The “Unlocking Digital Twins” GitHub repo

The Unlocking Digital Twins (UDT) repo is publicly accessible on the Sandtech-EnterpriseAI github account. This is available at the following link: https://github.com/Sand-EnterpriseAI/udt-clean-water-toolkit

Unlocking Digital Twins - What is it?

This project is a Proof-of-Concept (PoC) for a clean water toolkit that combines aspects of a digital twin with clean water modelling and analysis. The project was funded by Ofwat in collaboration with Thames Water and Severn Trent Water.

The core innovation of this toolkit is a performant and efficient algorithm for transforming water distribution network data from a traditional geospatial format into a rich graph-based data structure. This unlocks new analytical capabilities that are not easily achievable with standard GIS models.

Contained in the repository

The toolkit is organised into two primary components

- `cwm`: A core Python library for data transformation, network analysis, and modelling.
- `cwa`: A Django-based application that wraps the cwm module and exposes an API for digital twin interactions.

Technology stack

- All required precursors, and packages are accomodate for in the guided walk-through, including
  - `docker-compose.yml` - Used to start up services
  - `requirements.txt`, `dev-requirements.txt` - called for installing dependencies
- The technology stack includes Python, Django/GeoDjango, PostgreSQL/PostGIS, Neo4j, Docker, NetworkX, GeoPandas, and Momepy.
- The repository is licensed under the GNU General Public License version 3

Installing the application

▼ We start by cloning from github

- Make sure you have Git and Docker installed. These can respectively be found from https://git-scm.com/downloads and https://www.docker.com/get-started/.
- Open your terminal (Command Prompt, PowerShell, or Terminal on macOS/Linux). All illustrations will be shown from a mac for the purpose of this walk-through.

- We attempt to run through the first step in the github repository. This will create a folder named `udt-clean-water-toolkit` in your current directory with all the repository files.

1. **Clone the Repository**

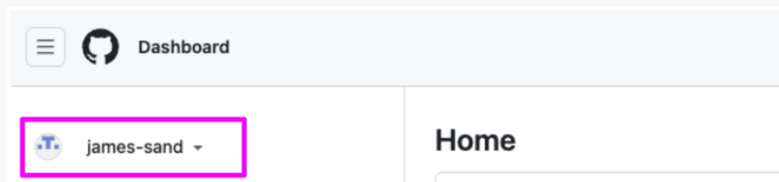
```
git clone https://github.com/Explore-AI/udt-clean-water-toolkit.git
cd udt-clean-water-toolkit
```

🛠️ Option 1: Using HTTPS protocol

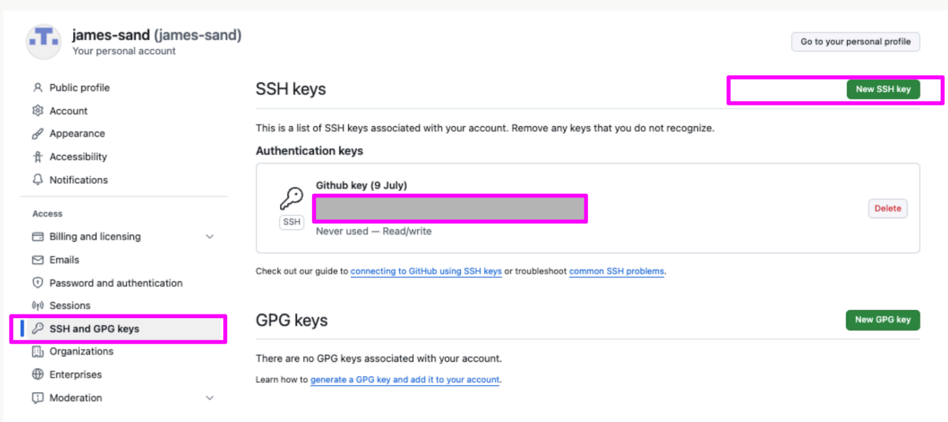
- `git clone https://github.com/Sand-EnterpriseAI/udt-clean-water-toolkit.git`
- Requires you to authenticate with a GitHub username and password (or a personal access token, since passwords are deprecated).
- If your Git config, credential manager, or GitHub settings are not set up for HTTPS authentication, you may get errors (like authentication failed, or asking for a password repeatedly)

