

Rel Basics

Chromia
Academy





Introduction to Rell

- Rell is a programming language that makes it possible to combine the features of the relational database and blockchain.
- With Rell, the developer can create a database and put it into the Chromia platform.
- The Rell syntax is interpreted down into Kotlin/SQL.






Download PostgreSQL

- First, we need to have PostgreSQL as the database manager
- Go to <https://www.postgresql.org/download/> and download the latest version.
- Make sure that pgAdmin is installed alongside PostgreSQL since we will be using pgAdmin in this tutorial, if you are familiar with PostgreSQL this step will be optional.

Create postchain role in postgres

-> Login/Group Role



 **Create - Login/Group Role**

General

Definition

Privileges

Membership


Name

postchain

Give postchain privileges in the definitions tab

Set the password to anything to your liking.

The default in Rell projects is 'postchain'

 **Create - Login/Group Role**

General

Definition

Privileges

Can login?

☒

Superuser?

☒

Create roles?

☒

Create databases?

☒

Inherit rights from the parent roles?

☒

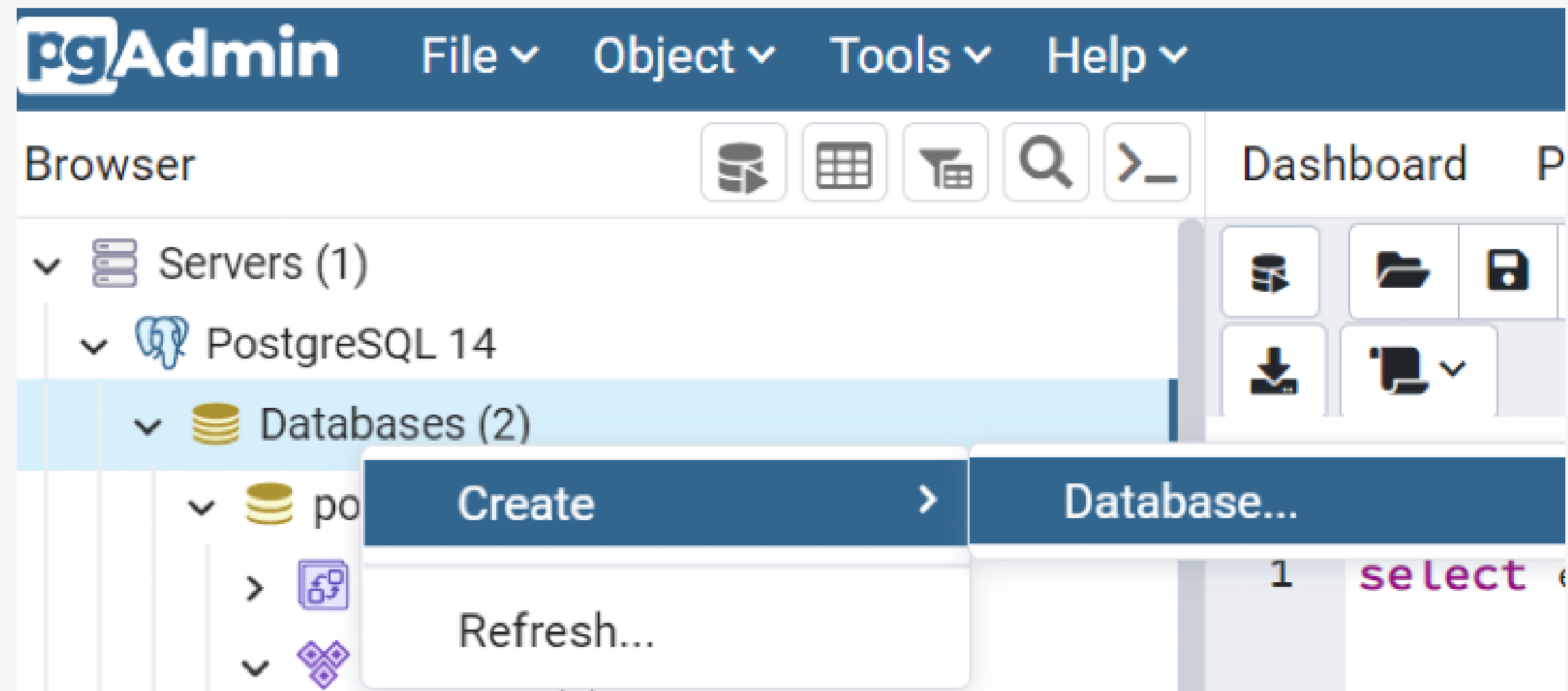
Can initiate streaming replication and backups?

