**Step 1: SETUP BASIC INFRASTRUCTURE**

In this assignment we were required to first set the infrastructure which includes installation and configuration of EMQX and TimescaleDB. These were successfully implemented and VS Code was configured as our prefered code editor.

Key steps:

# Stage 1: Install the required tools and packages

# Stage 2: Start EMQX (emqxstart). We have verified the EMQX status from: http://localhost:18083/status

# Stage 3: Create a timescale DB: CREATE DATABASE smart\_grid;

# Stage 4: Login to newly created DB: \c smart\_grid

# Stage 5: Create data generation script (that generate data formatted into JSON and subscribe to energy/meters/#) and store data in Postgres

**Step 2: DATA GENERATION IMPLEMENTATION**

A script that generate random data for 500 meters as the required minimum was created. The data generated were jsonized and sent to Postgres DB. Data insertion was verified.

smart\_grid=# SELECT COUNT(\*) FROM energy\_readings;

count

---------

2016000

(1 row)

Time: 58.984 ms

**Step 3: BASIC HYPERTABLE CREATION AND INITIAL DATA LOADING**

* We first converted our postgress to hypertable: CREATE EXTENSION IF NOT EXISTS timescaledb;

# Convert table: SELECT create\_hypertable('energy\_readings', 'timestamp', migrate\_data => true, chunk\_time\_interval => INTERVAL '1 day');

* We then generated and inserted the data (2 million observations) – 500 meters \* 12 times an hour (every 5 min) \* 24 hrs a day \* 14 days
* We turned the timer on to be able to record the tme taken to execute the command
* We have also set interractive pager off: \pset pager off

We executed 4 different queries and recorded the time taken for each:

-- Query 1: Average power consumption per hour today

SELECT time\_bucket('1 hour', timestamp) AS hour, AVG(power) as avg\_power

FROM energy\_readings

WHERE timestamp >= DATE\_TRUNC('day', NOW())

GROUP BY hour ORDER BY hour;

hour | avg\_power

------------------------+--------------------

2025-05-08 00:00:00+02 | 347.6398733333326

2025-05-08 01:00:00+02 | 100.38524999999986

2025-05-08 02:00:00+02 | 99.7987000000001

2025-05-08 03:00:00+02 | 100.00646666666695

2025-05-08 04:00:00+02 | 99.97759666666643

2025-05-08 05:00:00+02 | 100.25801499999982

2025-05-08 06:00:00+02 | 100.0476833333332

2025-05-08 07:00:00+02 | 100.24772166666641

2025-05-08 08:00:00+02 | 349.51751666666803

2025-05-08 09:00:00+02 | 351.2862650000008

2025-05-08 10:00:00+02 | 349.05111333333303

2025-05-08 11:00:00+02 | 349.6450799999996

2025-05-08 12:00:00+02 | 100.16174500000062

2025-05-08 13:00:00+02 | 100.19261666666691

2025-05-08 14:00:00+02 | 99.59812333333335

(15 rows)

Time: 56.364 ms

-- Query 2: Find peak consumption periods in the past week

SELECT time\_bucket('15 minutes'

, timestamp) AS period,

AVG(power) as avg\_power

FROM energy\_readings

WHERE timestamp >= NOW() - INTERVAL '7 days'

GROUP BY period ORDER BY avg\_power DESC LIMIT 10;

period | avg\_power

------------------------+--------------------

2025-05-07 20:15:00+02 | 355.63203999999996

2025-05-06 22:15:00+02 | 354.85644666666695

2025-05-07 08:15:00+02 | 354.72145333333344

2025-05-04 21:45:00+02 | 354.6076866666668

2025-05-06 23:15:00+02 | 354.41178000000025

2025-05-05 20:15:00+02 | 354.32904

2025-05-02 22:30:00+02 | 353.80574666666683

2025-05-05 11:45:00+02 | 353.619693333333

2025-05-03 11:00:00+02 | 353.39759999999995

2025-05-04 22:00:00+02 | 353.374653333333

(10 rows)

Time: 321.442 ms

-- Query 3: Monthly consumption per meter

SELECT meter\_id,

DATE\_TRUNC('month'

, timestamp) as month,

SUM(energy) as total\_energy

FROM energy\_readings

GROUP BY meter\_id, month

ORDER BY month, total\_energy DESC;

meter\_id | month | total\_energy

------------+------------------------+--------------------

1000000372 | 2025-04-01 00:00:00+02 | 30383.820000000018

1000000314 | 2025-04-01 00:00:00+02 | 30371.480000000036

1000000464 | 2025-04-01 00:00:00+02 | 30361.060000000052

1000000144 | 2025-04-01 00:00:00+02 | 30360.189999999948

1000000051 | 2025-04-01 00:00:00+02 | 30358.35999999999

……………………………………………………………………………………………………

1000000164 | 2025-05-01 00:00:00+02 | 34704.60999999997

1000000371 | 2025-05-01 00:00:00+02 | 34696.560000000005

1000000052 | 2025-05-01 00:00:00+02 | 34691.60000000003

1000000485 | 2025-05-01 00:00:00+02 | 34640.18999999999

1000000466 | 2025-05-01 00:00:00+02 | 34547.48999999997

(1000 rows)

Time: 580.186 ms

-- Query 4: Full dataset scan

SELECT COUNT(\*), AVG(power), MAX(power), MIN(power)

FROM energy\_readings;

count | avg | max | min

---------+--------------------+-----+-----

2016000 | 193.71918863591256 | 500 | 50

(1 row)

Time: 154.039 ms

**Step 4: CHUNK INTERVAL EXPERIMENTATION**

* We created 2 more tables with the same schema as the one we created before
* Data were loaded into these 2 tables.

