



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

Lab Exercise Guide

Table of Contents

Introduction	2
Pre-requisites	2
Exercise 1: Account Transfers (1-to-many)	2
High level steps	2
Detailed steps	2
Step 1: Clone Account Transfer project	2
Step 2: Confirm the project works.	2
Step 3: Add a reverse lookup	4
Step 4: Recompile and test	5

Introduction

In this lab, we will take the starter project and focus on understanding what reverse lookups are.

Pre-requisites

Completion of Module 3: Lesson 2 - One to many entities.

Exercise 1: Account Transfers with reverse lookup

High level steps

1. Git clone the tutorials-account-transfers project
2. Run it to ensure it is working
3. Add a new field in the schema and make it a reverse lookup
4. Requery the project with this new “virtual” field as a reverse look up

Detailed steps

Step 1: Clone Account Transfer project

```
git clone https://github.com/subquery/tutorials-account-transfers
```

Start by cloning the “tutorials-account-transfers” Github repository. This was the exercise for Module 3 - Lesson 2.

Step 2: Confirm the project works.

Run the basic commands to get the project up and running.

```
yarn install  
yarn codegen  
yarn build  
docker-compose pull && docker-compose up
```

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{
  accounts(first: 3){
    nodes{
      id
    }
  }
}
```

This will query the account entity returning the id. We have defined the id here as the “toAddress”, otherwise known as the receiving address. This will return the following:

```
{
  "data": {
    "accounts": {
      "nodes": [
        {
          "id": "11k5Gkwb9npuqWRq5Pyk51RSnRyskPrPtsyoCApteEUjNou"
        },
        {
          "id": "121dZJsfG7uNvszPSpYvBzwnrcF1P4ejjrE1G6FSWHqht5tC"
        },
        {
          "id": "121rwkQAH3yCD1EcaRgc3nELSoZn29RoTtCN55mcN7RkBA66"
        }
      ]
    }
  }
}
```

As noted in a previous exercise, the magic lies in the ability to query the account id from within the transfer entity. The example below shows that we are querying for transfers where we have an associated amount and blockNumber, but we can then link this to the receiving or “to” address as follows:

```
query{
  transfers(first: 3){
    nodes{
      id
      amount
      blockNumber
      to{
        id
      }
    }
  }
}
```

```
}  
}  
}  
}
```

The query above returns the following results:

```
{  
  "data": {  
    "transfers": {  
      "nodes": [  
        {  
          "id": "7280565-2",  
          "amount": "400009691000",  
          "blockNumber": "7280565",  
          "to": {  
            "id": "15kUt2i86LHRWCKE3D9Bg1HZAoc2smhn1fwPzDERTb1BXAkX"  
          }  
        },  
        {  
          "id": "7280566-2",  
          "amount": "23174700000000",  
          "blockNumber": "7280566",  
          "to": {  
            "id": "14uh77yjhC3TLAE6KaCLvkjN7yFeUkejm7o7fdaSsggwD1ua"  
          }  
        },  
        {  
          "id": "7280567-2",  
          "amount": "34192690000000",  
          "blockNumber": "7280567",  
          "to": {  
            "id": "12sj9HTNQ7aiQoRg5wLyuemgvmFcrWeUJRi3aEUnJLmAE56Y"  
          }  
        }  
      ]  
    }  
  }  
}
```

Step 3: Add a reverse lookup

Add an extra field to the Account entity called myToAddress. Make this of type Transfer and add the `@derived` annotation. This is making a “virtual field” called myToAddress that can be accessed from the Account entity. It is virtual because the database table structure does not actually change.

- Allows you to do a reverse lookup in GraphQL
- Adds a `GetElementByID()` on the child entities

```
type Account @entity {
  id: ID! #this primary key is set as the toAddress
  myToAddress: [Transfer] @derivedFrom(field:"to")
}

type Transfer @entity {
  id: ID! #this primary key is the block number + the event id
  amount: BigInt
  blockNumber: BigInt
  to: Account! #receiving address
}
```

Step 4: Recompile and test