Option 2: Using SSH

- `git clone` `git@github.com` `:Sand-EnterpriseAI/udt-clean-water-toolkit.git`
- It requires you to have an SSH key pair (private and public key) set up and the public key added to your GitHub account.
- If your SSH keys are properly configured, cloning is seamless and does not require a password.
- Generating SSH Key:
    - In terminal, type: `ssh-keygen -t ed25519 -C "your_github_name"`
    - (example, `ssh-keygen -t ed25519 -C "james-sand"`)



- Saving SSH key in github

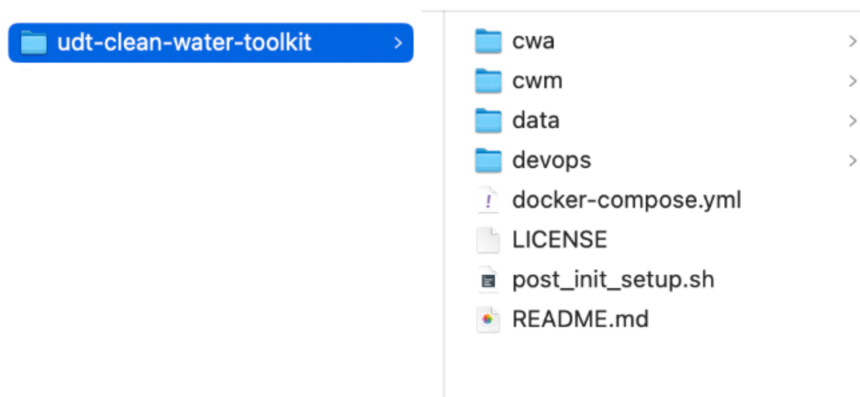


- The SSH login should now work
- Note: `ssh-keygen -R github.com` will remove an old key, should you experience the following log-in issue. Then repeat the SSH generation.

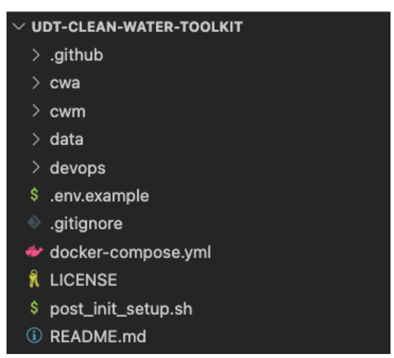
```
jamescombrink@za-jcombrink-mac-2 UDT code % git clone git@github.com:Sand-EnterpriseAI/udt-clean-water-toolkit.git
Cloning into 'udt-clean-water-toolkit'...
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
SHA256:uNiVztksCsDhcc0u9e8BujQXVUpKZIDTMczCvj3tD2s.
Please contact your system administrator.
Add correct host key in /Users/jamescombrink/.ssh/known_hosts to get rid of this message.
Offending RSA key in /Users/jamescombrink/.ssh/known_hosts:1
Host key for github.com has changed and you have requested strict checking.
Host key verification failed.
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
jamescombrink@za-jcombrink-mac-2 UDT code %
```

▼ Outcome of cloning:

The view within your home space will appear natively, as follows.



There are several hidden files in this view, which can be seen better with a code editor such as Visual Studio Code. Take specific note of `.env.example` which needs to be edited in the next section.



▼ Copy and configure environment configurations

2. **Configure Environment Variables** The toolkit uses an `.env` file to manage sensitive configuration like passwords and secret keys. A template is provided in the `.env.example` file.