☐

●●●

Create a Database in pgAdmin

Select Databases -> Create -> Database





Create a Database in pgAdmin

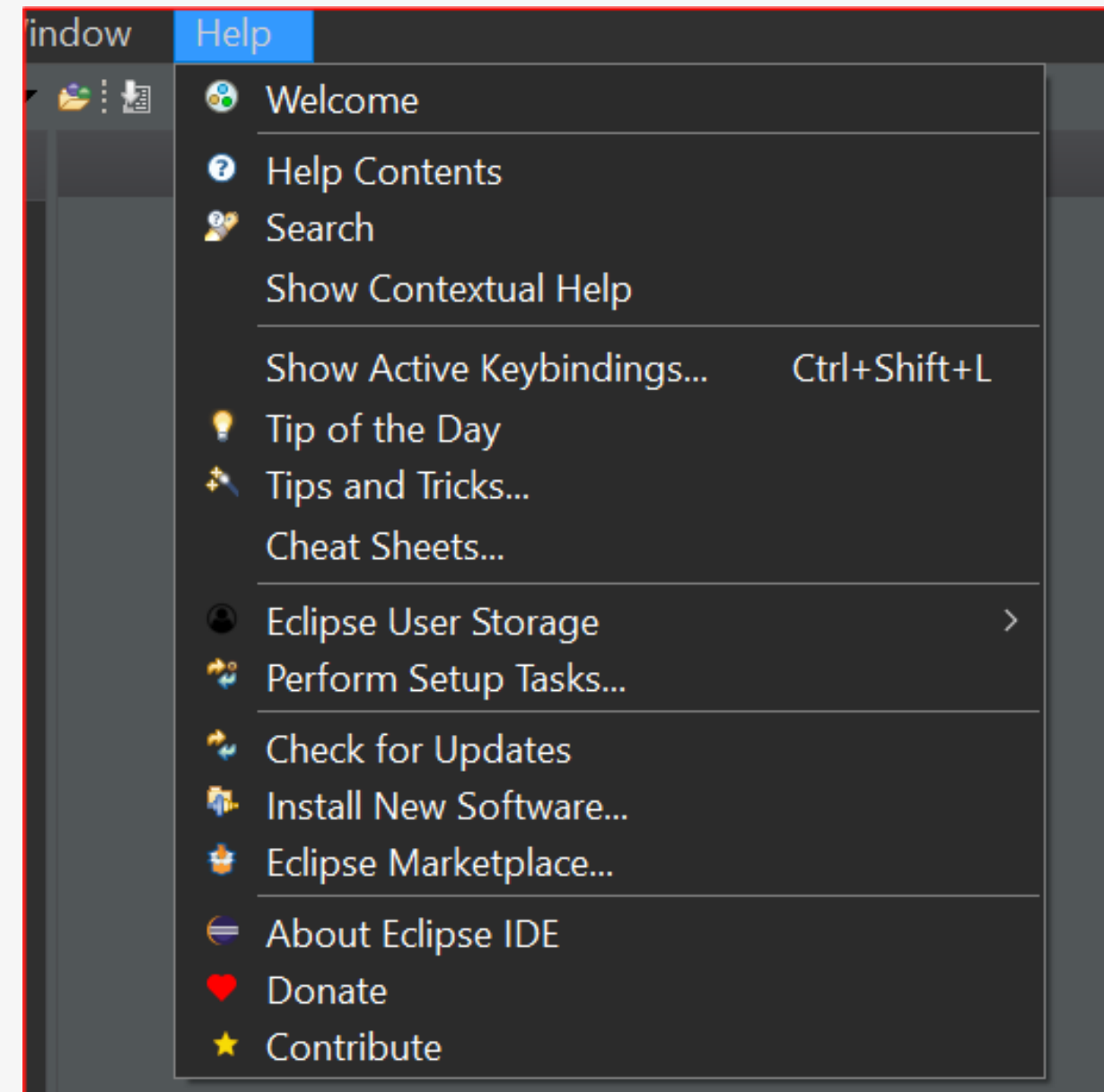
The screenshot shows the 'Create - Database' dialog box in pgAdmin. It has a title bar with a database icon and the text 'Create - Database'. Below the title bar are tabs for 'General', 'Definition', 'Security', 'Parameters', 'Advanced', and 'SQL'. The 'General' tab is selected. In the 'General' tab, there are two fields: 'Database' and 'Owner'. The 'Database' field contains the text 'postchain'. The 'Owner' field contains a blue user icon followed by the text 'postchain' and a dropdown arrow on the right.

If you have created the postchain role, set it to the owner of the new database. Click save and your database should be up and running!



