Module 4 - Querying Timestamp Data (TimescaleDB + InfluxDB)

Prerequisites

Module 1 setup:

- Docker Compose stack from Module 1 running with containers `timescaledb` and `influxdb`.
- TimescaleDB hypertable `sensor_data(time TIMESTAMPTZ, sensor_id INT, temperature DOUBLE PRECISION)` or `cpu_usage(time TIMESTAMPTZ, host TEXT, usage_percent DOUBLE PRECISION)`.
- InfluxDB bucket `example-bucket`, org `example-org`.

1) Time-range queries (WHERE time BETWEEN … / range(start: ...))

Concept: select rows/points by timestamp. Use `WHERE time > NOW() - INTERVAL '...'` in SQL. In Flux use `range(start: -15m)` or `range(start: time(v: "..."), stop: time(v: "..."))` for absolute intervals.

TimescaleDB — last 15 minutes

```
docker exec -i timescaledb psql -U admin -d metricsdb -c "\
SELECT time, sensor_id, temperature \
FROM sensor_data \
WHERE time > NOW() - INTERVAL '15 minutes' \
ORDER BY time ASC LIMIT 500;"
```

TimescaleDB — BETWEEN two absolute timestamps

```
docker exec -i timescaledb psql -U admin -d metricsdb -c "\
SELECT time, sensor_id, temperature \
FROM sensor_data \
WHERE time BETWEEN TIMESTAMPTZ '2025-09-01 00:00:00+00' AND TIMESTAMPTZ
'2025-09-18 23:59:59+00' \
ORDER BY time ASC;"
```

InfluxDB (Flux) — last 15 minutes

```
docker exec -i influxdb influx query --org example-org --raw <<'FLUX'
from(bucket:"example-bucket")
  |> range(start: -15m)
  |> filter(fn: (r) => r._measurement == "temperature" and r._field == "value")
  |> sort(columns: ["_time"])
FLUX
```

InfluxDB (Flux) — between two absolute times

```
docker exec -i influxdb influx query --org example-org --raw <<'FLUX'
from(bucket:"example-bucket")
  |> range(start: time(v: "2025-09-01T00:00:00Z"), stop: time(v: "2025-09-18T23:59:59Z"))
  |> filter(fn: (r) => r._measurement == "temperature" and r._field == "value")
  |> sort(columns: ["_time"])
FLUX
```

2) Aggregations (avg(), min(), max())

Concept: aggregations compute summary statistics. Use `AVG()`, `MIN()`, `MAX()` in SQL; `mean()`, `min()`, `max()` or `aggregateWindow()` in Flux.

TimescaleDB — average temperature last day

```
docker exec -i timescaledb psql -U admin -d metricsdb -c "\
SELECT AVG(temperature) AS avg_temp \
FROM sensor_data \
WHERE time > NOW() - INTERVAL '1 day';"
```

TimescaleDB — min and max over last 7 days

```
docker exec -i timescaledb psql -U admin -d metricsdb -c "\
SELECT MIN(temperature) AS t_min, MAX(temperature) AS t_max \
FROM sensor_data \
WHERE time > NOW() - INTERVAL '7 days';"
```

InfluxDB (Flux) — mean over last day

```
docker exec -i influxdb influx query --org example-org --raw <<'FLUX'
```

```
from(bucket:"example-bucket")
  |> range(start: -1d)
  |> filter(fn: (r) => r._measurement == "temperature" and r._field == "value")
  |> mean()
FLUX
```

InfluxDB (Flux) — min and max over last 7 days

You can run `min()` and `max()` separately:

```
docker exec -i influxdb influx query --org example-org --raw <<'FLUX'
from(bucket:"example-bucket")
  |> range(start: -7d)
  |> filter(fn: (r) => r._measurement == "temperature" and r._field == "value")
  |> min()
FLUX

docker exec -i influxdb influx query --org example-org --raw <<'FLUX'
from(bucket:"example-bucket")
  |> range(start: -7d)
  |> filter(fn: (r) => r._measurement == "temperature" and r._field == "value")
  |> max()
FLUX
```

3) Time bucketing (time_bucket in TimescaleDB, aggregateWindow in InfluxDB)

Concept: group points into fixed-size buckets (hour, day). `time_bucket()` in TimescaleDB returns the bucket start.