   - Copy the example file:

     ```
     cp .env.example .env
     ```

- We change directory into the folder, with

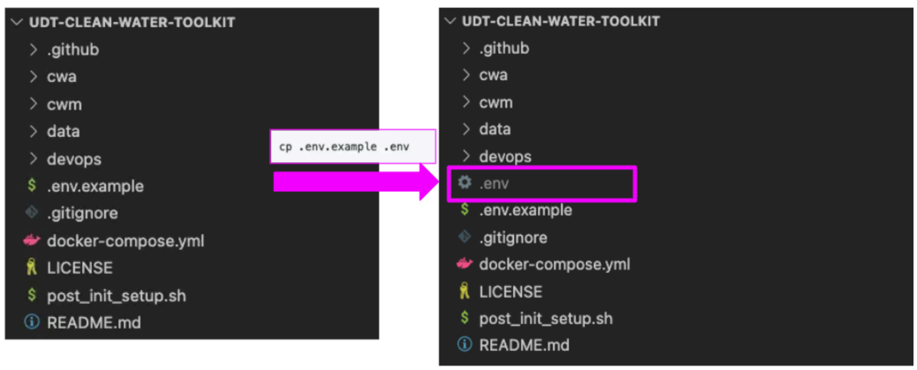
  `cd udt-clean-water-toolkit`

🛠️ If you are uncertain of your current environment, you can use `pwd` to find the present working directory. We want to navigate such that we are in `udt-clean-water-toolkit`

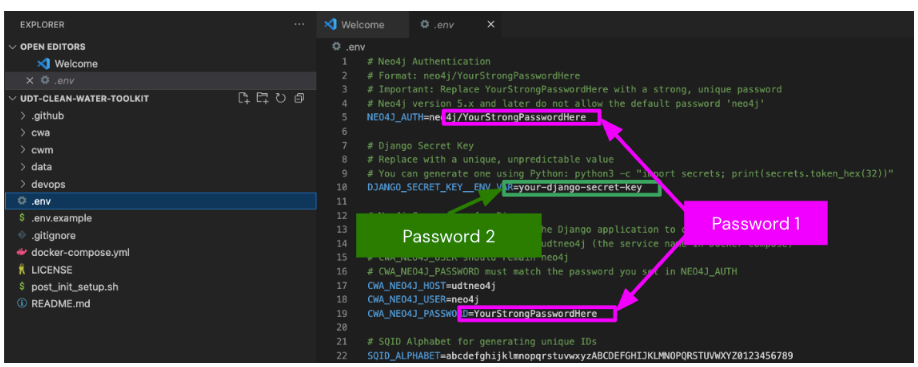
```
jamescombrink@za-jcombrink-mac-2 ~ % pwd
/Users/jamescombrink
jamescombrink@za-jcombrink-mac-2 ~ % cd udt-clean-water-toolkit
jamescombrink@za-jcombrink-mac-2 udt-clean-water-toolkit % pwd
/Users/jamescombrink/udt-clean-water-toolkit
```

- Now, we copy the environment file from the template:

  `cp .env.example .env`



- Edit file as needed (next)



▼ Running the Toolkit

🛠️ If services already running, need to compose down to terminate what is already running.
`docker compose down --volumes --rmi all --remove-orphans`

We compose the docker file. This command launches the full stack required for the UDT Clean Water Toolkit project, setting up databases, backend services, and making the platform ready for data/model operations... and it may take several minutes. Captured below are a few snippets of the progress as it unfolds.

`docker-compose up -d`

```
jamescombrink@za-jcombrink-mac-2 udt-clean-water-toolkit % docker-compose up -d
[+] Running 4/4
 ✔ Container udtneo4j            Running                                   0.0s
 ✔ Container udtpostgis          Healthy                                   0.5s
 ✔ Container udtcwageodjango     Running                                   0.0s
 ✔ Container udtorchestrator     Started                                   1.0s
```

```
[+] Building 15.7s (3/11)
 => [internal] load build definition from Dockerfile_cwa_geodjango                                         0.0s
 => => transferring dockerfile: 744B                                                                       0.0s
 => [internal] load .dockerignore                                                                          0.0s
 => => transferring context: 2B                                                                            0.0s
 => [internal] load metadata for docker.io/library/ubuntu:22.04                                            3.1s
 => [1/8] FROM docker.io/library/ubuntu:22.04@sha256:3c61d3759c2639d4b836d32a2d3c83fa0214e36f195a3421018db  12.5s
 => => resolve docker.io/library/ubuntu:22.04@sha256:3c61d3759c2639d4b836d32a2d3c83fa0214e36f195a3421018dba  0.0s
 => => sha256:0781dc6bbe44a53f76f5c3e20f0cd0975adfcd88e8b4987e7e8446ef1c948c66 2.31kB / 2.31kB              0.0s
 => => sha256:e730d307d74e767a94e2a36b22cfa82c38738f86f671c4fc0e3c90dadb75afbf 25.17MB / 27.36MB           12.5s
 => => sha256:3c61d3759c2639d4b836d32a2d3c83fa0214e36f195a3421018dbaaf79cbe37f 6.69kB / 6.69kB              0.0s
 => => sha256:ab34f0c9a1fbaa4c9211a7ae250cdda75d2f3351e015ffdf7bf4e50d9dc72744 424B / 424B                  0.0s
```