In Eclipse

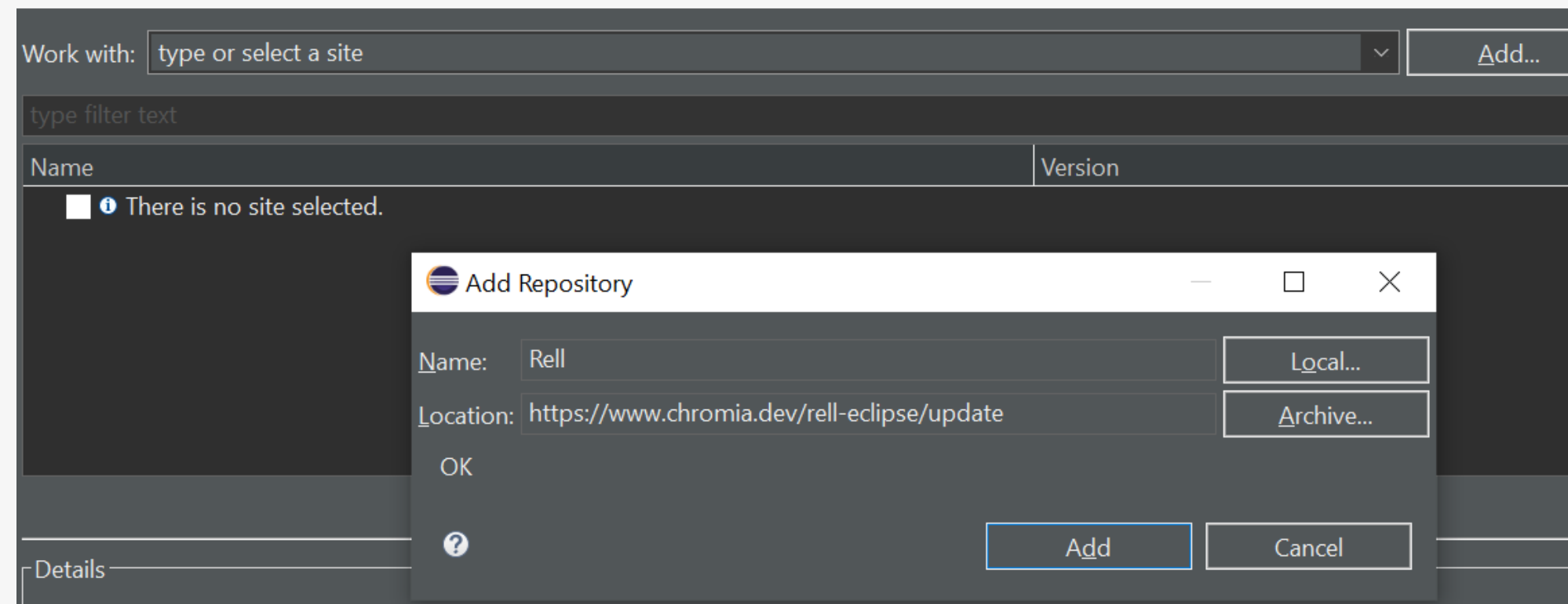
1. Go to the Help tab
2. Click on: Install new software





Add Rell Plugin

Click on Add and Have Rell as the name and <https://www.chromia.dev/rell-eclipse/update> as location and add.