TimescaleDB — hourly averages last 24 hours

```
docker exec -i timescaledb psql -U admin -d metricsdb -c "\
SELECT time_bucket('1 hour', time) AS bucket, \
       AVG(temperature) AS avg_temp, \
       MIN(temperature) AS min_temp, \
       MAX(temperature) AS max_temp \
FROM sensor_data \
WHERE time > NOW() - INTERVAL '24 hours' \
```

```
GROUP BY bucket \
ORDER BY bucket;"
```

TimescaleDB — daily counts last 30 days

```
docker exec -i timescaledb psql -U admin -d metricsdb -c "\
SELECT time_bucket('1 day', time) AS day, COUNT(*) AS samples \
FROM sensor_data \
WHERE time > NOW() - INTERVAL '30 days' \
GROUP BY day \
ORDER BY day;"
```

InfluxDB (Flux) — hourly averages last 24 hours

```
docker exec -i influxdb influx query --org example-org --raw <<'FLUX'
from(bucket:"example-bucket")
  |> range(start: -24h)
  |> filter(fn: (r) => r._measurement == "temperature" and r._field == "value")
  |> aggregateWindow(every: 1h, fn: mean, createEmpty: false)
  |> yield(name: "hourly_mean")
FLUX
```

InfluxDB (Flux) — daily counts last 30 days

```
docker exec -i influxdb influx query --org example-org --raw <<'FLUX'
from(bucket:"example-bucket")
  |> range(start: -30d)
  |> filter(fn: (r) => r._measurement == "temperature" and r._field == "value")
  |> aggregateWindow(every: 1d, fn: count, createEmpty: false)
FLUX
```

Hands-on — ready-to-run tasks (docker, using `sensor_data`)

> These commands assume you have the hypertable `sensor_data(time TIMESTAMPTZ, sensor_id INT, temperature DOUBLE PRECISION)` as used in Module 1. If your CPU-like metric is stored in `temperature`, these will run immediately.

4) Query CPU-like usage in last 15 minutes (use `sensor_data`)

```
docker exec -i timescaledb psql -U admin -d metricsdb -c "\
SELECT time, sensor_id, temperature AS cpu_like \
```

```
FROM sensor_data \
WHERE time > NOW() - INTERVAL '15 minutes' \
ORDER BY time ASC \
LIMIT 500;"
```

5) Compute daily averages (daily average temperature for last 30 days)

```
docker exec -i timescaledb psql -U admin -d metricsdb -c "\
SELECT time_bucket('1 day', time) AS day, \
      AVG(temperature) AS avg_temp, \
      COUNT(*) AS samples \
FROM sensor_data \
WHERE time > NOW() - INTERVAL '30 days' \
GROUP BY day \
ORDER BY day;"
```

6) Build a sliding window query — last 100 points per sensor

ROW_NUMBER() approach (recommended single-pass):

```
docker exec -i timescaledb psql -U admin -d metricsdb -c "\
SELECT sensor_id, time, temperature \
FROM ( \
  SELECT sensor_id, time, temperature, \
        ROW_NUMBER() OVER (PARTITION BY sensor_id ORDER BY time DESC) AS rn \
  FROM sensor_data \
) x \
WHERE rn <= 100 \
ORDER BY sensor_id, time DESC;"
```

CROSS JOIN LATERAL approach (alternative):

```
docker exec -i timescaledb psql -U admin -d metricsdb -c "\
SELECT s.sensor_id, t.time, t.temperature \
FROM (SELECT DISTINCT sensor_id FROM sensor_data) s \
CROSS JOIN LATERAL ( \
  SELECT sensor_id, time, temperature \
  FROM sensor_data \
  WHERE sensor_id = s.sensor_id \
  ORDER BY time DESC LIMIT 100 \
) t \
ORDER BY s.sensor_id, t.time DESC;"
```

InfluxDB equivalents (Flux examples)

Last 15 minutes (Flux):

```
docker exec -i influxdb influx query --org example-org --raw <<'FLUX'
from(bucket:"example-bucket")
  |> range(start: -15m)
  |> filter(fn: (r) => r._measurement == "temperature" and r._field == "value")
  |> sort(columns: ["_time"])
FLUX
```

