CityPulse ETL

An MVP pipeline for consolidating Smart City / IoT data

Author: Sam Patterson
Date: 19.12.2012
Repository: https://gitlab.com/s-a-m/citypulse-etl

Executive Summary

A minimum viable product (MVP) has been developed to run an extract, transform, load (ETL) pipeline on a variety of Smart City datasets.

The publicly accessible data has been successfully ingested into a database in a format appropriate for a data analyst to conduct visualisation and analysis.

Since only one iteration of development has been conducted, several potential improvements to the data model and pipeline are suggested. Nonetheless, the fully functioning end-to-end tool demonstrates a range of quality data engineering practices that place it as a solid foundation for further iteration.

Introduction

A mini-project has been undertaken over a weekend to build an extract, transform, load (ETL) pipeline for Smart City data collected from the CityPulse project.

The purpose of the document is to describe the project's activities and outcomes. After outlining the goals and constraints of the project, the following sections describe the steps taken to complete the project. Finally, some discussion about potential uses cases and future work is given.

Project goals and constraints

The primary goal of this project is to get the raw CityPulse datasets into a single datastore that can be queried for visualisation and analysis.

To this end, the desired outcome of the project is a functioning end-to-end pipeline which takes the raw data, applies basic transformations, and populates a target database. This will be developed with minimum viable product (MVP) mindset that will focus on delivering a tool that can be easily adapted and improved as its use cases evolve.

The project is naturally constrained in both time and resources. It is being performed by a single developer over a weekend as a recruitment challenge. Furthermore, the usual collaboration with the data analyst that the pipeline is being built for will not occur. For these reasons, only one iteration of design and development is being conducted. This means that while the primary goal will be achieved, there will be plenty of potential follow up work to further iterate, refine and improve the solution.

In terms of technological constraints, since this project is starting from scratch and is being developed in isolation from any other systems, only open-source libraries and local computing resources are going to be used. However, technologies will be chosen that will enable the tool to be used with other databases and deployment environments.

Explore and assess the data

The first step of the project is to explore the CityPulse datasets listed on their website to understand the variety and volume of data to be processed and ingested.

Datasets

The following datasets are listed on the CityPulse website.

Data type	City	Raw format	# Datasets listed
Road Traffic Data	Aarhus	Compressed CSV files	4
Pollution Data	Aarhus, Brasov	Compressed CSV files	2
Weather Data	Aarhus, Brasov	Compressed JSON files	4
Parking Data	Aarhus	CSV file	2
Social (Webcasted) Event Data	Surrey	CSV file	1
Cultural Event Data	Aarhus	CSV file	1
Library Event Data	Aarhus	CSV file	1

In addition to these, the following meta data is also provided.

Metadata	Relates to
Road Traffic Sensors	Road Traffic and Pollution Datasets
Parking Lots	Parking Datasets

To begin with, the 'raw' format of the data was manually downloaded. However, several of the files were found to be duplicated and miscategorised in CityPulse's indexing of the files.

Dataset index issues

Upon initial review of the linked datasets, the following issues were identified:

- The name of the linked file for the 'Aarhus Road Traffic Dataset-1' indicates that it is from Surrey, however the metadata for these records indicates it is in Aarhus so it is being used as if it is from Aarhus.
- The linked file for the 'Aarhus Road Traffic Dataset-4' points to either 'Aarhus Cultural Event Dataset-1' or 'Aarhus Road Traffic Dataset-3' in different pages on the site.
- The linked file for the 'Aarhus Parking Dataset-2' points to the same file as the first Aarhus parking dataset.
- The linked file for the 'Brasov Pollution Dataset-1', 'Brasov Weather Dataset-1' and 'Brasov Weather Dataset-2' files point to the same files from Aarhus.

An inspection into the backend file structure of the website was conducted, however the correct files did not appear to be there either. In these cases, the duplicated files are being ignored from here on and in the developed tool.

Extracting the data

To inspect the data more closely and begin building the initial 'extract' step of the ETL pipeline. Functions have been set up to automatically download and extract, where necessary, the datasets based on a JSON configuration file with the following format.

```json
[
    {
        "name": "Aarhus Parking Dataset-1",
        "data_type": "Parking Data",
        "url": "http://iot.ee.surrey.ac.uk:8080/datasets/parking/aarhus_parking.csv",
        "location": "Aarhus"
    },
    {
        "name": "Surrey Social Event Dataset-1",
        "data_type": "Social Event Data",
        "url": "http://iot.ee.surrey.ac.uk:8080/datasets/surreyevents/surrey_events.csv",
        "location": "Surrey"
    },
    ...
]
```

The following steps were taken to download each dataset, clean it, and read it into the Python development environment.