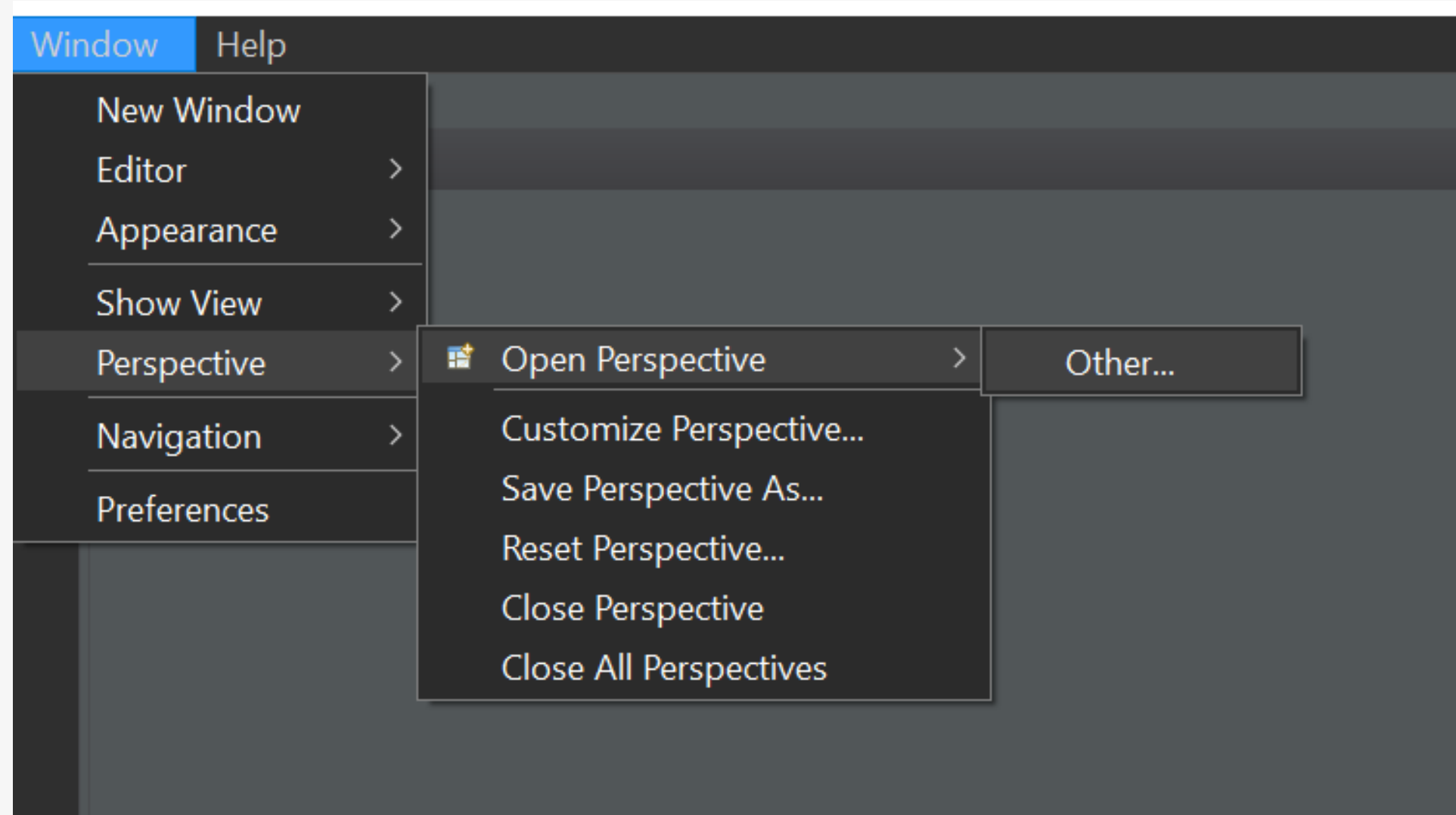


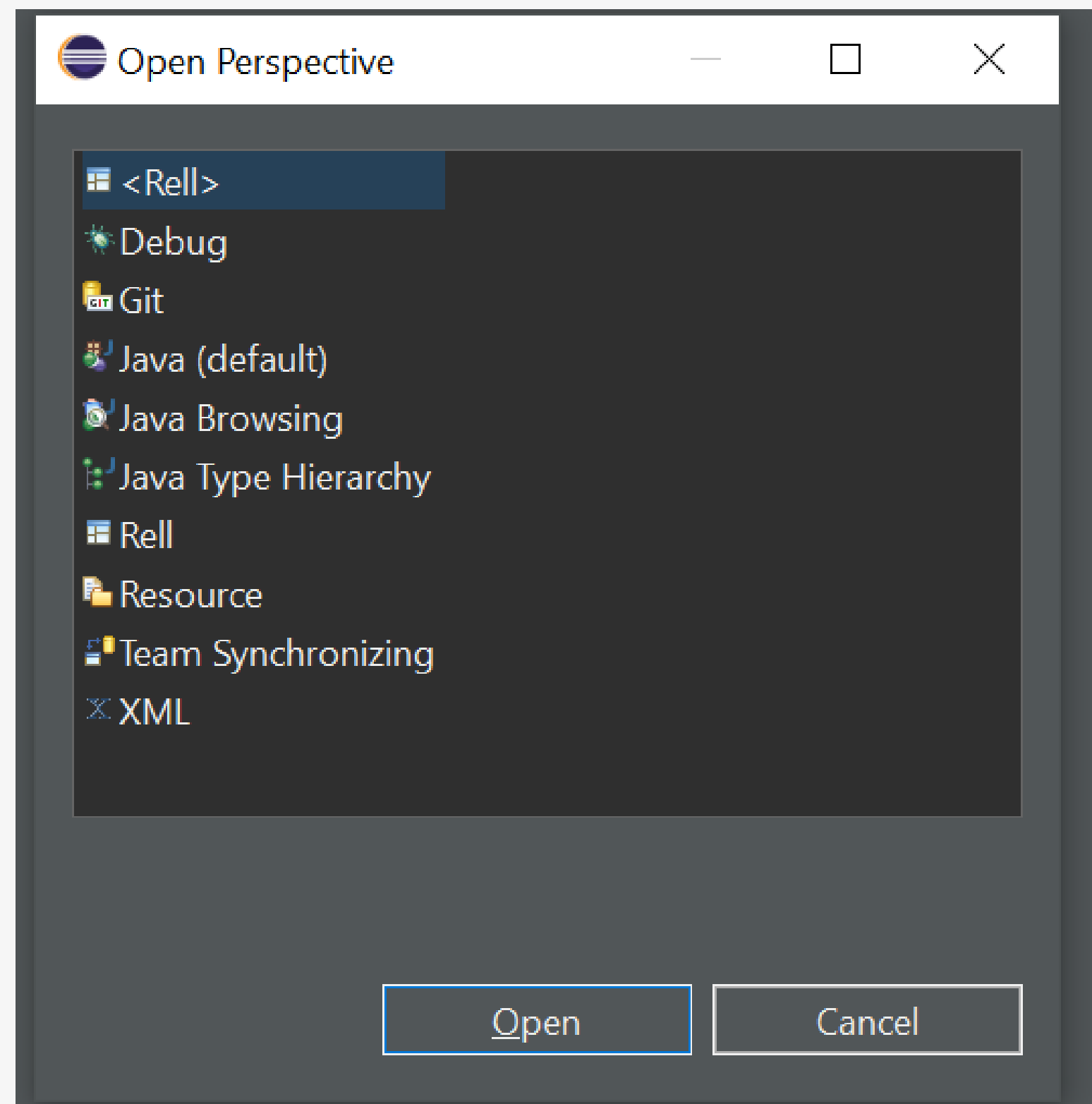
Check the box next to Rell, then click on next and finish.

