Int
  block: Int
}

type Councillor @entity {
  id: ID!
  numberOfVotes: Int
}
```

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

Update the manifest file to only include two Event handlers and update the filter method to council/Proposed and council/Voted.

```yaml
specVersion: 0.0.1
description: ''
repository: ''
schema: ./schema.graphql
network:
  endpoint: wss://polkadot.api.onfinality.io/public-ws
  dictionary:
https://api.subquery.network/sq/subquery/dictionary-polkadot
dataSources:
  - name: main
    kind: substrate/Runtime
    startBlock: 1
    mapping:
      handlers:
        - handler: handleCouncilProposedEvent
          kind: substrate/EventHandler
          filter:
            module: council
            method: Proposed
        - handler: handleCouncilVotedEvent
          kind: substrate/EventHandler
          filter:
            module: council
            method: Voted
```

Step 4: Update the mappings file

handleCouncilProposedEvent

This mappings file will contain three functions. Let's call the first function "handleCouncilProposedEvent".

We can access the values of the event with the following code:

```
  const {
    event: {
      data: [accountId, proposal_index, proposal_hash, threshold],
    },
  } = event;
```

Then we instantiate a new Proposal object,

```
  const proposal = new Proposal(proposal_hash.toString());
```

and then assign each of the events to a variable in the Proposal object and save it.

```
  proposal.index = proposal_index.toString();
  proposal.account = accountId.toString();
  proposal.hash = proposal_hash.toString();
  proposal.voteThreshold = threshold.toString();
  proposal.block = event.block.block.header.number.toBigInt();
  await proposal.save();
```

handleCouncilVotedEvent

This function follows a similar format of handleCouncilProposedEvent from above. The event parameters are first obtained,

```
const {
    event: {
      data: [councilorId, proposal_hash, approved_vote, numberYes,
numberNo],
    },
  } = event;
```

but before storing the values into the voteHistory object, a helper function is used to check if the councillorId already exists.

```
  await ensureCouncillor(councilorId.toString());
  // Retrieve the record by the accountID
  const voteHistory = new VoteHistory(
    `${event.block.block.header.number.toNumber()}-${event.idx}`
  );
```

ensureCouncillor (helper function)

This helper function checks if the councillor entity exists. If it does NOT exist, a new one is created and the number of votes is set to zero. Otherwise, the number of votes is incremented by one.

```
async function ensureCouncillor(accountId: string): Promise<void> {
  // ensure that our account entities exist
  let councillor = await Councillor.get(accountId);
  if (!councillor) {
    councillor = new Councillor(accountId);
    councillor.numberOfVotes = 0;
  }
  councillor.numberOfVotes += 1;
  await councillor.save();
```

The complete mapping files look like the following:

```
import { SubstrateEvent } from "@subql/types";
import { bool, Int } from "@polkadot/types";
import { Proposal, VoteHistory, Councillor } from "../types/models";

export async function handleCouncilProposedEvent(event: SubstrateEvent):
Promise<void> {
  const {
    event: {
      data: [accountId, proposal_index, proposal_hash, threshold],
    },
  } = event;
  const proposal = new Proposal(proposal_hash.toString());
  proposal.index = proposal_index.toString();
  proposal.account = accountId.toString();
```

```
  proposal.hash = proposal_hash.toString();
  proposal.voteThreshold = threshold.toString();
  proposal.block = event.block.block.header.number.toBigInt();
  await proposal.save();
}

export async function handleCouncilVotedEvent(event: SubstrateEvent):
Promise<void> {
  // logger.info(JSON.stringify(event.event.data));
  const {
    event: {
      data: [councilorId, proposal_hash, approved_vote, numberYes,
numberNo],
    },
  } = event;

