LEVEL3_Data Modeling

BTTF Data Engineer Assignment

Objective

Design a **relational data model** that supports:

- Combining shipments with weather
- Time-based joins using timestamps
- City-level aggregation (lat/lon or name)
- Efficient KPI analysis (e.g., average fuel consumption by weather)

TOOLS

Task	Tool
Documentation (comments, reasoning, explanation)	**Obsidian**
Schema design + queries	**DBeaver** (or VSCode for writing .sql)
ER Diagram (optional visual)	**Draw.io**
CSV/Weather File preview	**Pandas** or Excel
Raw shipment data	Already in **PostgreSQL** via pg_restore
Merging logic & ETL (optional)	Python or PySpark (if you automate joins later)

Understand the analytics KPIs supported

KPIs to support:

- Average fuel consumption by **temperature range**
- Average fuel consumption by **weather condition**
- Fuel usage by **city**
- Weather pattern impact on deliveries (if we have delivery delays)
- Aggregation by **hour**, **day**, or **weather condition**

Available data sources

Table/File	Source Location	Description
shipments.shipments	PostgreSQL (Restored via pg_restore)	Shipment records, includes timestamp, fuel usage
shipments.cities	PostgreSQL	City master list with coordinates
weather_data_YYYY_MM_DD.csv	data/raw/weather/ (generated)	Hourly weather by city + timestamp

---

Data Model Design

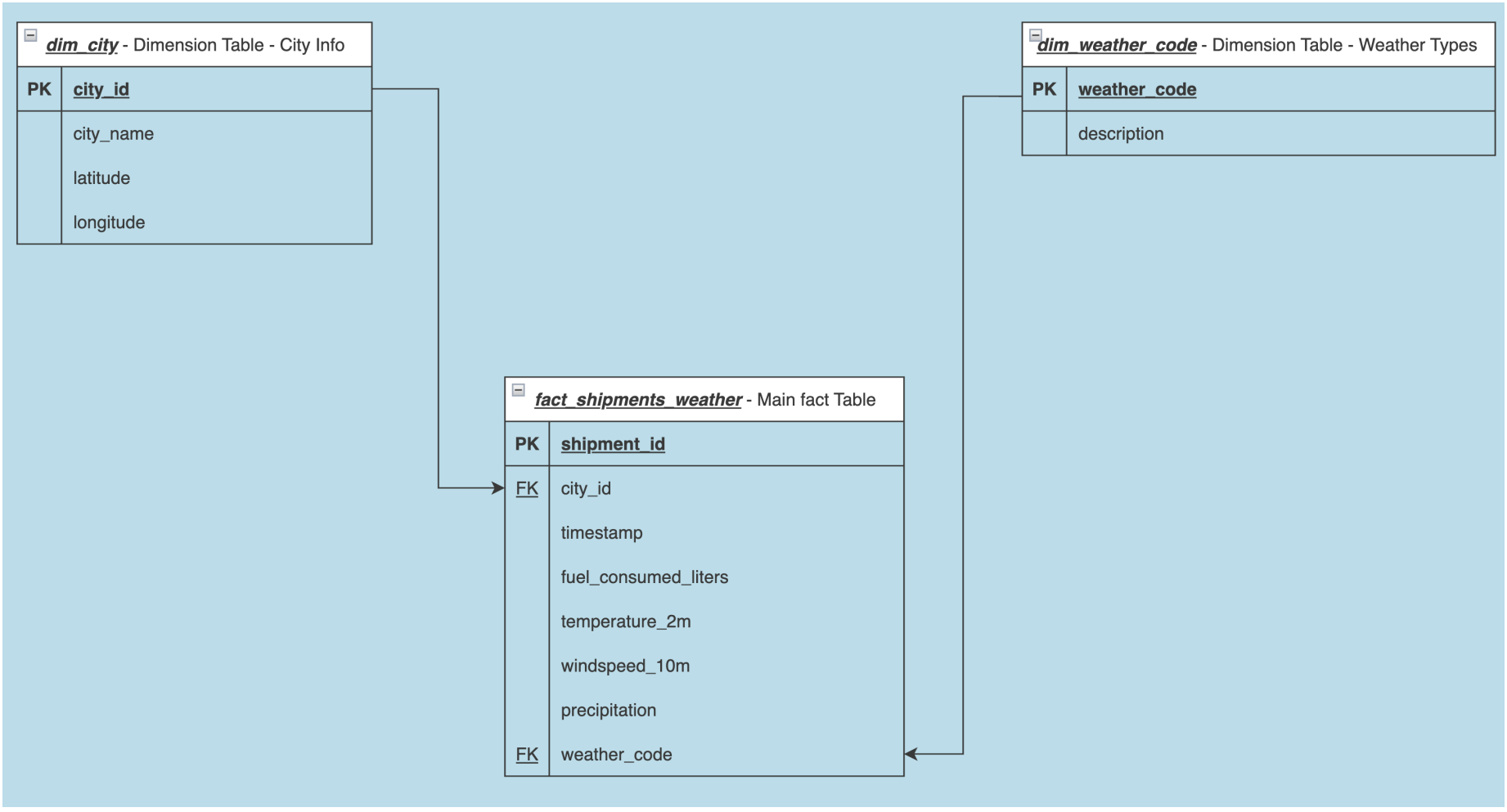
ER Diagram: Enriched Shipment-Star Schema

```
The diagram below models the relationship between shipments and external weather data.

- `fact_shipments_weather` is the central fact table.
- `dim_city` stores city metadata and coordinates.
- `dim_weather_code` decodes categorical weather conditions.
```

Relationships
- fact_shipments_weather.city_id → dim_city.city_id
- fact_shipments_weather.weather_code → dim_weather_code.weather_code

This schema enables scalable analytics on how fuel efficiency varies by location, weather, and time.