- Downloading raw files using the requests library and saving to a local data cache
- Unpacking from either `.tar.gz` or `.zip` archives where necessary
- Ignored non-data files (e.g. `__MACOSX` hidden directories)
- Reading the CSV or JSON files into pandas DataFrame objects
- Applied column names to CSVs where they were missing (based on other similar files or the developer's best guess)
- Column names were converted into the same (`snake_case`) styling for consistency across the datasets
- Setting correct data types (e.g. `datetime` s for timestamps, `integer` s for IDs and counts, `float` s for continuous variables)
- Removing duplicated rows (e.g. columns with identical values from the road traffic datasets)
- Removing duplicated columns (e.g. urls in columns 2 and 4 of 'Surrey Social Event' dataset are identical)

Define the data model

The initial design of the CityPulse ETL package comprises of a conceptual data model (which aligns with the tables of the relational database) along with the functional requirements of the command line tool that runs the ETL pipeline.

First pass conceptual data model

A first iteration of the conceptual data model was developed based on the key types of data.

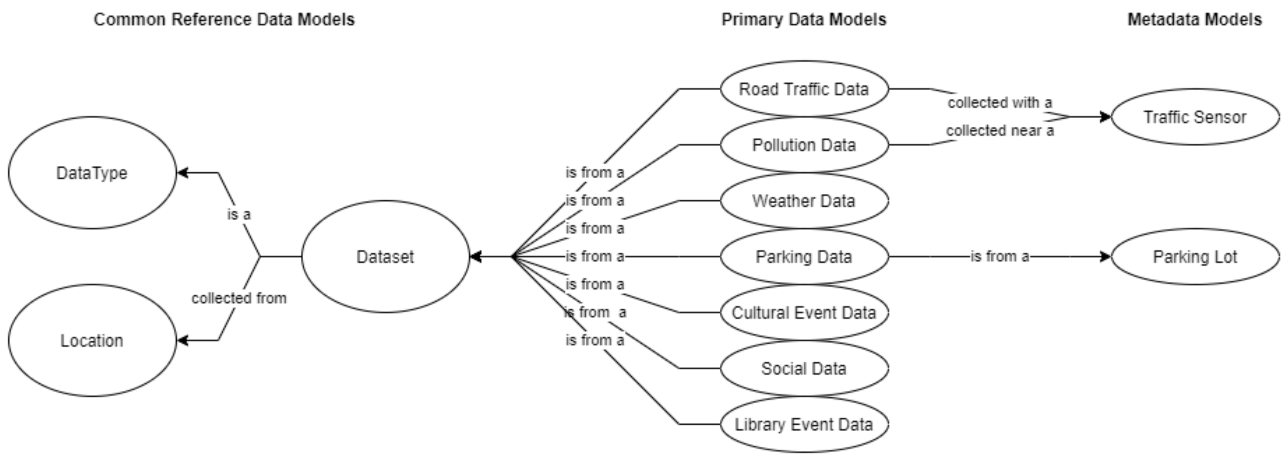


Figure 1. Conceptual Data Model

The primary data models are the seven different types of datasets hosted CityPulse. Alongside these are the two metadata models provided that relate to the Road Traffic, Pollution, and Parking Lot data.

There are also three reference data models which align with how CityPulse indexes the data.

While there is potential for further normalisation of the data model, this model achieves the project's primary goal within the time constraints. Furthermore, it would be recommend to further refine and clarify the user cases for the data before investing more effort to design a more sophisticated model, following an agile approach.

Functional design

A Command Line Interface (CLI) will be responsible for setting up the database and running the pipeline.