INSERT INTO energy\_readings\_3h SELECT \* FROM energy\_readings;

INSERT INTO energy\_readings\_week SELECT \* FROM energy\_readings;

* This was followed by restarting Postgres before the next step to ensure no caching effect

Results

|  |  |  |  |
| --- | --- | --- | --- |
| Query | 3-hour chunks | 1-day chunks | 1-week chunks |
| 1 | 35.663 ms | 22.934 ms | 112.399 ms |
| 2 | 184.263 ms | 121.732 ms | 242.430 ms |
| 3 | 373.960 ms | 608.152 ms | 281.976 ms |
| 4 | 155.912 ms | 143.156 ms | 388.335 ms |

Chuck distribution analysis:

chunk\_schema | chunk\_name | range\_start | range\_end | chunk\_size

-----------------------+--------------------+------------------------+------------------------+------------

\_timescaledb\_internal | \_hyper\_1\_109\_chunk | 2025-04-24 02:00:00+02 | 2025-04-25 02:00:00+02 | 7424 kB

\_timescaledb\_internal | \_hyper\_1\_110\_chunk | 2025-04-25 02:00:00+02 | 2025-04-26 02:00:00+02 | 15 MB

\_timescaledb\_internal | \_hyper\_1\_111\_chunk | 2025-04-26 02:00:00+02 | 2025-04-27 02:00:00+02 | 15 MB

\_timescaledb\_internal | \_hyper\_1\_112\_chunk | 2025-04-27 02:00:00+02 | 2025-04-28 02:00:00+02 | 15 MB

\_timescaledb\_internal | \_hyper\_1\_113\_chunk | 2025-04-28 02:00:00+02 | 2025-04-29 02:00:00+02 | 15 MB

\_timescaledb\_internal | \_hyper\_1\_114\_chunk | 2025-04-29 02:00:00+02 | 2025-04-30 02:00:00+02 | 15 MB

\_timescaledb\_internal | \_hyper\_1\_115\_chunk | 2025-04-30 02:00:00+02 | 2025-05-01 02:00:00+02 | 15 MB

\_timescaledb\_internal | \_hyper\_1\_116\_chunk | 2025-05-01 02:00:00+02 | 2025-05-02 02:00:00+02 | 15 MB

\_timescaledb\_internal | \_hyper\_1\_117\_chunk | 2025-05-02 02:00:00+02 | 2025-05-03 02:00:00+02 | 15 MB

\_timescaledb\_internal | \_hyper\_1\_118\_chunk | 2025-05-03 02:00:00+02 | 2025-05-04 02:00:00+02 | 15 MB

\_timescaledb\_internal | \_hyper\_1\_119\_chunk | 2025-05-04 02:00:00+02 | 2025-05-05 02:00:00+02 | 15 MB

\_timescaledb\_internal | \_hyper\_1\_120\_chunk | 2025-05-05 02:00:00+02 | 2025-05-06 02:00:00+02 | 15 MB

\_timescaledb\_internal | \_hyper\_1\_121\_chunk | 2025-05-06 02:00:00+02 | 2025-05-07 02:00:00+02 | 15 MB

\_timescaledb\_internal | \_hyper\_1\_122\_chunk | 2025-05-07 02:00:00+02 | 2025-05-08 02:00:00+02 | 15 MB

\_timescaledb\_internal | \_hyper\_1\_123\_chunk | 2025-05-08 02:00:00+02 | 2025-05-09 02:00:00+02 | 8064 kB

(15 rows)

Time: 41.129 ms

Step 5: compression implementation

* Initial size:

hypertable\_name | size

----------------------+--------

energy\_readings | 211 MB

energy\_readings\_3h | 215 MB

energy\_readings\_week | 211 MB

(3 rows)

Time: 67.908 ms

Manual compression: we also performed manual compression:

SELECT compress\_chunk(format('%I.%I', chunk\_schema, chunk\_name)::regclass)

FROM timescaledb\_information.chunks

WHERE hypertable\_name IN (

'energy\_readings', 'energy\_readings\_3h', 'energy\_readings\_week')

AND is\_compressed = false;

After compression: the size significantly reduced as can be seen from the table below:

hypertable\_name | size

----------------------+-------

energy\_readings | 89 MB

energy\_readings\_3h | 96 MB

energy\_readings\_week | 88 MB

(3 rows)

|  |  |  |  |
| --- | --- | --- | --- |
| Query | 3-hour chunks | 1-day chunks | 1-week chunks |
| 2 | 56.462 ms | 55.461 ms | 74.197 ms |
| 3 | 186.553 ms | 166.829 ms | 173.673 ms |

Step 6: **CONTINUOUS AGGREGATION**

smart\_grid=# CREATE MATERIALIZED VIEW energy\_readings\_15min

WITH (timescaledb.continuous) AS

SELECT

meter\_id,

time\_bucket('15 minutes', timestamp) AS bucket,

AVG(power) AS avg\_power,

MAX(power) AS max\_power,

SUM(energy) AS total\_energy

FROM energy\_readings

GROUP BY meter\_id, bucket;

NOTICE: refreshing continuous aggregate "energy\_readings\_15min"

HINT: Use WITH NO DATA if you do not want to refresh the continuous aggregate on creation.

CREATE MATERIALIZED VIEW

Time: 4944.849 ms (00:04.945)

smart\_grid=#

Then we added automatic refresh policy

After this, we compared 2 queries, one taking 31.756 ms to execute, another 2.625 ms to execute (the second one).

Step 7: The dashboard was developed. Execute the command below to launch the dashboard: streamlit run "/Users/mac/Others/AUCA/Big Data Analytics/Assignment 1/dashboard.py"