SQL SELECT Statements

```sql
-- Dimension: City
CREATE TABLE dim_city (
    city_id SERIAL PRIMARY KEY,
    city_name TEXT NOT NULL,
    latitude NUMERIC,
    longitude NUMERIC
);

-- Optional Dimension: Weather Condition
CREATE TABLE dim_weather_code (
    weather_code INT PRIMARY KEY,
    description TEXT
);
```

```sql
-- Fact Table: Shipment + Weather
CREATE TABLE fact_shipments_weather (
    shipment_id INT,
    city_id INT,
    timestamp TIMESTAMP,
    fuel_consumed_liters NUMERIC,
    temperature_2m NUMERIC,
    windspeed_10m NUMERIC,
    precipitation NUMERIC,
    weather_code INT,

    PRIMARY KEY (shipment_id, timestamp),
    FOREIGN KEY (city_id) REFERENCES dim_city(city_id),
    FOREIGN KEY (weather_code) REFERENCES dim_weather_code(weather_code)
);
```

Join conditions (city, timestamp)

- Join logic for city:
  - `shipments.city_name = weather.city`

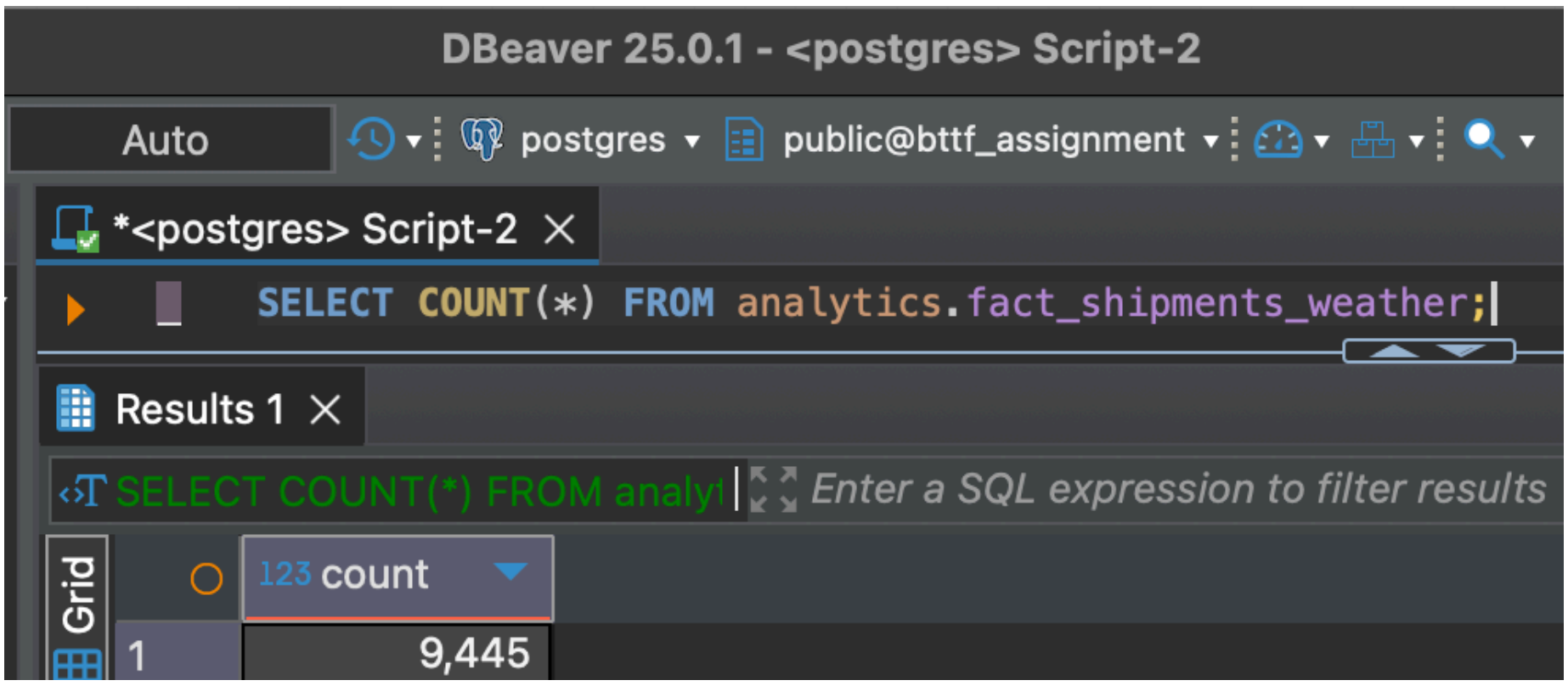
- Or: use `(latitude, longitude)` match if needed
- Join logic for timestamp:
  - Truncate shipment `timestamp` to hourly
  - Join with `weather.timestamp`