  await ensureCouncillor(councilorId.toString());
  // Retrieve the record by the accountID
  const voteHistory = new VoteHistory(
    `${event.block.block.header.number.toNumber()}-${event.idx}`
  );
  voteHistory.proposalHashId = proposal_hash.toString();
  voteHistory.approvedVote = (approved_vote as bool).valueOf();
  voteHistory.councillorId = councilorId.toString();
  voteHistory.votedYes = (numberYes as Int).toNumber();
  voteHistory.votedNo = (numberNo as Int).toNumber();
  voteHistory.block = event.block.block.header.number.toNumber();
  // logger.info(JSON.stringify(voteHistory));
  await voteHistory.save();
}

async function ensureCouncillor(accountId: string): Promise<void> {
  // ensure that our account entities exist
  let councillor = await Councillor.get(accountId);
  if (!councillor) {
    councillor = new Councillor(accountId);
    councillor.numberOfVotes = 0;
  }
  councillor.numberOfVotes += 1;
  await councillor.save();
}
```

Step 5: Generate the associated typescript

Next, we will generate the associated typescript with the following command:

```
yarn codegen
```

OR

```
npm run-script codegen
```

Step 6: Build the project

The next step is to build the project with the following command:

```
yarn build
```

OR

```
npm run-script build
```

This bundles the app into static files for production.

Step 7: Start the Docker container

Run the docker command to pull the images and to start the container.

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

Step 8: Run a query

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 {
    councillors (first: 3 orderBy: NUMBER_OF_VOTES_DESC) {
        nodes {
            id
            numberOfVotes
            voteHistory (first: 5) {
              totalCount
              nodes {
                approvedVote
              }
          }
        }
    }
}
```

This will query the councillors, and for each councillor return the number of votes and for each councillor also return the totalCount and the number of approved votes as can be seen below.


```
{
  "data": {
    "councillors": {
      "nodes": [
        {
          "id": "12hAtDZJGt4of3m2GqZcUCVAjZPALfvPwvtUTFZPQUbdX1Ud",
          "numberOfVotes": 61,
          "voteHistory": {
            "totalCount": 61,
            "nodes": [
              {
                "approvedVote": true
              },
              {
                "approvedVote": true
              },
              {
                "approvedVote": true
              },
              {
                "approvedVote": true
              },
              {
```

```
            "approvedVote": true
          }
        ]
      }
    },
    {
      "id": "1363HWTPzDrzAQ6ChFiMU6mP4b6jmQid2ae55JQcKtZnpLGv",
      "numberOfVotes": 60,
      "voteHistory": {
        "totalCount": 60,
        "nodes": [
          {
            "approvedVote": true
          },
          {
            "approvedVote": true
          },
          {
            "approvedVote": true
          },
          {
            "approvedVote": true
          },
          {
            "approvedVote": true
          }
        ]
      }
    },
    {
      "id": "12NLgzqfhuJkc9mZ5XUTTG85N8yhhzfptwqF1xVhtK3ZX7f6",
      "numberOfVotes": 56,
      "voteHistory": {
        "totalCount": 56,
        "nodes": [
          {
            "approvedVote": true
          },
          {
            "approvedVote": true
          },
          {
            "approvedVote": true
          },
          {
            "approvedVote": true
          },
          {
            "approvedVote": true
```

```
              },
              {
                "approvedVote": true
              }
            ]
          }
        }
      ]
    }
  }
}
```

Bonus

Including a reverse lookup on the schema file will allow us to customise the fields that we can query on.

```
type Proposal @entity {
  id: ID!
  index: String!
  account: String
  hash: String
  voteThreshold: String
  block: BigInt
  voteHistory_p: [VoteHistory] @derivedFrom(field: "proposalHash")
}

type VoteHistory @entity {
  id: ID!
  proposalHash: Proposal
  approvedVote: Boolean!
  councillor: Councillor
  votedYes: Int
  votedNo: Int
  block: Int
}

type Councillor @entity {
  id: ID!
  numberOfVotes: Int
  voteHistory_c: [VoteHistory] @derivedFrom(field: "councillor")
}
```

By adding voteHistory_p and voteHistory_b, voteHistories becomes voteHistory_c for example in the screenshot below.