# InclusiveSpaces (CAT) — Database Operations Guide

***Scope: Supabase Postgres + PostGIS + pgRouting operations for routing + comfort layers***

## 0) Big picture (what lives where)

### 0.1 What the backend needs (hard requirement)

Per city, the API expects a **pgRouting-style graph**:

- **Edges table**: <prefix>_ways
    - required columns (minimum): gid, source, target, cost, the_geom
    - plus **many *_weight columns** used in /api/accessibility cost expression
- **Vertices table**: <prefix>_ways_vertices_pgr
    - required columns (minimum): id, the_geom (Point)

### 0.2 What is NOT required in DB (for frontend visualization)

Most visualization layers are loaded in the frontend from public/data/<city>/... or WMS.

---

## 1) Prerequisites (install once)

### 1.1 Tools (install once per machine)

**Environment**

- Windows 11 + WSL2 (Ubuntu 22.04+) (recommended)
- Or Linux (Ubuntu)

**Required for CAT database operations (OSM + PostGIS uploads + CLI access)**

- psql (PostgreSQL client only)
    - Used to connect to Supabase Postgres and run SQL scripts.
- GDAL (ogr2ogr)
    - Used to upload GeoPackage/GeoJSON layers into PostGIS.
- osmium-tool (osmium)
    - Used to extract city OSM data from a large .osm.pbf file and convert it to .osm.
- osm2pgrouting
    - Used to import .osm into pgRouting tables (<>_ways, <>_ways_vertices_pgr).

Optional (recommended for visual checks / fixing CRS & geometry)

- QGIS: Use it to check CRS (= EPSG:4326), visualize and inspect layers before upload.

---

## 1.2 Install tools

**Windows + WSL2**

1. **Install WSL2 + Ubuntu**

- Windows PowerShell (Admin):

```
wsl --install -d Ubuntu
```

- Restart if prompted, then open **Ubuntu** terminal.

2. **Install the required CLI tools inside Ubuntu (WSL)**

```
sudo apt update
sudo apt install -y postgresql-client gdal-bin osmium-tool osm2pgrouting
```

3. **Install QGIS (recommended)**

- **Preferred (simplest):** install **QGIS on Windows** using the official installer (GUI runs smoother than inside WSL).

---

## 1.3 Verify installations

- Run:

```
psql --version
ogr2ogr --version
osmium --version
osm2pgrouting --version
```