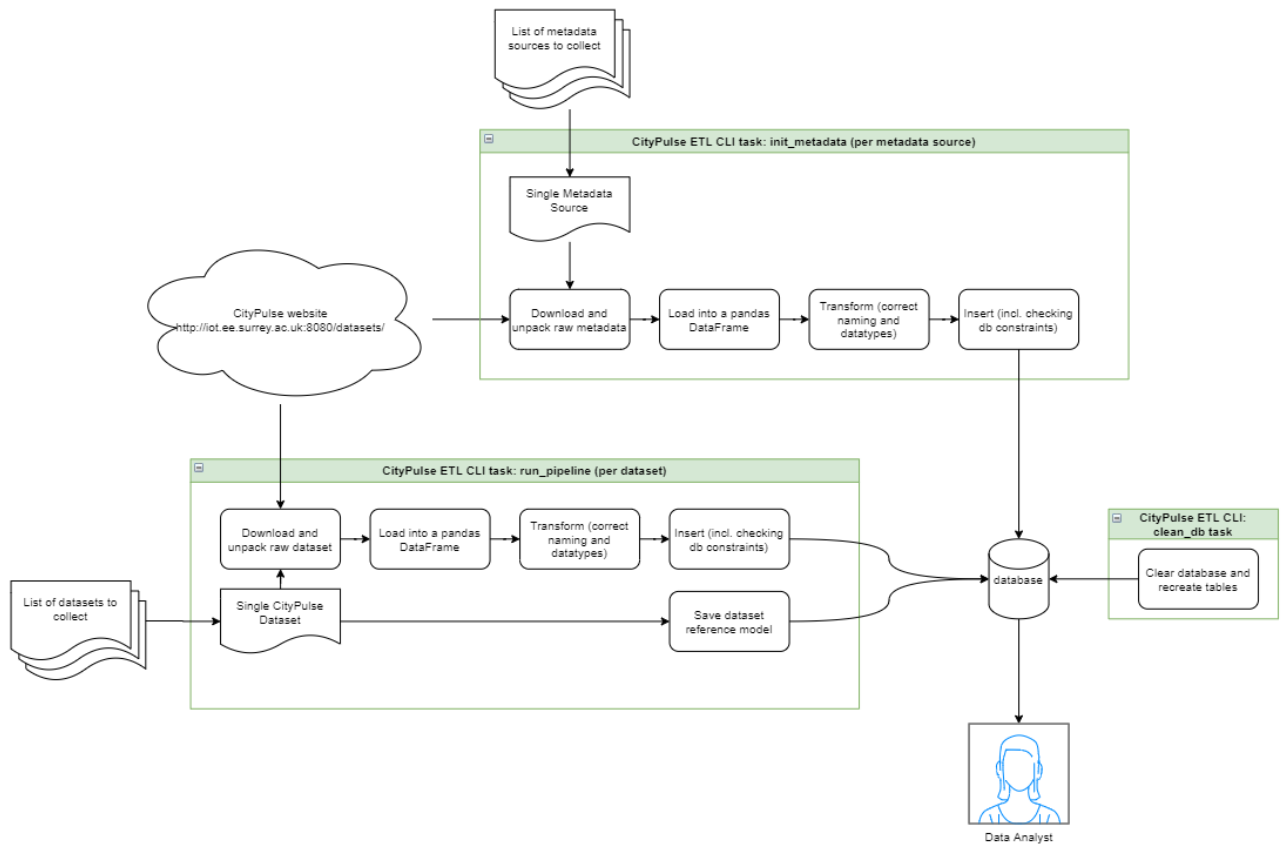

List of metadata sources to collect
CityPulse ETL CLI task: init_metadata (per metadata source)
Single Metadata Source
CityPulse website http://iot.ee.surrey.ac.uk:8080/datasets/
Download and unpack raw metadata
Load into a pandas DataFrame
Transform (correct naming and datatypes)
Insert (incl. checking db constraints)
CityPulse ETL CLI task: run_pipeline (per dataset)
Download and unpack raw dataset
Load into a pandas DataFrame
Transform (correct naming and datatypes)
Insert (incl. checking db constraints)
List of datasets to collect
Single CityPulse Dataset
Save dataset reference model
database
CityPulse ETL CLI: clean_db task
Clear database and recreate tables
Data Analyst

Figure 2. CLI Functional Design

The CLI will run three tasks:

- `run-pipeline` - This runs the ETL pipeline on a collection of datasets hosted by CityPulse. It takes a json file with a list of dictionaries that defines the datasets to be processed by the pipeline.
- `init-metadata` - This initialises the metadata tables. It should be run each time the database is recreated and potentially more often if new metadata becomes available.
- `clean-db` - This clears all the data from the database and creates the empty tables. For development it is useful when making large changes to the pipeline. For deployment to production, it only needs to be run once.

Run ETL to model the data

The above design has been developed as an MVP and is available in CityPulse ETL GitLab repository.

The resulting database schema is shown below.