This command below will be used to watch the progress and output of the `orchestrator` service, so you know when setup tasks (like migrations) are complete or if brought any errors.

- `docker-compose logs -f orchestrator`

```
jamescombrink@za-jcombrink-mac-2 udt-clean-water-toolkit % docker-compose logs -f orchestrator
udtorchestrator  | the container ID == 133ae7633d55
udtorchestrator  | Collecting django==5.0 (from -r requirements.txt (line 1))
udtorchestrator  |   Downloading Django-5.0-py3-none-any.whl.metadata (4.1 kB)
udtorchestrator  | Collecting psycopg2-binary==2.9.9 (from -r requirements.txt (line 2))
udtorchestrator  |   Downloading psycopg2_binary-2.9.9-cp310-cp310-manylinux_2_17_aarch64.manylinux2014_aarch64.whl.metadata (4.4 kB)
udtorchestrator  | Collecting matplotlib==3.8.2 (from -r requirements.txt (line 3))
udtorchestrator  |   Downloading matplotlib-3.8.2-cp310-cp310-manylinux_2_17_aarch64.manylinux2014_aarch64.whl.metadata (5.8 kB)
udtorchestrator  | Collecting momepy==0.7.0 (from -r requirements.txt (line 4))
udtorchestrator  |   Downloading momepy-0.7.0-py3-none-any.whl.metadata (1.3 kB)
udtorchestrator  | Collecting neo4j==5.15.0 (from -r requirements.txt (line 5))
udtorchestrator  |   Downloading neo4j-5.15.0.tar.gz (196 kB)
udtorchestrator  |   Installing build dependencies: started
udtorchestrator  |   Installing build dependencies: finished with status 'done'
udtorchestrator  |   Getting requirements to build wheel: started
udtorchestrator  |   Getting requirements to build wheel: finished with status 'done'
udtorchestrator  |   Preparing metadata (pyproject.toml): started
udtorchestrator  |   Preparing metadata (pyproject.toml): finished with status 'done'
udtorchestrator  | Collecting neomodel==5.2.1 (from -r requirements.txt (line 6))
udtorchestrator  |   Downloading neomodel-5.2.1-py3-none-any.whl.metadata (6.0 kB)
udtorchestrator  | Collecting networkx==3.2.1 (from -r requirements.txt (line 7))
udtorchestrator  |   Downloading networkx-3.2.1-py3-none-any.whl.metadata (5.2 kB)
udtorchestrator  | Collecting sqids==0.4.1 (from -r requirements.txt (line 8))
udtorchestrator  |   Downloading sqids-0.4.1-py3-none-any.whl.metadata (4.6 kB)
udtorchestrator  | Collecting shapely==2.0.3 (from -r requirements.txt (line 9))
udtorchestrator  |   Downloading shapely-2.0.3-cp310-cp310-manylinux_2_17_aarch64.manylinux2014_aarch64.whl.metadata (7.0 kB)
udtorchestrator  | Collecting wntr==1.1.0 (from -r requirements.txt (line 10))
udtorchestrator  |   Downloading wntr-1.1.0.tar.gz (3.2 MB)
udtorchestrator  |      ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.2/3.2 MB 2.3 MB/s eta 0:00:00
udtorchestrator  |   Installing build dependencies: started
udtorchestrator  |   Installing build dependencies: finished with status 'done'
udtorchestrator  |   Getting requirements to build wheel: started
udtorchestrator  |   Getting requirements to build wheel: finished with status 'done'
udtorchestrator  |   Preparing metadata (pyproject.toml): started
```

Generating a synthetic network

The toolkit is build to enable data from multiple sources to be brought in. We will use the inbuild synthetic data generator capabilities within this guide. The command below will generate a sample network, including pipes, hydrants, valves, and synthetic flow data.