Analytical Value of the Model

- Enables filtering shipments by weather condition (e.g., rain vs clear)
- Supports temperature-driven fuel efficiency analysis
- City-level aggregations and comparisons
- Enables temporal trend analysis (by hour/day/week)
- Provides explainability to variations in delivery KPIs

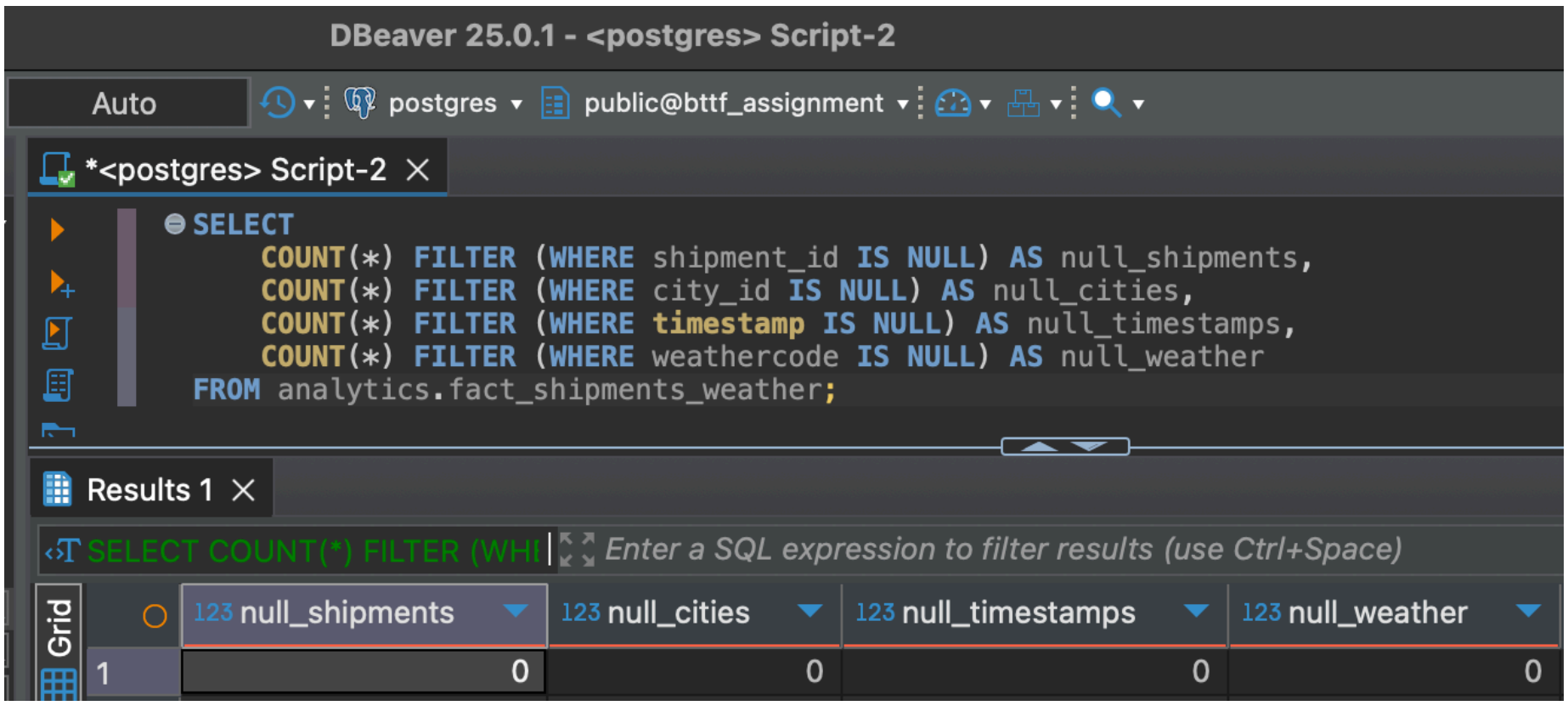
Data Validation for Analytics.fact_shipments_weather