●●●

Changing to a Rel perspective

Go to window → Perspective → Open Perspective → Other

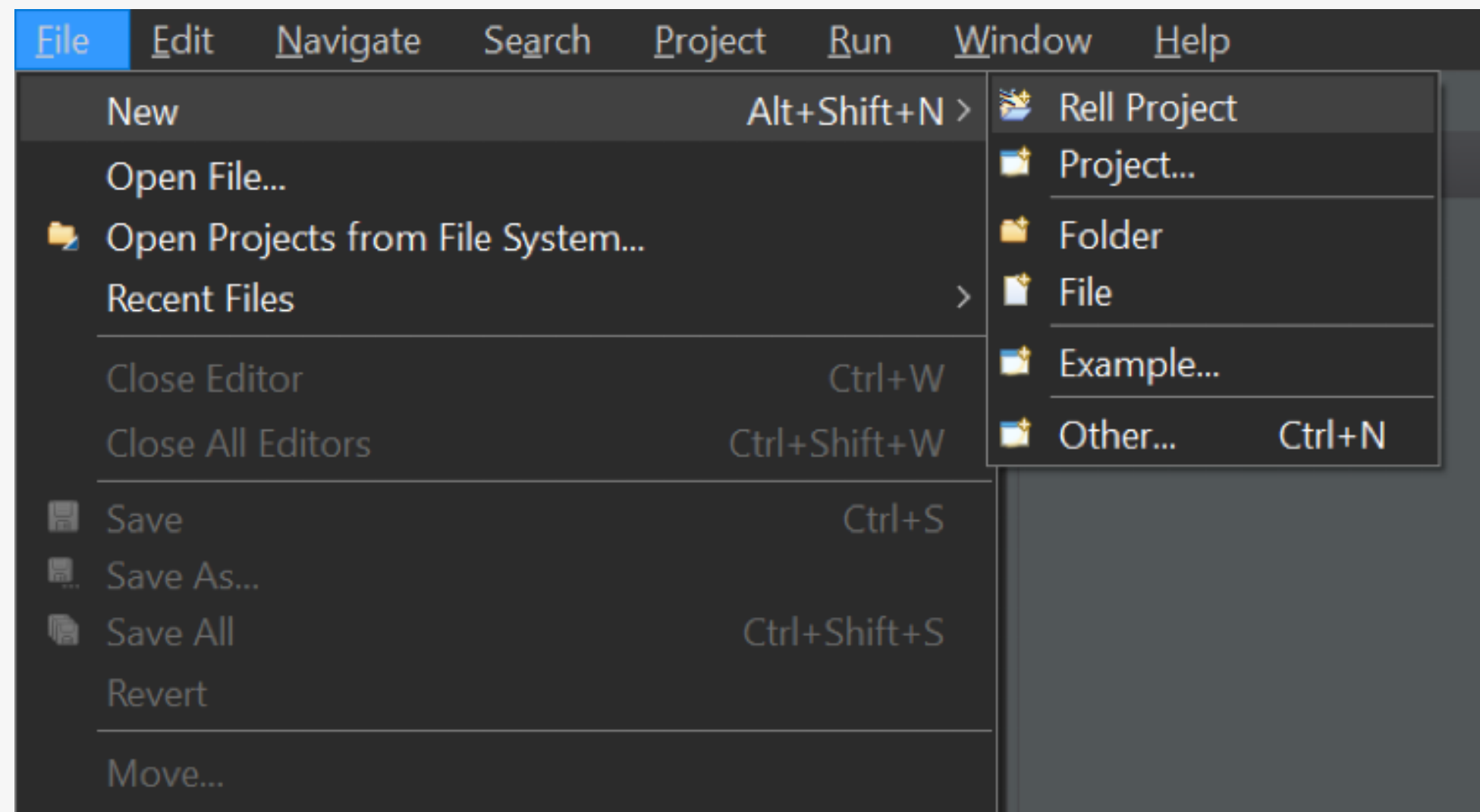






Open a Rell Project

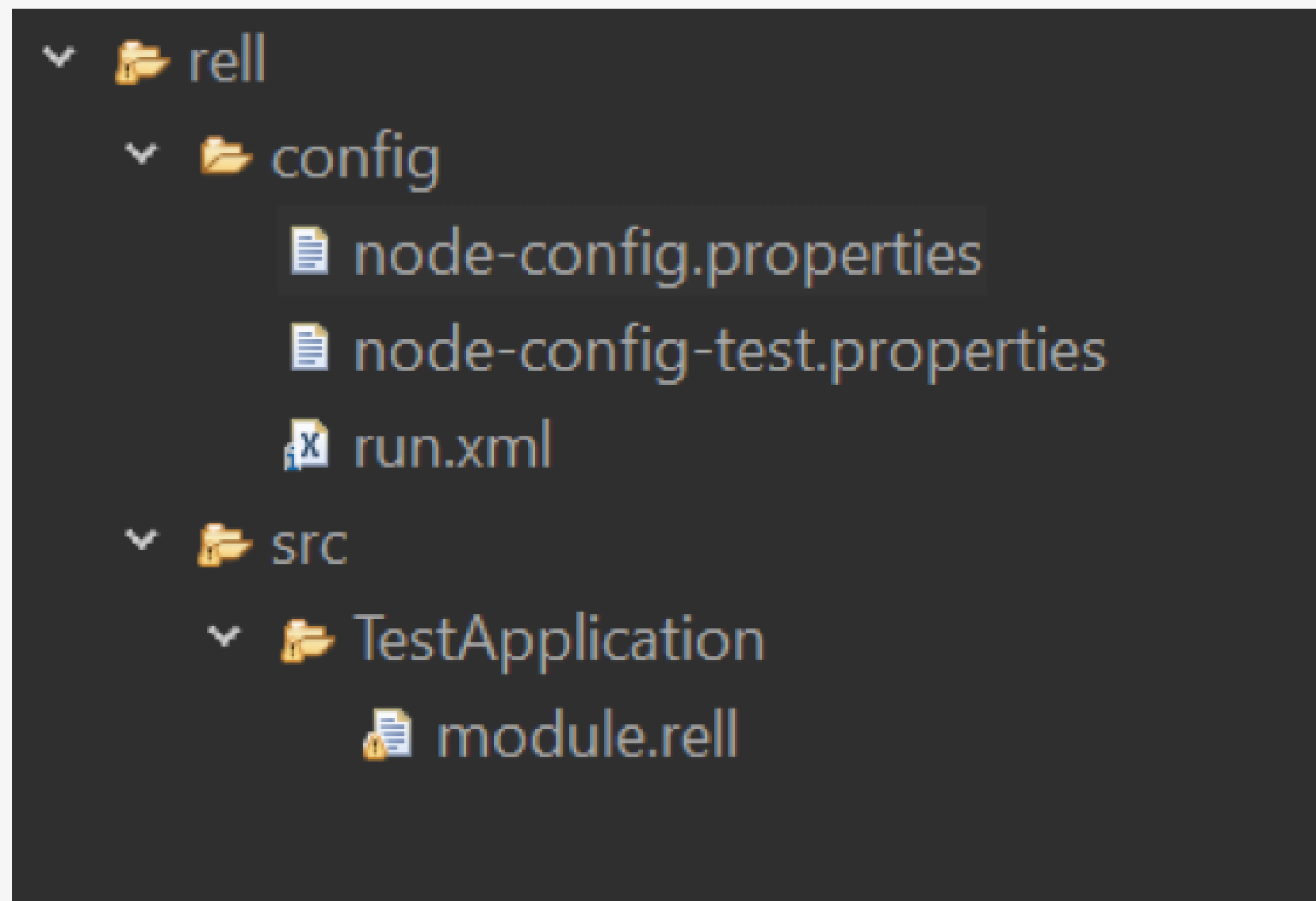
Now we can open a Rell project





Config Properties

In the `node-config.properties` file, you can change the user logging into the Postgres database and what port the client will listen to.





Entity

In Rell, an Entity is the data model we use in the database

The key identifier means that the attribute is unique for each instance of the entity.

```
4  
5 entity dog {  
6  
7     key ID : integer;  
8     name : text;  
9  
10 }  
11
```

•••

Operations and queries

- Operation is used when we want to modify data in the database

```
• operation add_dog( dog_ID: integer, dog_name: text, pubkey){  
    require(is_signer(pubkey));  
  
    create dog(ID = dog_ID, name = dog_name );  
}
```