Run the following command from your host machine's terminal:

```
docker-compose exec cwageodjango python3 manage.py generate_synthetic_network
```

- `docker-compose exec cwageodjango python3 manage.py generate_synthetic_network`

```
jamescombrink@za-jcombrink-mac-2 udt-clean-water-toolkit % docker-compose exec cwageodjango python3 manage.py generate_synthetic_network
Starting synthetic network generation...
Cleaning up old data...
Creating Utility and DMA...
Generating pipe mains...
Generating hydrants and valves...
Generating synthetic flow data...
Successfully generated synthetic network.
```

These records are inserted into the corresponding tables of the PostGIS database (not as files on disk). You will not see new files in your project directory; instead, the data is accessible via the database, which can be queried through the Django app or other tools connected to the PostGIS service.

The network is now able to be be fed the synthetic data:

- `docker-compose exec cwageodjango python3` `manage.py` `load_network_to_neo4j`

```
jamescombrink@za-jcombrink-mac-2 udt-clean-water-toolkit % docker-compose exec cwageodjango python3 manage.py load_network_to_neo4j
Clearing Neo4j database...
Starting PostGIS to Neo4j transformation...
Instantiating GisToNeo4j...
Fetching pipe data from PostGIS...
Calculating graph components...
Creating Neo4j graph...
Successfully loaded network into Neo4j.
```

This command reads the water network data (such as pipes, hydrants, valves, etc.) from the PostGIS database. It then transforms and loads that spatial data into the Neo4j graph database, which is also running as part of your Docker Compose stack. Now the data can also be visualised from a Front-End compatible to Graph databases. The most accesible example in this case, being through Neo4j directly.

▼ Outcome of spinning up Neo4j within docker

We can run the command `dockerps`

To identify whether the Neo4j service is running. We should see 3 serivces running at present, GeoDjango; PostGIS, and Neo4j.

```
jamescombrink@za-jcombrink-mac-2 udt-clean-water-toolkit % docker ps
CONTAINER ID   IMAGE                             COMMAND                  CREATED       STATUS                       PORTS                                                      NAMES
8aebdb02ddd6   cwa_geodjango                     "tail -f /dev/null"      13 days ago   Up About an hour                                                                        udtcwageodjango
5bf8460c77a0   postgis/postgis:16-3.4            "docker-entrypoint.s…"   13 days ago   Up About an hour (healthy)   0.0.0.0:5432->5432/tcp                                     udtpostgis
5372ba9ca254   neo4j:5.20.0-community-bullseye   "tini -g -- /startup…"   13 days ago   Up About an hour             0.0.0.0:7474->7474/tcp, 7473/tcp, 0.0.0.0:7687->7687/tcp   udtneo4j
```

▼ Visualising the Graph database

We will guide you in logging into Neo4j, and running a few basic queries on the graph database.

1. **Neo4j Browser**
   - URL: http://localhost:7474/browser/ (click directly and follow prompts)
   - Connection:
     - **Connect URL**: `bolt://localhost:7687` (usually pre-filled)
     - **Authentication type**: Username / Password
     - **Username**: The username part of your `NEO4J_AUTH` value in the `.env` file (e.g., `neo4j`).
     - **Password**: The password part of your `NEO4J_AUTH` value in the `.env` file (e.g., `YourStrongPasswordHere`).

When we query a graph database, we use a **cypher.**

Cypher is a declarative graph query language, designed specifically for querying and updating graph databases (primarily Neo4j)... Think of it like SQL for graphs: it describes what to retrieve or modify, not how to do it.