1.1 Basic row count

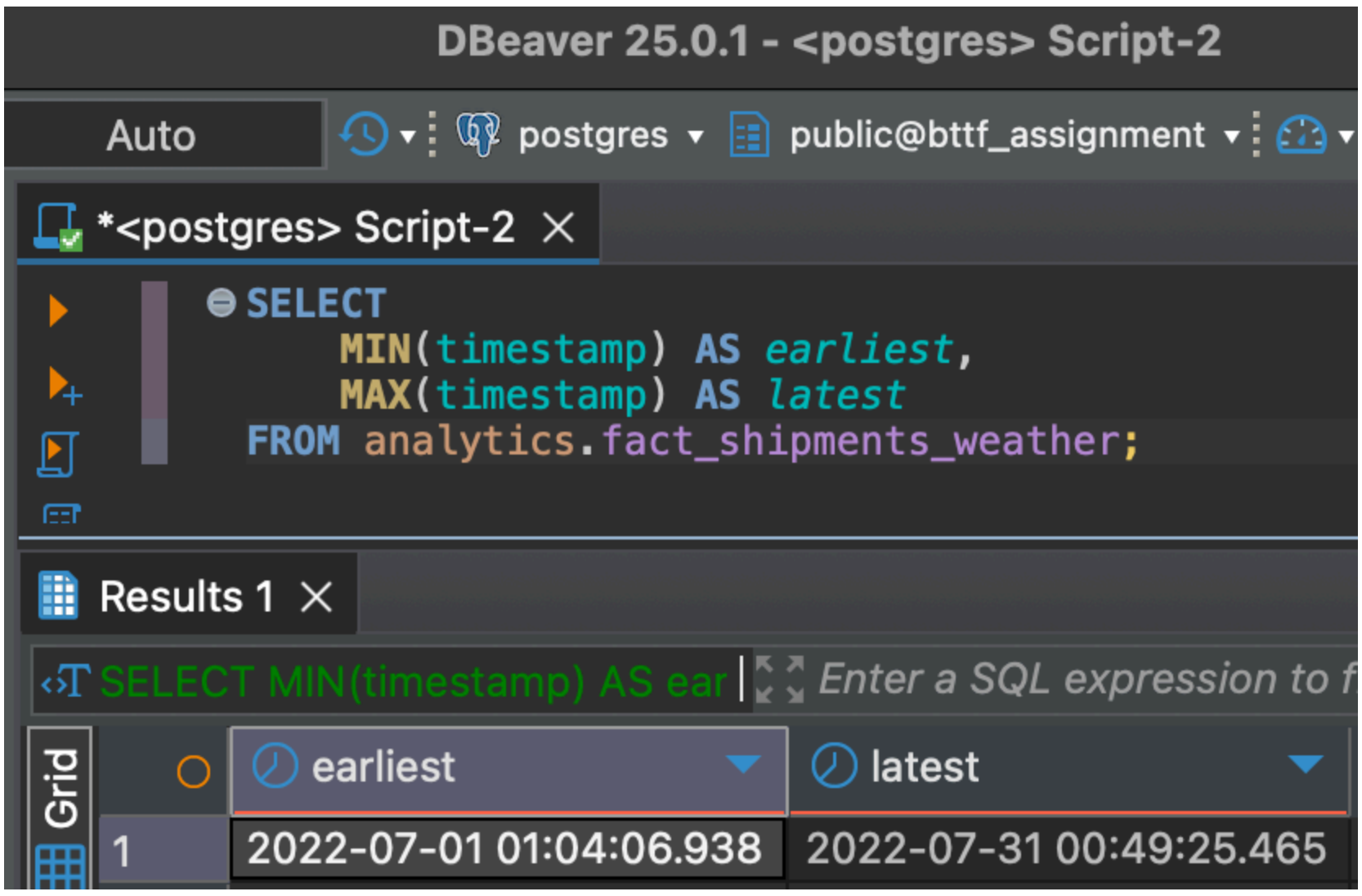


1.2 Check nulls in important fields

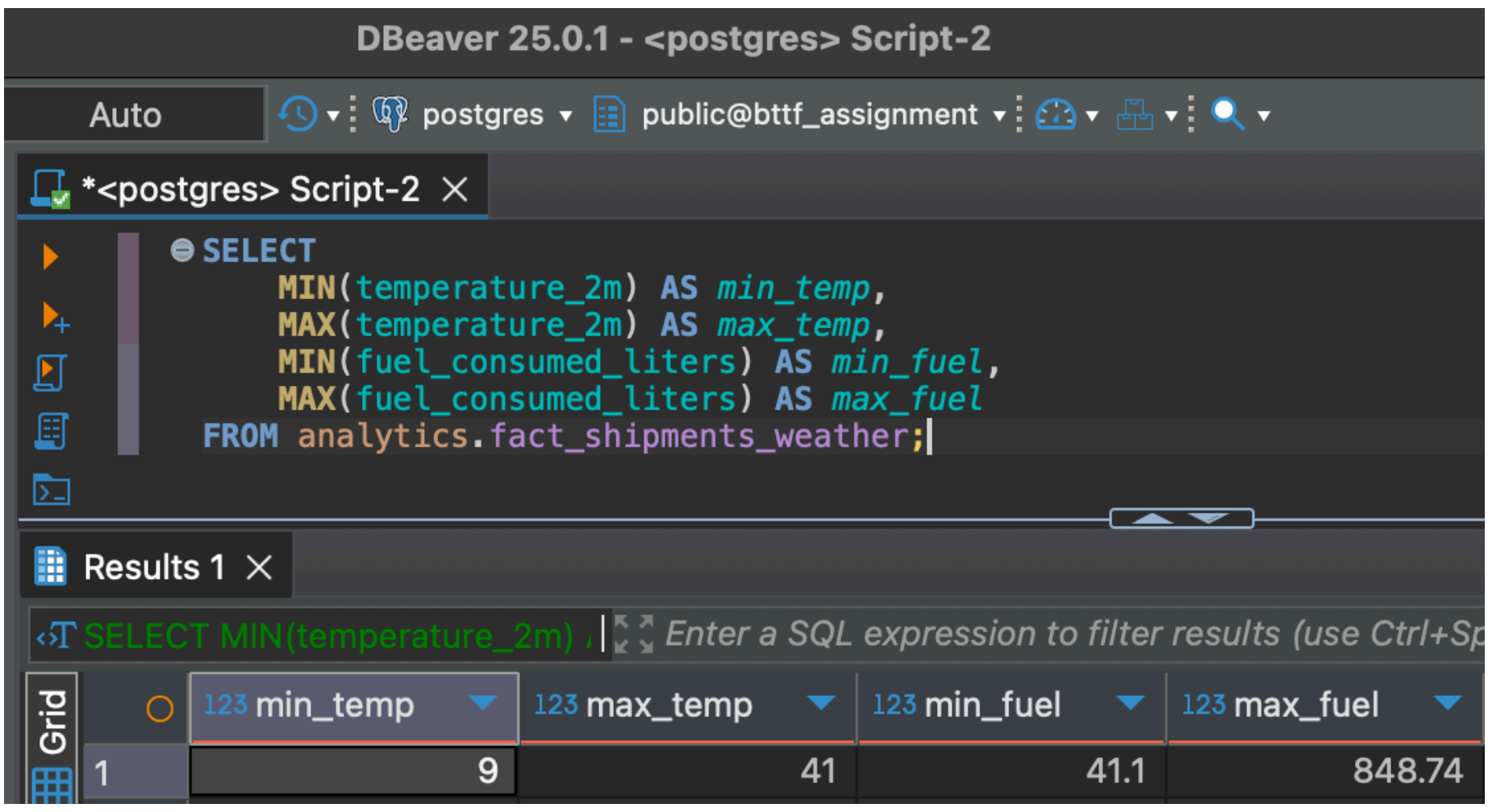
```
SELECT
    COUNT(*) FILTER (WHERE shipment_id IS NULL) AS null_shipments,
    COUNT(*) FILTER (WHERE city_id IS NULL) AS null_cities,
    COUNT(*) FILTER (WHERE timestamp IS NULL) AS null_timestamps,
    COUNT(*) FILTER (WHERE weathercode IS NULL) AS null_weather
FROM analytics.fact_shipments_weather;
```