Daily averages (aggregateWindow):

```
docker exec -i influxdb influx query --org example-org --raw <<'FLUX'
from(bucket:"example-bucket")
  |> range(start: -30d)
  |> filter(fn: (r) => r._measurement == "temperature" and r._field == "value")
  |> aggregateWindow(every: 1d, fn: mean, createEmpty: false)
FLUX
```

Last 100 points per sensor (group + limit):

```
docker exec -i influxdb influx query --org example-org --raw <<'FLUX'
from(bucket:"example-bucket")
  |> range(start: -30d)
  |> filter(fn: (r) => r._measurement == "temperature" and r._field == "value")
  |> group(columns: ["sensor_id"])
  |> sort(columns: ["_time"], desc: true)
  |> limit(n: 100)
  |> sort(columns: ["sensor_id","_time"])
FLUX
```

Insert synthetic test data (TimescaleDB `sensor_data`)

Run this to populate `sensor_data` with many recent points for sensors 1..5 (you can adjust counts):

```
docker exec -i timescaledb psql -U admin -d metricsdb -c "\
-- create table if not present (safe if Module1 already created it)
CREATE TABLE IF NOT EXISTS sensor_data (
  time TIMESTAMPTZ NOT NULL,
  sensor_id INT NOT NULL,
  temperature DOUBLE PRECISION
);
SELECT create_hypertable('sensor_data','time', if_not_exists => TRUE);"

# Insert synthetic recent points (5 sensors, 500 points each spaced 1 minute apart)
docker exec -i timescaledb psql -U admin -d metricsdb -c "\
INSERT INTO sensor_data (time, sensor_id, temperature)
SELECT NOW() - ( (s.idx*500 + g.i) || ' minutes')::INTERVAL AS t,
       s.sensor_id,
       15 + random()*15 AS temperature
FROM (SELECT unnest(ARRAY[1,2,3,4,5]) AS sensor_id, generate_series(0,4) AS idx) s
CROSS JOIN generate_series(0,499) g(i)
ON CONFLICT DO NOTHING;"
```

Flux query for Last 100 points per sensor

It sorts by `_time` descending per group, limits each group to 100 rows, then optionally re-sorts for nicer output:

```
docker exec -i influxdb influx query --org example-org --raw <<'FLUX'
from(bucket:"example-bucket")
  |> range(start: -30d)
  |> filter(fn: (r) => r._measurement == "temperature" and r._field == "value")
  |> group(columns: ["sensor_id"])
  |> sort(columns: ["_time"], desc: true)
  |> limit(n: 100)
  |> sort(columns: ["sensor_id","_time"])
FLUX
```

How to check whether `sensor_id` is a tag (InfluxDB)

Flux grouping works best when `sensor_id` is a tag. Run this to list tag keys for the `temperature` measurement:

```
docker exec -i influxdb influx query --org example-org --raw <<'FLUX'
import "influxdata/influxdb/schema"
schema.tagKeys(
  bucket: "example-bucket",
  predicate: (r) => r._measurement == "temperature",
  start: -90d
)
FLUX
```