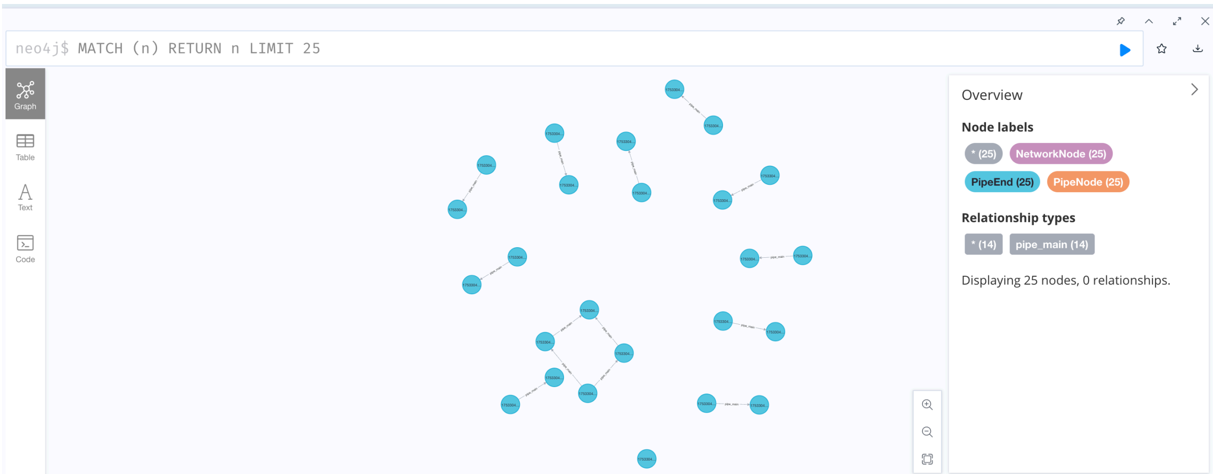
To run a cypher, we need to understand a few concepts:

- **Nodes** represent entities (like pipes, valves, DMAs).
- **Relationships** represent connections between nodes (like `CONNECTED_TO`, `HAS_ASSET`, or `IN_DMA`).
- Both nodes and relationships can have **properties** (e.g. `diameter: 300`, `type: "hydrant"`).

Cypher example 1:

```
MATCH (n) RETURN n LIMIT 25
```

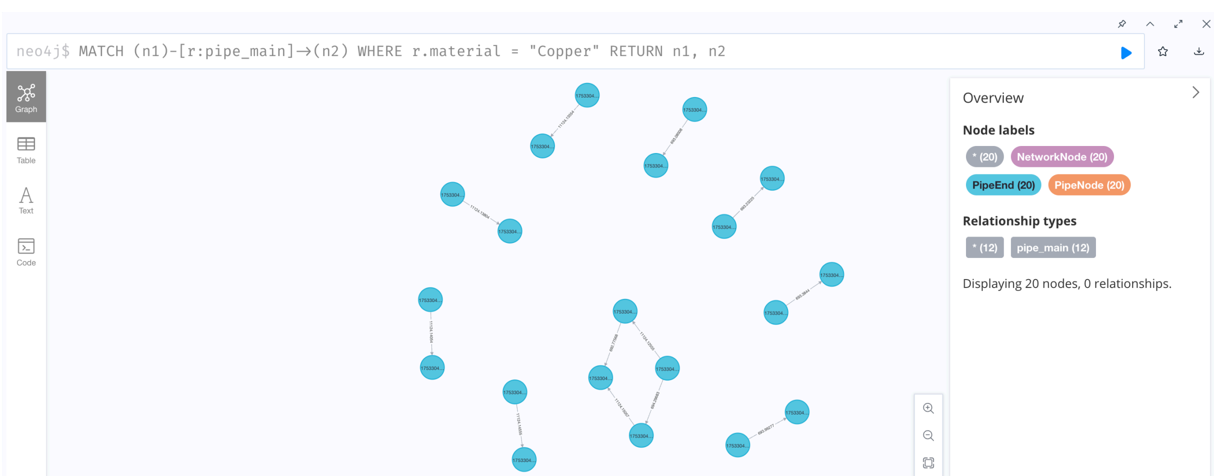
- Select all nodes in the Neo4j graph database (`MATCH (n)` matches every node).
- Return up to 25 of those nodes (`RETURN n LIMIT 25`)



Cypher example 2:

```
MATCH (n1)-[r:pipe_main]→(n2)
WHERE r.material = "Copper"
RETURN n1, n2
```