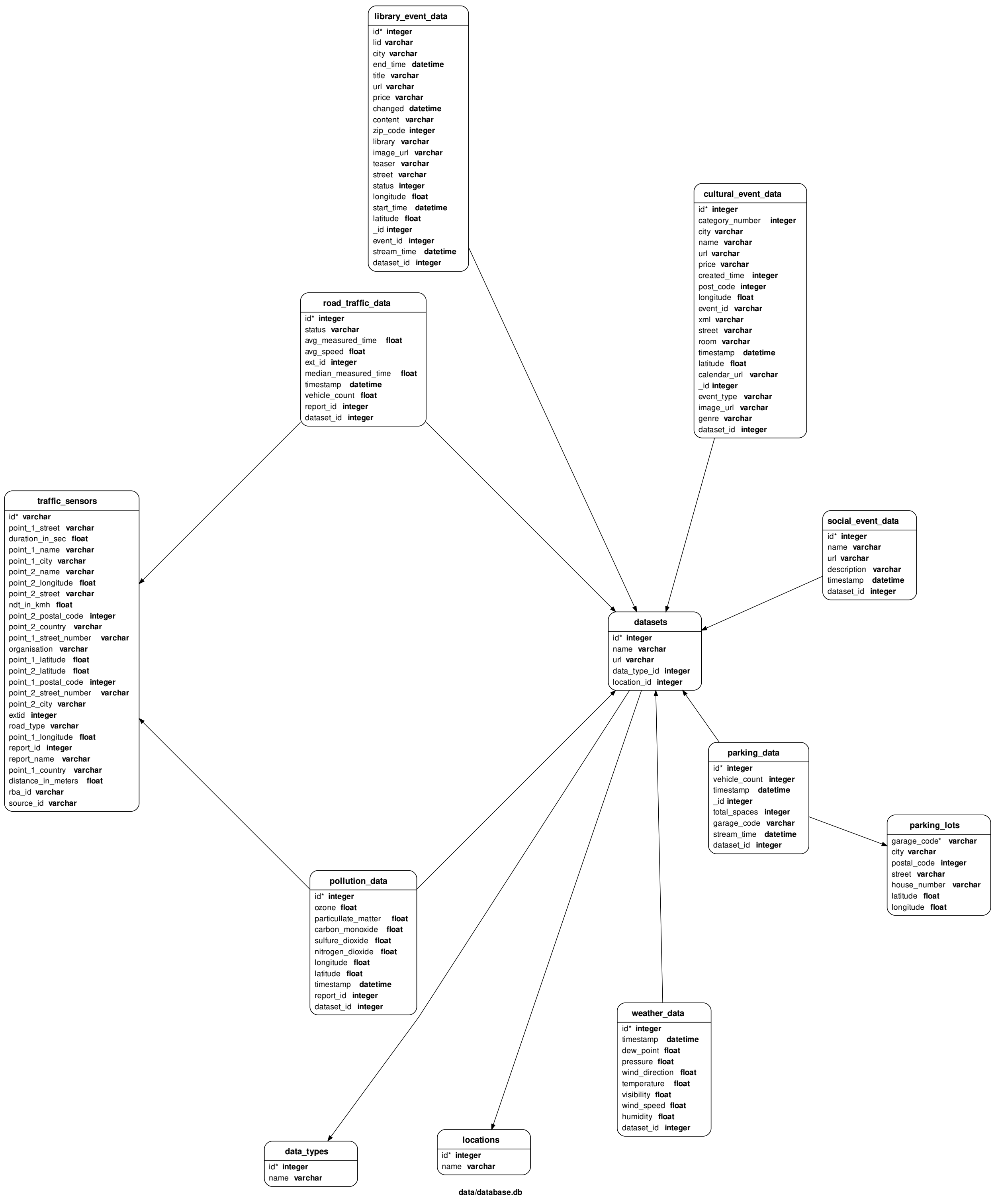


Figure 3. Implemented Database Schema

Technologies

The collection of technologies and libraries used have been selected based on familiarity to the developer, their maturity, and their popularity across the industry.

- Python 3 - often considered the default language to use in the data engineering / data science fields
- SQLAlchemy - provides an abstraction away from the database which will allow the tool to work on different databases if required
- pandas - gives powerful functionality for transforming large quantities of data
- requests - a widely-used and robust library for HTTP resources
- SQLite - a simple and self-contained SQL database

The tool has been built as an installable Python package that provides the CLI tool to the console when installed for ease of use and deployment.

As well as being useful for running the pipeline, installing the package also gives access to the SQLAlchemy ORM (object–relational mapping) classes which could be used by a consumer to interact with the data via Python objects instead of SQL.

A dependency graph of the developed package is available in the appendix.

Using the tool

The project's README file describes how to install the tool.