```
query{
  accounts(first:5){
    nodes{
      id
      myToAddress{
        nodes{
          id
          amount
        }
      }
    }
  }
}
```

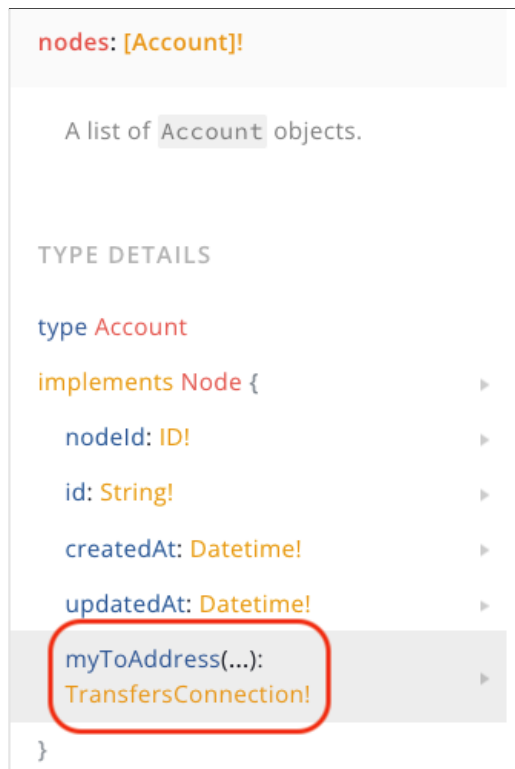
You should get something similar to the following:

```
{
  "data": {
    "accounts": {
```

```
"nodes": [
  {
    "id": "1112NRMkvMb5x3EwGsLSzXyw7kSLxug4uFH1ec3CnDe7ZoG",
    "myToAddress": {
      "nodes": [
        {
          "id": "1206531-14",
          "amount": "123000000000"
        },
        {
          "id": "1206533-9",
          "amount": "30000000000"
        },
        {
          "id": "1249840-2",
          "amount": "100000000000"
        }
      ]
    }
  },
  {
    "id": "1117zZ65F4sz3EH9hZdAivERch99XMXADHicJn7ZmKUrrxT",
    "myToAddress": {
      "nodes": [
        {
          "id": "1256968-5",
          "amount": "86880000000"
        },
        {
          "id": "1256984-5",
          "amount": "12299500000000"
        }
      ]
    }
  },
  {
    "id": "11212d8rV4pj73RLoXqiEJweNs2qU1SsfwbzzRWVzn2o5ZCt",
    "myToAddress": {
      "nodes": [
        {
          "id": "1212424-9",
          "amount": "50000000000"
        },
        {
          "id": "1212680-3",
          "amount": "150000000000"
        }
      ]
    }
  }
]
```

```
    },
    {
      "id": "1212719-3",
      "amount": "22622363200000"
    },
    {
      "id": "1240252-2",
      "amount": "41055764800000"
    },
    {
      "id": "1258672-6",
      "amount": "49000000000"
    }
  ]
}
]
```

Adding the `@derivedFrom` keyword to the `myToAddress` field allows a “virtual” field to appear in the `Account` object. This can be seen in the documentation tab. This allows a “reverse lookup” where the `Transfer.to` field can be accessed from `Account.myToAddress`.



The screenshot shows a documentation panel for a GraphQL schema. At the top, it says `nodes: [Account]!` with a description "A list of `Account` objects." Below this is a section titled "TYPE DETAILS" for the `Account` type. It lists that `Account` implements the `Node` interface. The fields listed are `nodeId: ID!`, `id: String!`, `createdAt: Datetime!`, `updatedAt: Datetime!`, and `myToAddress(...): TransfersConnection!`. The `myToAddress(...)` field is highlighted with a red rounded rectangle, indicating its significance in the context of the text above.