1.3 Timestamp range sanity



1.4 Temperature and fuel consumed sanity



1.5 Distribution across cities

DBeaver 25.0.1 - <postgres> Script-2

Auto | postgres | public@bttf_assignment

*<postgres> Script-2

```
SELECT city_id, COUNT(*) AS shipments
FROM analytics.fact_shipments_weather
GROUP BY city_id
ORDER BY shipments DESC;
```

fact_shipments_weather 1

SELECT city_id, COUNT(*) AS s | Enter a SQL expression to filter results

Grid | Text

	123 city_id	123 shipments
1	2,673,730	566
2	625,144	551
3	2,643,743	551
4	756,135	550
5	2,911,298	536
6	2,950,159	534
7	3,117,735	532
8	3,128,760	527
9	3,054,643	526
10	3,169,070	520
11	745,044	520
12	683,506	516
13	703,448	510
14	498,817	508
15	323,786	505
16	2,988,507	504
17	524,901	498
18	2,761,369	491

Analytical SQL queries using the analytics.fact_shipments_weather table

1. Average Fuel Consumption by City

DBeaver 25.0.1 - <postgres> Script-2

Auto | postgres | public@bttf_assignment

*<postgres> Script-2

```sql
SELECT
    fsw.city_id,
    dc.id AS city_dim_id,
    dc.name AS city_name,
    ROUND(AVG(fsw.fuel_consumed_liters)::NUMERIC, 2) AS avg_fuel_consumed
FROM analytics.fact_shipments_weather fsw
LEFT JOIN shipments.cities dc ON fsw.city_id = dc.id
GROUP BY fsw.city_id, dc.id, dc.name
ORDER BY avg_fuel_consumed DESC;
```

fact_shipments_weather(+) 1

SELECT fsw.city_id, dc.id AS cit | Enter a SQL expression to filter results (use Ctrl+Space)

Grid | Text

	123 city_id	123 city_dim_id	A-Z city_name	123 avg_fuel_consumed
1	524,901	524,901	Moscow	431.62
2	498,817	498,817	Saint Petersburg	427.44
3	2,673,730	2,673,730	Stockholm	380.43
4	3,117,735	3,117,735	Madrid	359.31
5	2,643,743	2,643,743	London	344.05
6	323,786	323,786	Ankara	324.44
7	2,988,507	2,988,507	Paris	317.81
8	3,128,760	3,128,760	Barcelona	312.55
9	625,144	625,144	Minsk	285.69
10	745,044	745,044	Istanbul	282.11
11	2,911,298	2,911,298	Hamburg	277.33
12	703,448	703,448	Kyiv	276.36
13	2,950,159	2,950,159	Berlin	247.85
14	3,169,070	3,169,070	Rome	245.55
15	756,135	756,135	Warsaw	245.43
16	683,506	683,506	Bucharest	241.43
17	3,054,643	3,054,643	Budapest	218.59
18	2,761,369	2,761,369	Vienna	217.78