Once installed, the database can be initialised (incl. metadata) by running the following command:

```
$ citypulse-etl --metadata-json=metadata.json clean-db init-metadata
```

Which produces the following logs when successful:

```
[2021-12-19 17:16:07,057] INFO cli - Database cleared.
[2021-12-19 17:16:07,112] INFO cli - Database initialised.
[2021-12-19 17:16:07,112] INFO metadata - Downloading the metadata...
[2021-12-19 17:16:07,239] INFO metadata - Read in metadata file...
[2021-12-19 17:16:07,348] INFO metadata - Downloading the metadata...
[2021-12-19 17:16:07,446] INFO metadata - Read in metadata file...
[2021-12-19 17:16:07,456] INFO cli - Metadata initialised.
```

Once the database is initialised, the ETL pipeline can then be run with the following command:

```
$ citypulse-etl --dataset-json=all-datasets.json run-pipeline
```

Which produces the following logs showing the progress (truncated for brevity):

```
[2021-12-19 17:17:08,105] INFO cli - Running pipeline for dataset: Aarhus Weather Dataset-1
[2021-12-19 17:17:08,105] INFO pipeline - Creating the dataset record (and location if required)...
[2021-12-19 17:17:08,138] INFO pipeline - Downloading raw dataset files...
[2021-12-19 17:17:08,334] INFO pipeline - Unpacking / listing dataset files...
[2021-12-19 17:17:08,351] INFO pipeline - Reading data/raw/dewptm.txt...
[2021-12-19 17:17:08,476] INFO pipeline - Reading data/raw/pressurem.txt...
[2021-12-19 17:17:08,569] INFO pipeline - Reading data/raw/wdird.txt...
...
```

Since the models have been implemented with column data type, foreign key, uniqueness constraints, these constraints are checked upon insertion and are raised as exceptions by SQLAlchemy if they are violated. Transaction sessions are used to ensure that a only valid insertions from a completely successful pipeline is committed to the database.

Querying the data

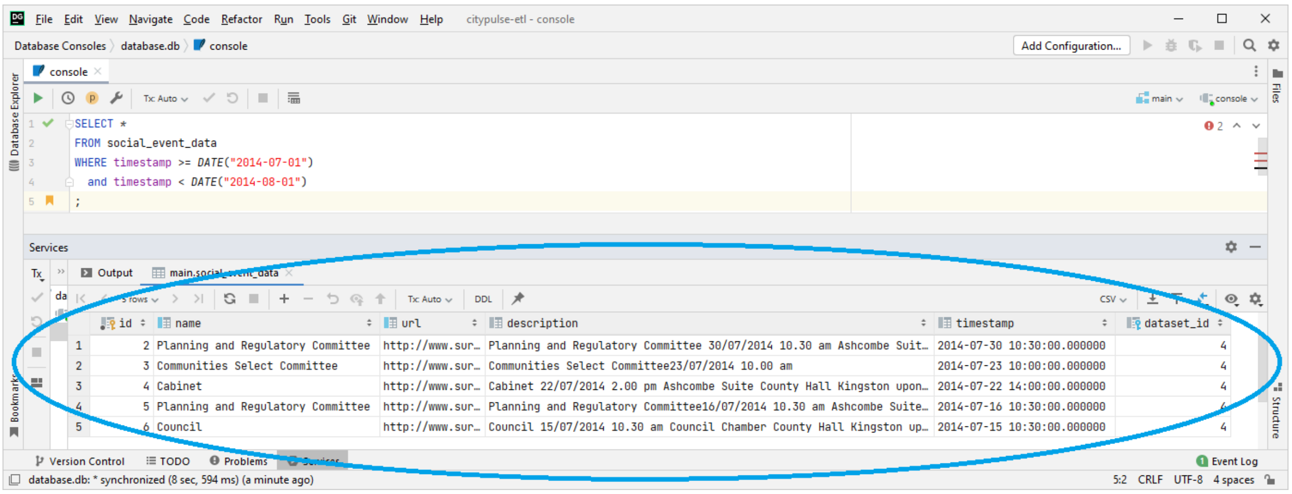
Once the ETL pipeline has run, the target database can be queried with the consumer's preferred tool.

For example, a basic query can be run get all of the social events in July 2014:

```sql
SELECT *
FROM social_event_data
WHERE timestamp >= DATE("2014-07-01")
```

```
  and timestamp < DATE("2014-08-01")
;
```