Module 4 — Docker-ready script

Save this file as `module4_queries.sh`, make it executable and run it. It runs the main examples shown above.

```
#!/usr/bin/env bash
# module4_queries.sh  — Docker-ready (fixed Flux sort)
set -euo pipefail

PG_CONTAINER="timescaledb"
PG_USER="admin"
PG_DB="metricsdb"
INFLUX_CONTAINER="influxdb"
INFLUX_ORG="example-org"
INFLUX_BUCKET="example-bucket"

echo "=== TimescaleDB: Last 15 minutes (sensor_data) ==="
docker exec -i "$PG_CONTAINER" psql -U "$PG_USER" -d "$PG_DB" -c \
  "SELECT time, sensor_id, temperature AS cpu_like FROM sensor_data WHERE time >
NOW() - INTERVAL '15 minutes' ORDER BY time ASC LIMIT 200;"

echo
echo "=== TimescaleDB: Daily averages (last 30 days) ==="
docker exec -i "$PG_CONTAINER" psql -U "$PG_USER" -d "$PG_DB" -c \
  "SELECT time_bucket('1 day', time) AS day, AVG(temperature) AS avg_temp, COUNT(*) AS
samples FROM sensor_data WHERE time > NOW() - INTERVAL '30 days' GROUP BY day
ORDER BY day;"

echo
echo "=== TimescaleDB: Last 100 points per sensor (ROW_NUMBER) ==="
docker exec -i "$PG_CONTAINER" psql -U "$PG_USER" -d "$PG_DB" -c "\
```

```
SELECT sensor_id, time, temperature FROM ( \
  SELECT sensor_id, time, temperature, ROW_NUMBER() OVER (PARTITION BY sensor_id
ORDER BY time DESC) AS rn \
  FROM sensor_data \
) x WHERE rn <= 100 ORDER BY sensor_id, time DESC;"

echo
echo "=== InfluxDB: Last 15 minutes (Flux) ==="
docker exec -i "$INFLUX_CONTAINER" influx query --org "$INFLUX_ORG" --raw <<'FLUX'
from(bucket:"example-bucket")
  |> range(start: -15m)
  |> filter(fn: (r) => r._measurement == "temperature" and r._field == "value")
  |> sort(columns: ["_time"])
FLUX

echo
echo "=== InfluxDB: Daily averages (aggregateWindow) ==="
docker exec -i "$INFLUX_CONTAINER" influx query --org "$INFLUX_ORG" --raw <<'FLUX'
from(bucket:"example-bucket")
  |> range(start: -30d)
  |> filter(fn: (r) => r._measurement == "temperature" and r._field == "value")
  |> aggregateWindow(every: 1d, fn: mean, createEmpty: false)
FLUX

echo
echo "=== InfluxDB: Last 100 points per sensor (group+limit) — FIXED ==="
docker exec -i "$INFLUX_CONTAINER" influx query --org "$INFLUX_ORG" --raw <<'FLUX'
from(bucket:"example-bucket")
  |> range(start: -30d)
  |> filter(fn: (r) => r._measurement == "temperature" and r._field == "value")
  |> group(columns: ["sensor_id"])
  |> sort(columns: ["_time"], desc: true)
  |> limit(n:100)
  |> sort(columns: ["sensor_id","_time"])
FLUX

echo
echo "module4_queries.sh finished."
```

Make and run:

```
chmod +x module4_queries.sh
./module4_queries.sh
```

Cleanup (docker-safe, idempotent)

Stop containers but keep data:

```
docker compose down
```

Reset stack and remove volumes (fresh start):

```
docker compose down -v
docker compose up -d
```

Drop TimescaleDB table/hypertable (inside container, docker):
This attempts to drop `sensor_data` safely if exists. Use with caution.

```
docker exec -i timescaledb psql -U admin -d metricsdb -v ON_ERROR_STOP=1 <<'SQL'
DO $$
BEGIN
  IF EXISTS (SELECT 1 FROM timescaledb_information.hypertables WHERE
hypertable_name='sensor_data') THEN
    EXECUTE format('DROP TABLE IF EXISTS %I CASCADE', 'sensor_data');
  ELSE
    EXECUTE format('DROP TABLE IF EXISTS %I CASCADE', 'sensor_data');
  END IF;
END;
$$;
SQL
```

Reset InfluxDB (simple): remove volumes and recreate services:

```
docker compose down -v
docker compose up -d
```

This deletes InfluxDB data and starts clean.

Short tips

- Use `docker exec -i timescaledb psql ... -c "SQL"` so you run SQL inside the TimescaleDB container without opening an interactive shell.

- Use `docker exec -i influxdb influx query --org ... --raw <<'FLUX' ... FLUX` to run Flux queries non-interactively.
- `time_bucket('1 hour', time)` groups rows into hourly buckets (TimescaleDB). `aggregateWindow(every: 1h, fn: mean)` does the same in Flux and is efficient.
- For “last N points per sensor”, prefer `ROW_NUMBER()` approach in SQL for many sensors; use `group + limit` in Flux.
- Always pick a reasonable `range(start: -Nd)` in Flux — Flux requires an explicit time range to limit the scan.