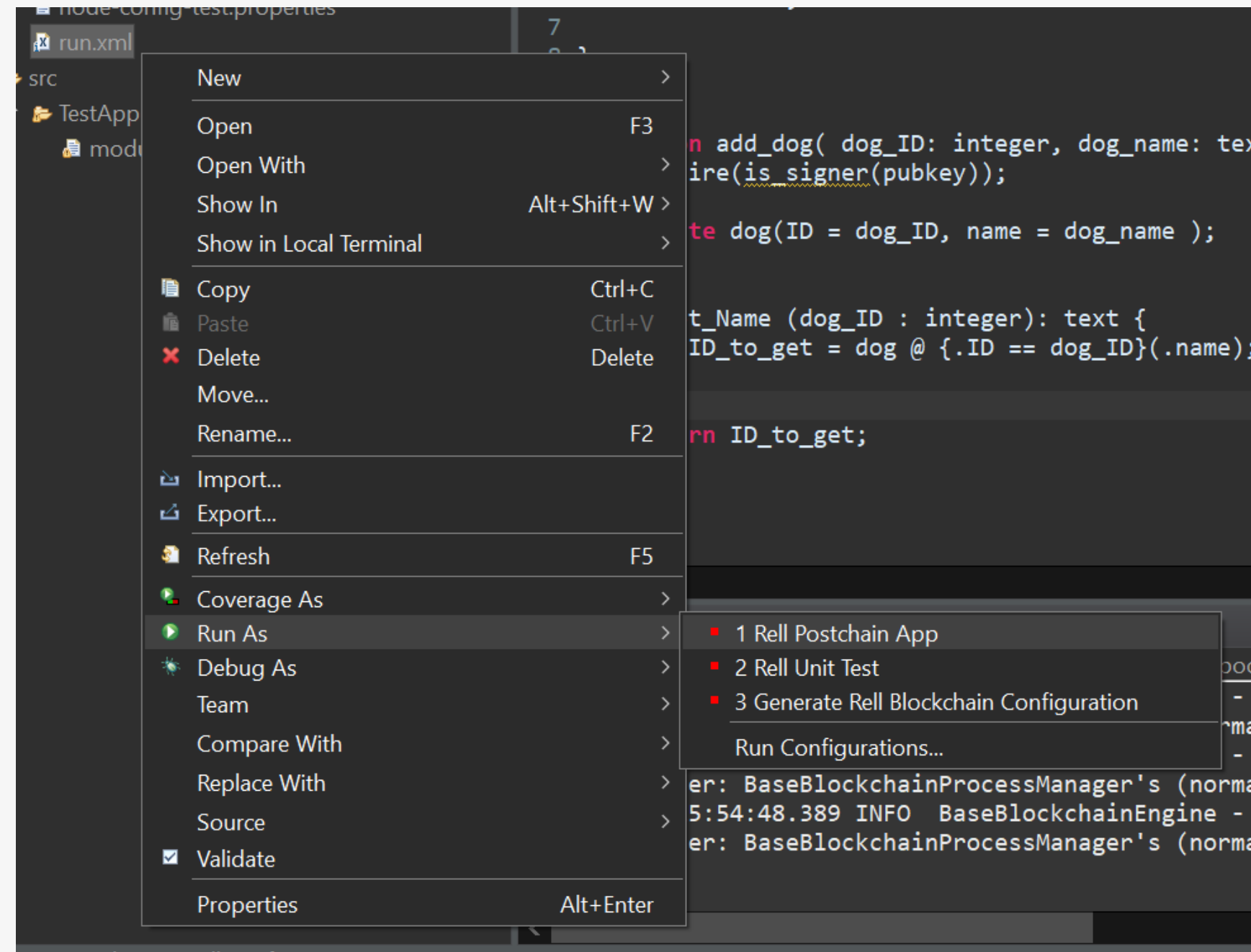
- Query is used when we want to retrieve data from the database

```
• query get_Name (dog_ID : integer): text {  
    val ID_to_get = dog @ { .ID == dog_ID }( .name );  
  
    return ID_to_get;  
}
```

●●●

Running the Rell program

- Right-click on the run.xml file in your eclipse project, from there, choose
- Run As -> Rell Postchain App
- You now have a node up and running, now we will write a clientside program to communicate with it.





Clientside

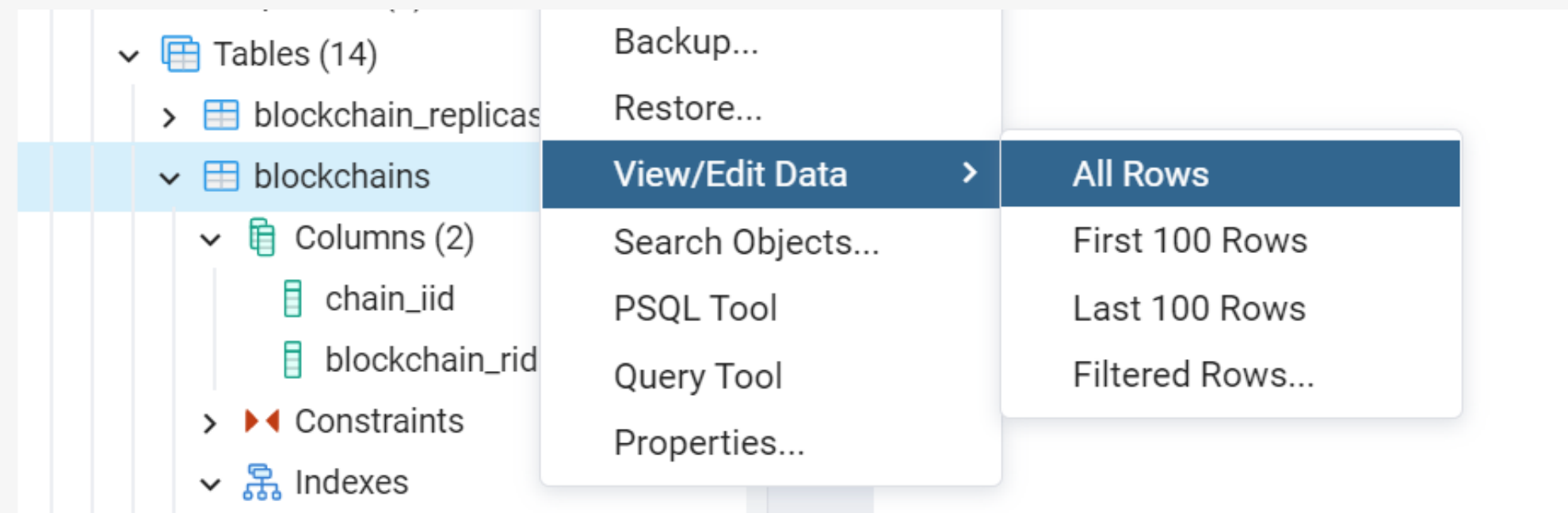
- Download the package 'postchain-client' using npm
- With the command, npm install 'postchain-client'
- This also requires you to have node js installed

```
npm install 'postchain-client'
```