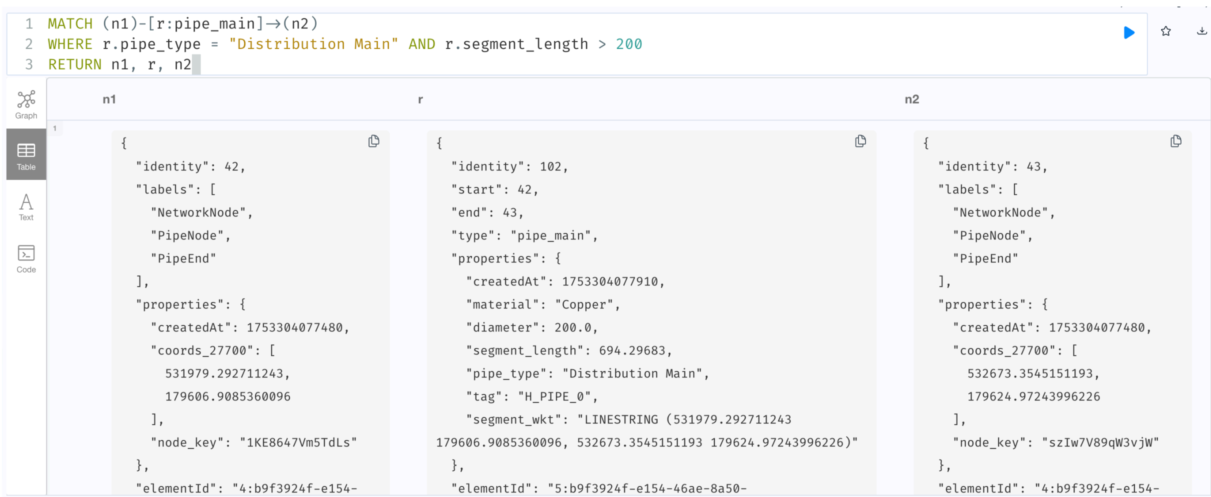
- Finds pairs of nodes (`n1` and `n2`) that are directly connected by a relationship of type `pipe_main`.
- Filters only those relationships where the property `material` is equal to `"Copper"`.
- Returns the connected nodes (`n1` and `n2`).



Cypher example 3:

```
MATCH (n1)-[r:pipe_main]→(n2)
WHERE r.pipe_type = "Distribution Main" AND r.segment_length > 200
RETURN n1, r, n2
```

- Find all pairs of nodes (`n1` and `n2`) that are directly connected by a relationship (`r`) of type `pipe_main`.
- Filter those relationships to only include cases where:
  - The property `pipe_type` is exactly `"Distribution Main"`.
  - The property `segment_length` is greater than 200.
- Return the two connected nodes (`n1`, `n2`) and the relationship (`r`) that connects them.



▼ Advanced Cyphers

In the **Unlocking Digital Twins** project, Cypher has been valuable in **navigating and analysing water distribution networks** once they had been transformed into graph-based models. It powered queries for asset coverage (e.g. acoustic logger deployment), structural resilience, and spatial relationships.

Cyphers are powerful for complex queries. They are great for relationship-heavy tasks like pathfinding, clustering, or pattern recognition. Listed below, are several examples (**not contained in the sample database**) where cyphers can rapidly enable high value.

Example below:

List all valves in the network and the pipes they are on [example query, not usable with synthetic data]

```
MATCH (valve:NetworkOptValve)-[:ON_PIPE]→(pipe:pipe_main)
RETURN valve, pipe
```

Find all pipes connected to a specific utility [example query, not usable with synthetic data]

```
MATCH (utility:Utility {name: "synthetic_utility"})-
[:MANAGES]→(dma:DMA)-[:CONTAINS]→(pipe:pipe_main)
RETURN pipe
```

Count the number of hydrants per DMA [example query, not usable with synthetic data]

```
MATCH (dma:DMA)-[:CONTAINS]→(pipe:pipe_main)-[:HAS_ASSET]→(hydrant:Hydrant)
RETURN dma.code, count(hydrant) AS hydrant_count
```