Returning these records:



```
SELECT *
FROM social_event_data
WHERE timestamp >= DATE("2014-07-01")
  and timestamp < DATE("2014-08-01")
;
```

	id	name	url	description	timestamp	dataset_id
1	2	Planning and Regulatory Committee	http://www.sur…	Planning and Regulatory Committee 30/07/2014 10.30 am Ashcombe Suit…	2014-07-30 10:30:00.000000	4
2	3	Communities Select Committee	http://www.sur…	Communities Select Committee23/07/2014 10.00 am	2014-07-23 10:00:00.000000	4
3	4	Cabinet	http://www.sur…	Cabinet 22/07/2014 2.00 pm Ashcombe Suite County Hall Kingston upon…	2014-07-22 14:00:00.000000	4
4	5	Planning and Regulatory Committee	http://www.sur…	Planning and Regulatory Committee16/07/2014 10.30 am Ashcombe Suite…	2014-07-16 10:30:00.000000	4
5	6	Council	http://www.sur…	Council 15/07/2014 10.30 am Council Chamber County Hall Kingston up…	2014-07-15 10:30:00.000000	4

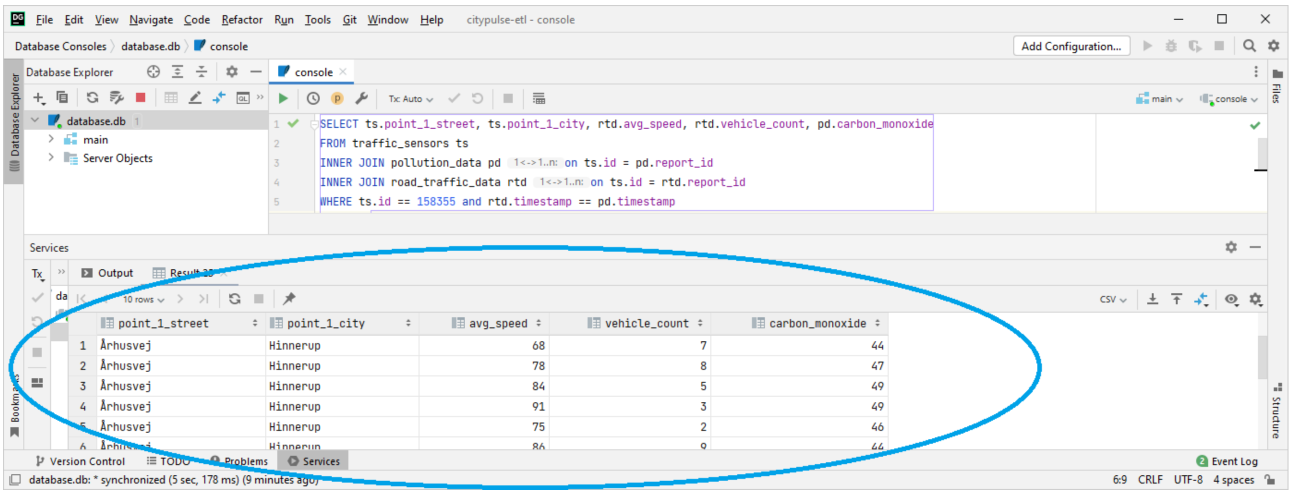
database.db: * synchronized (8 sec, 594 ms) (a minute ago)

Figure 4. Results from social events query

Or a more slightly more advanced query can join the traffic and pollution data along with the traffic sensor metadata:

```
SELECT ts.point_1_street, ts.point_1_city, rtd.avg_speed, rtd.vehicle_count, pd.carbon_monoxide
FROM traffic_sensors ts
INNER JOIN pollution_data pd on ts.id = pd.report_id
INNER JOIN road_traffic_data rtd on ts.id = rtd.report_id
WHERE ts.id == 158355 and rtd.timestamp == pd.timestamp
;
```

Returning these records:



```
SELECT ts.point_1_street, ts.point_1_city, rtd.avg_speed, rtd.vehicle_count, pd.carbon_monoxide
FROM traffic_sensors ts
INNER JOIN pollution_data pd  1<->1..n: on ts.id = pd.report_id
INNER JOIN road_traffic_data rtd  1<->1..n: on ts.id = rtd.report_id
WHERE ts.id == 158355 and rtd.timestamp == pd.timestamp
```

	point_1_street	point_1_city	avg_speed	vehicle_count	carbon_monoxide
1	Århusvej	Hinnerup	68	7	44
2	Århusvej	Hinnerup	78	8	47
3	Århusvej	Hinnerup	84	5	49
4	Århusvej	Hinnerup	91	3	49
5	Århusvej	Hinnerup	75	2	46
6	Århusvej	Hinnerup	86	9	44

database.db: * synchronized (5 sec, 178 ms) (9 minutes ago)

Figure 5. Result from traffic and pollution query

Discussion

This section discusses the developed tool from the perspective of using it or adapting it to meet some additional requirements.

Streaming data in real time

Real-time collection of the data could be achieved with this tool in a number of ways.