●●●

Finding the BRID

- Now, this next part will be to query our database inside pgadmin, we want to find the BRID to paste into our client, but to do that we have to find it in the database.

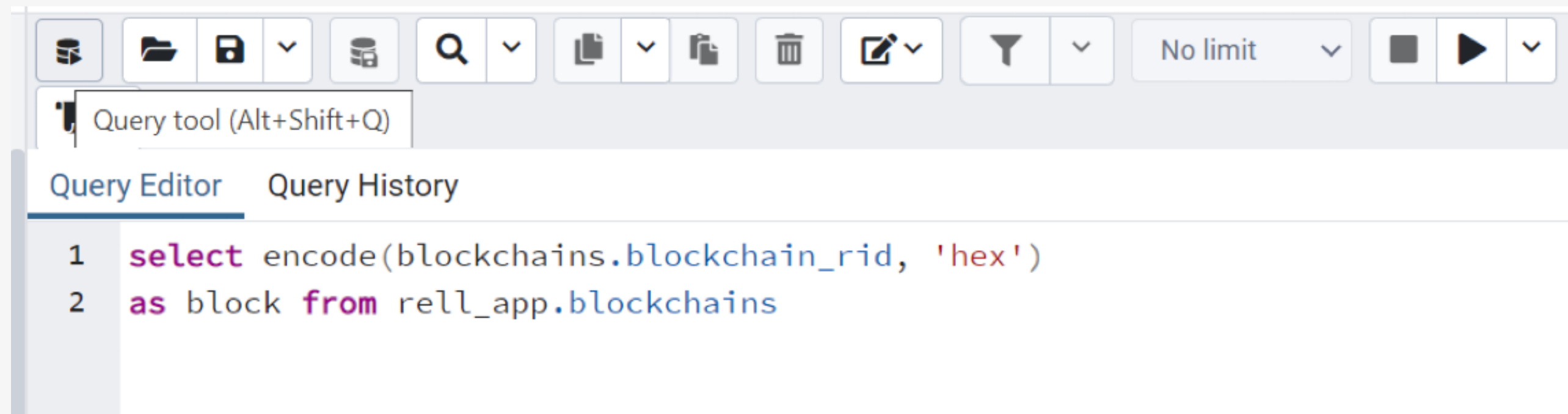


- Start by going into the postchain database
- in pgAdmin,
- select schemas >rell_app -> tables -> blockchains. Right-click on blockchains and select View/Edit Data -> All Rows

● ● ●

Finding the BRID

- We will now click on the query tool and type in the following query:



- Press the play button in the top right corner to perform the query, the BRID will be presented below the editor



Connect client to database

- Open a text editor for javascript and put this piece of code at the beginning of the file to connect to the database. Replace the blockchainRID with the BRID you received from the task in the previous slide.

```
const pcl = require('postchain-client');

const node_api_url = "http://localhost:7740";

const blockchainRID = "61c263954042fa1474a4c9a9bc5a47496932e0ae48a757b1ce4c45df17d2715c";

const rest = pcl.restClient.createRestClient(node_api_url, blockchainRID, 5);

const gtx = pcl.gtxClient.createClient(
  rest,
  Buffer.from(blockchainRID, 'hex'),
  []
);
```



Calling on operation `add_dog` in the client

```
16  async function createDog (ID, name) {  
17  
18    const user = pcl.util.makeKeyPair();  
19    const pubKey = pcl.util.toBuffer(user.pubKey);  
20  
21    const tx = gtx.newTransaction([user.pubKey]);  
22    tx.addOperation("add_dog", name, pubKey);  
23    tx.sign(user.privKey, user.pubKey);  
24    await tx.postAndWaitConfirmation();  
25  
26  }
```

All operations require that we sign the transaction, we, therefore, create a key pair.

With (general transaction client) GTX we create a new transaction which we later sign when we are finished with the operations.



Calling on query get_Name in the client

When we call on the query in the client we have to define which parameter in the client corresponds to the parameter in the Rel query.

```
28  async function get_Name(ID){  
29      return gtx.query("get_Name", {dog_ID : ID})  
30  
31  }  
32
```



Use the created functions

We can now run the program with this simple implementation.

```
async function test() {  
  
  let ID = 100;  
  let name = "Laika";  
  
  await createDog(ID, name);  
  console.log(await getName(ID));  
}  
  
test();
```