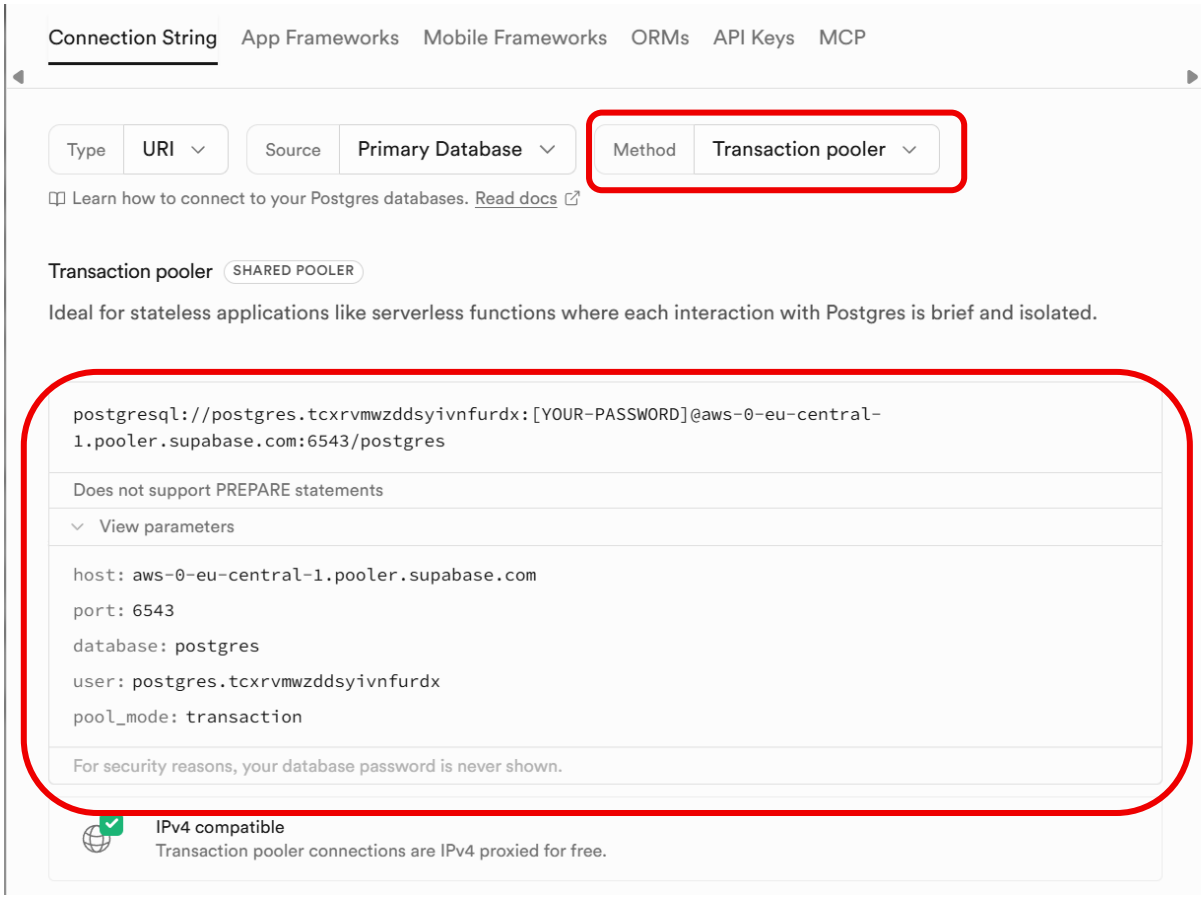
---

# 2) Supabase connection

## 2.1 Get the connection parameters

```
Host: aws-0-eu-central-1.pooler.supabase.com
Port: 6543
Database: postgres
User: postgres.tcxrvmwzddsyivnfurdx
Pool mode: transaction
```

You can also check from: **Supabase Dashboard →** Connect




## 2.2 Use the Pooler connection string

Use this format (replace [YOUR-PASSWORD])

```
postgresql://postgres.tcxrvmwzddsyivnfurdx: [YOUR-PASSWORD] @aws-0-eu-central-1.pooler.supabase.com:6543/postgres
```

## 2.3 Quick connection test (psql)

```
psql " postgresql://postgres.tcxrvmwzddsyivnfurdx: [YOUR-PASSWORD] @aws-0-eu-central-1.pooler.supabase.com:6543/postgres "
```

[YOUR-PASSWORD]: incspace123456

---

# 4) Database initialization (run once per DB)

Run in Supabase SQL editor or psql:

```
CREATE EXTENSION IF NOT EXISTS postgis;
CREATE EXTENSION IF NOT EXISTS pgrouting;
```

Verify:

```
SELECT PostGIS_Version();
SELECT pgr_version();
```

---

# 5) Routing graph build (per city) — OSM → osmium → osm2pgrouting

This is the core of /api/accessibility.js

## 5.1 Inputs you must have (choose based on data availability)

**Data needed**

- **An OSM extract in .osm.pbf or .osm** that contains the city road network.

**Optional: only needed if you must cut a smaller area (common for small cities / districts)**

- **City boundary polygon** in GeoJSON, CRS EPSG:4326, valid Polygon/MultiPolygon.
  (Used for osmium extract --polygon …)

---

## 5.2 Get OSM data (pick one route)

**Route A — City-level extract available (recommended if available)**

- Download a **city-level .osm.pbf** from an extract provider (e.g., Geofabrik). (Geofabrik Download Server)
  Result: <city>.osm.pbf

**Route B — No city-level extract (use regional + clip)**

- Download a **regional .osm.pbf** (country/region-level) from an extract provider such as Geofabrik. (Geofabrik Download Server)
  Example: greece-latest.osm.pbf (for Penteli-style cases)
- Ensure you have <city>_boundary.geojson (EPSG:4326, valid polygon).

---

## 5.3 Extract city OSM using osmium (ONLY for Route B)

Syntax:

```
cd /path/to/resource/data/<City>
osmium extract --polygon <city>_boundary.geojson <region>.osm.pbf -o
<city>.osm.pbf
```

## 5.4 Convert to .osm (do this for both Route A and Route B) :

Syntax:

```
osmium cat <city>.osm.pbf -o <city>.osm --overwrite
```

Example:

```
osmium extract --polygon penteli_boundary.geojson greece-latest.osm.pbf -o
penteli.osm.pbf
osmium cat penteli.osm.pbf -o penteli.osm --overwrite
```

Why convert to .osm? osm2pgrouting expects OSM XML; passing .osm.pbf often fails or creates empty tables.

## 5.5 Import using osm2pgrouting (table prefix matters)

osm2pgrouting supports --schema and --prefix (GitHub).

Use --prefix to point to cities, so tables become <prefix>_ways and <prefix>_ways_vertices_pgr:

```
osm2pgrouting \
  -f <city>.osm \
  -d postgres \
  -U postgres.tcxrvmwzddsyivnfurdx \
  -W <DB_PASSWORD> \
  -h aws-0-eu-central-1.pooler.supabase.com \
  -p 6543 \
  --schema public \
  --prefix <prefix> \
  --clean
```

<DB_PASSWORD>: incspace123456

## 5.6 Verify routing tables immediately

In SQL:

```
SELECT COUNT(*) AS n_edges FROM <prefix>_ways;
SELECT COUNT(*) AS n_vertices FROM <prefix>_ways_vertices_pgr;

-- Check geometry SRID (expect 4326 in most CAT workflows)
SELECT ST_SRID(the_geom) FROM <prefix>_ways LIMIT 5;
SELECT ST_SRID(the_geom) FROM <prefix>_ways_vertices_pgr LIMIT 5;
```

If SRID not 4326: decide one standard and keep it consistent with your API nearest-vertex query.