In it's current form, the tool could be used quite easily for real-time data collection if the new datasets were made available with similar HTTP requests as they are now. In this case, the ETL pipeline itself would work, and all that would need to added is for the tool to automatically find new datasets to process, instead of being given a fixed list of them in the json configuration file. This could be achieved by periodically crawling the CityPulse server for datasets that have not already been processed then running them through the ETL pipeline as normal.

Alternatively, if the data became available via a different communication protocol more suited to real-time streaming (e.g. MQTT or WebSockets), then the tool could be adapted to run as a micro-service while it listens for updates. This would be relatively straightforward as it would use the majority of the existing pipeline code, and just require some alteration to the logic which runs the command and does the initial download of the raw data.

100x more IoT

In order to scale up to a much larger amount of IoT sensors, the first recommended change to the tool would be to hook it up to a larger database than SQLite. Since it uses SQLAlchemy, this would just be a matter of changing the database driver & connection string that it is using.

Aside from using a more scalable database, the current tool itself should be quite scalable since a run is defined by a json configuration and the pipeline processes and commits each dataset from an independent database session/transaction. This means that pipelines could be orchestrated and run in a distributed computing environment. For example, setting up a shared queue of pipeline *jobs* and running them in parallel across multiple worker machines or in a serverless architecture (e.g. Azure Functions / AWS Lambda).

Converting geographic coordinates into human-readable addresses

Converting the longitude and latitude data into human-readable addresses would be particularly useful for presenting the data in the dashboard for end-users.

This could be easily achieved by either appending the existing pipeline or setting up an additional pipeline that uses a reverse geocoding web service, like Google's Reverse Geocoding API or the Open Streem Map Nominatim API, to convert the longitude and latitude values in to addresses then saves that to the database.

A demonstration script, `reverse-geocoding-demo.py` , has been implemented to show a minimal example of doing this for the two locations associated with each traffic sensor. It uses the Nominatim API and the result of the script is as followes (truncated for brevity):

```
Traffic Sensor #158895 measures traffic between:
    Søftenvej, Tilst 8381, Danmark
    and
    Søftenvej, Aarhus 8200, Danmark
Traffic Sensor #178600 measures traffic between:
    Grenåvej, Aarhus 8240, Danmark
    and
    Grenåvej, Aarhus 8240, Danmark
...
```

Harmonising the cultural and library event datasets

The cultural and library event datasets contain a lot of similar data. For example, the following fields exist across both datasets:

- Name of the event
- Address (City / Street / Postal Code)
- Longitude & Latitude
- URL & Image URL
- Price

With further transformation to align naming and datatypes, these could be combined into a shared data model (table).

For other fields available in one but not the other, sensible defaults could be chosen to fill them in. For example the theatre events have a 'room' field which isn't present in the library data, this could simply be 'library' or similar for the library events.

Doing this would allow more generalised exploration, visualisation and analysis of the data. It would also open up opportunity to integrate events from other locations into the same model, for example the social event data from Surrey which has a subset of these fields.

Other potential uses

Analysing the consolidation of the various types of datasets hosted by CityPulse in larger volume and in real-time opens up a wide range of potential smart city use cases. For example:

- Learning relationship between the datasets to predict the impact of them on one-another, e.g. how weather and events impact local traffic, parking availability and pollution

- Supporting public transport operators to make more informed and proactive decisions about the required capacity of the public transport system (based on current and predicted weather, events, and traffic)
- Dynamic pricing and allocation of parking spots based on expected use (e.g. for markets or other temporary events)
- Informing citizens about upcoming events with integrated weather updates and travel recommendations
- Providing citizens with real-time updates about traffic and pollution to inform their transit decisions
- Analysis to support scheduling events across city owned buildings in a way that minimises the impact on traffic and pollution

Recommended improvements / future work

In addition to the potential improvements discussed above, several improvements are recommended.

To increase the robustness & reliability of the tool, the following suggestions are made:

- Implement unit and integration testing
- Use database migrations to safely manage changes to the data model (e.g. using Alembic)
- Establish production environment for deployment (e.g. Azure SQL Database + Azure Functions)
- Set up continuous integration and deployment (CI/CD) to test, build and deploy the script to a production environment (e.g. using GitLab CI/CD or Azure DevOps)
- Document the system architecture more formally, using something like arc42

To further refine and improve the data model, the following work is recommended in collaboration with the data analyst:

- Find out where the wrongly indexed/linked CityPulse datasets are
- Investigate the data modelling framework associated with the broader CityPulse academic research project
- Conduct more research into the context of the data collection, it's purpose and the environment it has been collected in to help understand more about the value of the data and how it can be analysed
- Validate column naming where not provided in raw data (i.e. Cultural Event Data)
- Improve the data model to better suit more specific requirements of the visualisation and analysis in collaboration with the data analyst
- More detailed analysis into the quality of the data to build more sophisticated cleaning and transformation

Summary

The developed MVP has successfully met the primary goal of the project within the constraints.