2. Hottest Hour per City

DBeaver 25.0.1 - <postgres> Script-2

Auto | postgres | public@bttf_assignment

*<postgres> Script-2

```
SELECT DISTINCT ON (c.name)
    c.name AS city_name,
    fsw.timestamp,
    fsw.temperature_2m
FROM analytics.fact_shipments_weather fsw
JOIN shipments.cities c ON fsw.city_id = c.id
WHERE fsw.temperature_2m IS NOT NULL
ORDER BY c.name, fsw.temperature_2m DESC, fsw.timestamp;
```

cities(+) 1

SELECT DISTINCT ON (c.name) | Enter a SQL expression to filter results (use Ctr

Grid | Text

	city_name	timestamp	temperature_2m
1	Ankara	2022-07-29 16:51:05.573	35
2	Barcelona	2022-07-15 12:10:00.160	34.3
3	Berlin	2022-07-20 15:05:19.829	37.7
4	Bucharest	2022-07-24 15:52:41.946	36.1
5	Budapest	2022-07-21 16:09:57.571	38
6	Hamburg	2022-07-20 17:27:05.262	38.9
7	Istanbul	2022-07-23 19:09:01.143	31
8	Kyiv	2022-07-06 14:58:04.259	32.9
9	London	2022-07-19 13:45:15.961	37.9
10	Madrid	2022-07-14 17:53:37.795	41
11	Minsk	2022-07-02 16:31:24.022	28.4
12	Moscow	2022-07-07 12:28:25.264	30.9
13	Paris	2022-07-19 16:11:34.919	39.1
14	Rome	2022-07-03 15:22:25.770	38
15	Saint Petersburg	2022-07-01 15:11:50.005	30.6
16	Stockholm	2022-07-21 15:34:47.303	31.3
17	Vienna	2022-07-21 16:04:51.711	33.5
18	Warsaw	2022-07-01 16:09:10.713	33.4

3. Rain Impact on Fuel Efficiency

DBeaver 25.0.1 - <postgres> Script-2

Auto | postgres | public@bttf_assignment

*<postgres> Script-2

```
SELECT
    CASE
        WHEN fsw.precipitation > 0 THEN 'Rainy'
        ELSE 'Non-Rainy'
    END AS weather_condition,
    ROUND(AVG(fsw.fuel_consumed_liters)::NUMERIC, 2) AS avg_fuel_consumed,
    COUNT(*) AS shipment_count
FROM analytics.fact_shipments_weather fsw
GROUP BY weather_condition
ORDER BY weather_condition;
```

Results 1

SELECT CASE WHEN fsw.precip | Enter a SQL expression to filter results (use Ctrl+Space)

Grid | Text

	weather_condition	avg_fuel_consumed	shipment_count
1	Non-Rainy	300.88	8,578
2	Rainy	312.69	867

4. Avg Wind Speed vs Fuel Consumption (Correlation Check)

DBeaver 25.0.1 - <postgres> Script-2

Auto | postgres | public@bttf_assignment

*<postgres> Script-2

```sql
SELECT
    ROUND(windspeed_10m)::INT AS wind_speed_bucket,
    ROUND(AVG(fuel_consumed_liters)::NUMERIC, 2) AS avg_fuel_consumed,
    COUNT(*) AS shipment_count
FROM analytics.fact_shipments_weather
GROUP BY wind_speed_bucket
ORDER BY wind_speed_bucket;
```

Results 1

SELECT ROUND(windspeed_10r | Enter a SQL expression to filter results (use Ctrl+Space)

Grid | Text

	123 wind_speed_bucket	123 avg_fuel_consumed	123 shipment_count
1	0	308.45	14
2	1	269.59	49
3	2	294.47	153
4	3	304.11	170
5	4	297.15	345
6	5	302.02	468
7	6	290.93	718
8	7	292.57	623
9	8	304.67	716
10	9	308.8	644
11	10	317.28	680
12	11	310.45	520
13	12	308.28	612
14	13	308.65	478
15	14	309.61	520
16	15	298.9	389
17	16	301.5	417
18	17	300.65	345
19	18	299.68	354
20	19	300.49	227
21	20	287.77	195
22	21	292.54	144
23	22	317.94	162
24	23	308.65	111
25	24	273.34	91
26	25	260.56	42
27	26	266.27	65
28	27	299.08	43
29	28	291.91	55
30	29	272.58	22
31	30	280.43	14
32	31	274.54	11
33	32	257.16	15
34	33	261.79	17
35	34	239.12	8
36	35	306.24	5
37	36	430.41	1
38	41	94.05	2

5. Daily Aggregated Weather Stats (per city)

DBeaver 25.0.1 - <postgres> Script-2

```
SELECT
    c.name AS city_name,
    DATE(fsw.timestamp) AS date,
    ROUND(MIN(fsw.temperature_2m)::NUMERIC, 2) AS min_temp,
    ROUND(MAX(fsw.temperature_2m)::NUMERIC, 2) AS max_temp,
    ROUND(AVG(fsw.temperature_2m)::NUMERIC, 2) AS avg_temp,
    ROUND(AVG(fsw.windspeed_10m)::NUMERIC, 2) AS avg_wind_speed,
    ROUND(SUM(fsw.precipitation)::NUMERIC, 2) AS total_precipitation
FROM analytics.fact_shipments_weather fsw
JOIN shipments.cities c ON fsw.city_id = c.id
GROUP BY c.name, DATE(fsw.timestamp)
ORDER BY c.name, date;
```

cities 1

	city_name	date	min_temp	max_temp	avg_temp	avg_wind_speed	total_precipitation
1	Ankara	2022-07-01	15	23.9	19.78	10.26	9.2
2	Ankara	2022-07-02	13.1	27.3	20.12	10.35	0
3	Ankara	2022-07-03	13.2	26.2	18.6	8.89	0
4	Ankara	2022-07-04	11.2	26.5	19.23	10.65	0
5	Ankara	2022-07-05	12.3	27.1	19.94	10.03	0
6	Ankara	2022-07-06	13.4	28.1	21.2	7.16	0
7	Ankara	2022-07-07	18	30.1	24.75	8.78	0.1
8	Ankara	2022-07-08	16.6	29.5	24.01	5.54	0
9	Ankara	2022-07-09	17.7	29.8	25.05	5.28	1.8
10	Ankara	2022-07-10	15.3	25.3	21.28	13.52	2
11	Ankara	2022-07-11	13.7	26	19.71	9.49	0
12	Ankara	2022-07-12	16.1	28.2	21.48	8.16	0
13	Ankara	2022-07-13	15.2	27	20.54	7.29	0
14	Ankara	2022-07-14	16.6	27.4	22.29	13.06	0
15	Ankara	2022-07-15	14.2	27.2	21.74	11.62	0
16	Ankara	2022-07-16	15.7	29.6	20.98	7.12	0
17	Ankara	2022-07-17	15.6	30.8	23.88	13.27	0
18	Ankara	2022-07-18	15.6	27.5	21.37	14.96	0
19	Ankara	2022-07-19	15.8	26.6	20.92	15.84	0
20	Ankara	2022-07-20	13.9	27.8	19.81	12.25	0
21	Ankara	2022-07-21	13.7	28.4	22.53	10.66	0
22	Ankara	2022-07-22	15.5	29.4	26	12.5	0
23	Ankara	2022-07-23	16.3	28.8	24.31	11.31	0
24	Ankara	2022-07-24	14.2	29	21.38	7.73	0
25	Ankara	2022-07-25	14.5	30.6	23.02	7.52	0
26	Ankara	2022-07-26	16.2	29.4	21.76	7.21	0
27	Ankara	2022-07-27	16	32	25	7.01	0
28	Ankara	2022-07-28	17.6	33.8	26.43	6.28	0
29	Ankara	2022-07-29	18.6	35	26.21	7.56	0
30	Ankara	2022-07-30	18.5	34.2	27.91	9.21	0
31	Barcelona	2022-07-01	20.7	25.1	23.24	9.68	0
32	Barcelona	2022-07-02	19.5	26.8	22.79	7.24	0
33	Barcelona	2022-07-03	20.5	27.7	24.01	5.44	0
34	Barcelona	2022-07-04	22.1	30	26.21	11.08	0
35	Barcelona	2022-07-05	22.6	29.9	26.51	7.33	0
36	Barcelona	2022-07-06	22.3	29.8	24.72	9.65	17
37	Barcelona	2022-07-07	20.5	29.2	24.54	9.95	0.2
38	Barcelona	2022-07-08	21.1	28.6	24.43	8.94	0

KPI-style SQL analytical queries

1. Average Fuel Consumption per Shipment (Overall)

*<postgres> Script-2

```sql
SELECT
    ROUND(AVG(fuel_consumed_liters)::NUMERIC, 2) AS avg_fuel_consumed
FROM analytics.fact_shipments_weather;
```

Results 1

SELECT ROUND(AVG(fuel_consumed_liters)::NUM | Enter a SQL expression to filter resul

Grid

	123 avg_fuel_consumed
1	301.97

2. City with Highest Average Fuel Consumption

*<postgres> Script-2

```sql
SELECT
    c.name AS city_name,
    ROUND(AVG(fsw.fuel_consumed_liters)::NUMERIC, 2) AS avg_fuel_consumed
FROM analytics.fact_shipments_weather fsw
JOIN shipments.cities c ON fsw.city_id = c.id
GROUP BY c.name
ORDER BY avg_fuel_consumed DESC
LIMIT 1;
```

cities 1

SELECT c.name AS city_name, ROUND(AVG(fsw.fu | Enter a SQL expression to filter results (use

Grid

	A-Z city_name	123 avg_fuel_consumed
1	Moscow	431.62

3. Hottest Hour Overall

*<postgres> Script-2

```sql
SELECT
    c.name AS city_name,
    fsw.timestamp,
    ROUND(fsw.temperature_2m::NUMERIC, 2) AS temperature
FROM analytics.fact_shipments_weather fsw
JOIN shipments.cities c ON fsw.city_id = c.id
ORDER BY fsw.temperature_2m DESC
LIMIT 1;
```

cities(+) 1

SELECT c.name AS city_name, fsw.timestamp, RO | Enter a SQL expression

Grid

	A-Z city_name	timestamp	123 temperature
1	Madrid	2022-07-14 17:56:19.882	41

4. Total Precipitation per City

*<postgres> Script-2

```sql
SELECT
    c.name AS city_name,
    ROUND(SUM(fsw.precipitation)::NUMERIC, 2) AS total_precipitation
FROM analytics.fact_shipments_weather fsw
JOIN shipments.cities c ON fsw.city_id = c.id
GROUP BY c.name
ORDER BY total_precipitation DESC;
```