The `citypulse-etl` Python package and CLI tool allows the user to initialise an SQL database and run the ETL pipeline on the data hosted on the CityPulse website to populate the target database.

Finally, this report has provided discussion into future potential uses which highlights the flexibility of the tool along with recommendations for future improvements.

Appendix

Package structure

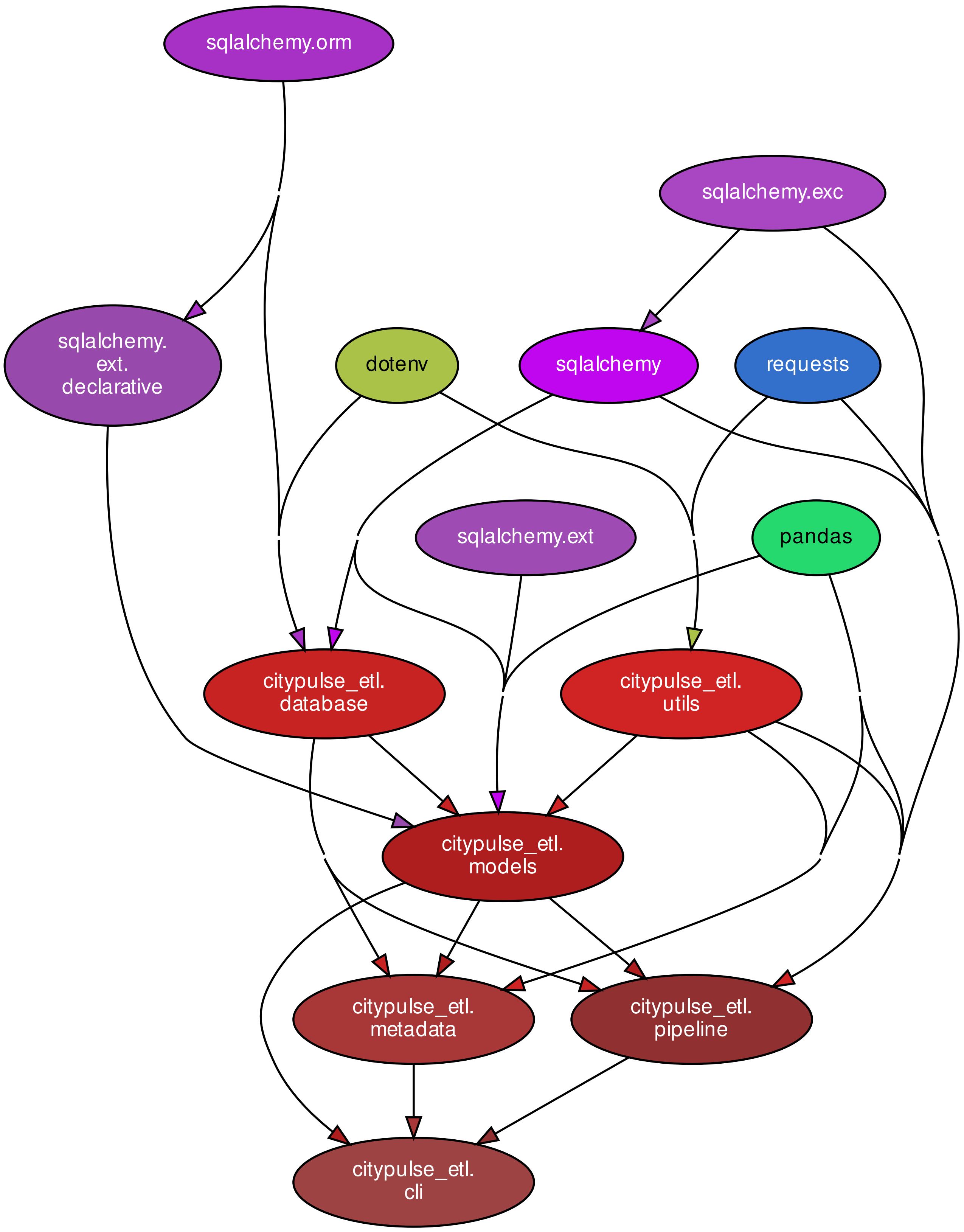


Figure 6. Dependency Graph of the `citypulse_etl` Python package