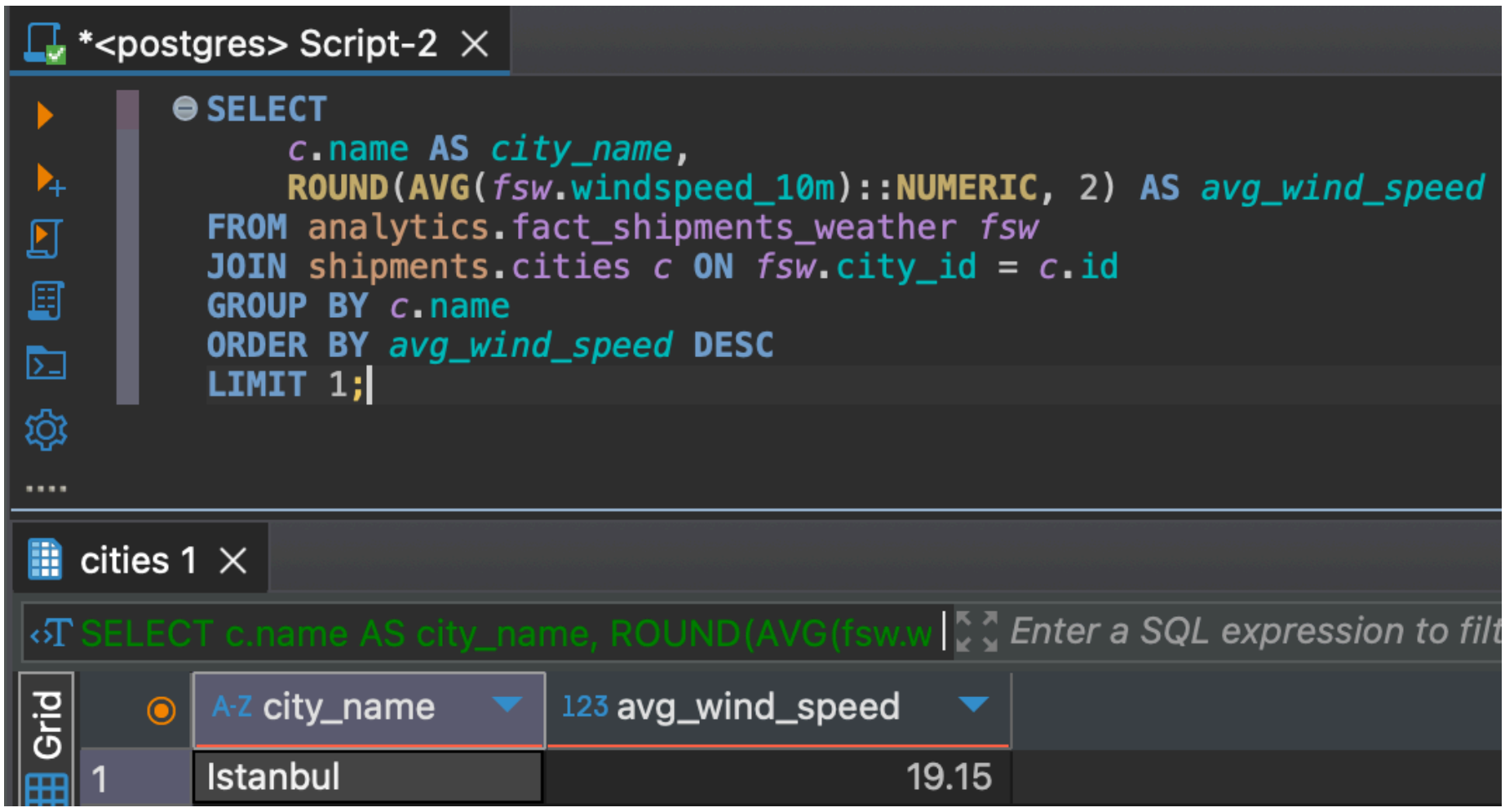
cities 1

SELECT c.name AS city_name, ROUND(SUM(fsw.p | Enter a SQL expression to filter res

Grid | Text

	A-Z city_name	123 total_precipitation
1	Minsk	104.1
2	Warsaw	97.6
3	Kyiv	73
4	Saint Petersburg	66.8
5	Barcelona	33.5
6	Stockholm	32.8
7	Vienna	30.4
8	Berlin	29.8
9	Moscow	27.6
10	Hamburg	27.4
11	Bucharest	24.8
12	Ankara	13.1
13	Istanbul	9.7
14	Budapest	9.6
15	Paris	8.7
16	London	7.2
17	Rome	5.6
18	Madrid	0

Windy City (Highest Avg Wind Speed)



Next Step

→ Proceed to LEVEL4_Data_Pipeline_Design - a unified data model with schema definitions that integrates shipment and weather data to support analytical use cases like fuel efficiency, city-wise performance, and weather-based metrics.