### 5.7 Indexing (strongly recommended for speeding up) :

```
CREATE INDEX IF NOT EXISTS <prefix>_ways_geom_gist ON <prefix>_ways USING GIST
(the_geom);
CREATE INDEX IF NOT EXISTS <prefix>_ways_source_idx ON <prefix>_ways (source);
CREATE INDEX IF NOT EXISTS <prefix>_ways_target_idx ON <prefix>_ways (target);
CREATE INDEX IF NOT EXISTS <prefix>_vertices_geom_gist ON
<prefix>_ways_vertices_pgr USING GIST (the_geom);

ANALYZE <prefix>_ways;
ANALYZE <prefix>_ways_vertices_pgr;
```

---

## 6) Upload comfort/environment layers to PostGIS

Syntax:

```
ogr2ogr -f "PostgreSQL" \
   PG:"host=aws-0-eu-central-1.pooler.supabase.com
user=postgres.tcxrvmwzddsyivnfurdx dbname=postgres password=<DB_PASSWORD>
port=6543" \
   /path/input.geojson \
   -nln <target_table> \
   -progress
```

<DB_PASSWORD>: incspace123456

---

## 7) *_weight columns on <prefix>_ways (required for comfort routing) — schema + population

### 7.1 What these columns mean (IMPORTANT)

In pages/api/accessibility.js, routing cost is computed like:

- If a DB column indicates the factor is “present / active”, then the SQL uses the **front-end user value** as a multiplier.
- Otherwise it uses 1 (neutral, no penalty) when calculates the reachable area.
- Whether the factor is “present / active” depends on how strong the environment factor is. (e.g. noise level > 65db → noise_weight = 1)

So **DB columns must be binary flags (0/1)**, it only says whether a factor applies on that edge.

There are **two types** of flags, matching the current accessibility.js CASE logic:

**Type A — Negative factor (bad = 1, neutral = 0)**

noise_weight, trafficLight_weight, temp_weight_s, temp_weight_w, path_width_weight, stair_weight, obstacle_weight, slope_weight, uneven_surfaces_weight, poor_pavement_weight, kerbs_h_weight, pedestrian_flow_weight

**Type B — Positive factor (good = 1, missing/negative = 0)**

light_weight, tactile_weight, tree_weigh, green_weight, station_weight, wc_d_weight, facilities_weight

## 7.2 Add missing columns (safe + repeatable)

Use defaults 1.0 (positive environmental factor) or 0.0 (negative environmental factor) so routing still works even if you haven't computed a factor.

```
ALTER TABLE <prefix>_ways
  ADD COLUMN IF NOT EXISTS noise_weight double precision NOT NULL DEFAULT 0.0,
  ADD COLUMN IF NOT EXISTS light_weight double precision NOT NULL DEFAULT 1.0,
  ADD COLUMN IF NOT EXISTS trafficLight_weight double precision NOT NULL DEFAULT 0.0,
  ADD COLUMN IF NOT EXISTS tactile_weight double precision NOT NULL DEFAULT 1.0,
  ADD COLUMN IF NOT EXISTS tree_weight double precision NOT NULL DEFAULT 1.0,
  ADD COLUMN IF NOT EXISTS temp_weight_s double precision NOT NULL DEFAULT 0.0,
  ADD COLUMN IF NOT EXISTS temp_weight_w double precision NOT NULL DEFAULT 0.0,
  ADD COLUMN IF NOT EXISTS blue_weight double precision NOT NULL DEFAULT 1.0,
  ADD COLUMN IF NOT EXISTS green_weight double precision NOT NULL DEFAULT 1.0,
  ADD COLUMN IF NOT EXISTS station_weight double precision NOT NULL DEFAULT 1.0,
  ADD COLUMN IF NOT EXISTS wc_d_weight double precision NOT NULL DEFAULT 1.0,
  ADD COLUMN IF NOT EXISTS path_width_weight double precision NOT NULL DEFAULT 0.0,
  ADD COLUMN IF NOT EXISTS stair_weight double precision NOT NULL DEFAULT 0.0,
  ADD COLUMN IF NOT EXISTS obstacle_weight double precision NOT NULL DEFAULT 0.0,
  ADD COLUMN IF NOT EXISTS slope_weight double precision NOT NULL DEFAULT 0.0,
  ADD COLUMN IF NOT EXISTS uneven_surfaces_weight double precision NOT NULL DEFAULT 0.0,
  ADD COLUMN IF NOT EXISTS poor_pavement_weight double precision NOT NULL DEFAULT 0.0,
  ADD COLUMN IF NOT EXISTS kerbs_h_weight double precision NOT NULL DEFAULT 0.0,
  ADD COLUMN IF NOT EXISTS facilities_weight double precision NOT NULL DEFAULT 1.0,
  ADD COLUMN IF NOT EXISTS pedestrian_flow_weight double precision NOT NULL DEFAULT 0.0
```

# 8) Populate flags from layers (binary 0/1) — the "how"

## 8.1 General workflow (per factor)

1. **Import the layer** to DB (Step 6) → a table like streetlight, noise_distribution, obstacle, etc.
2. Ensure layer has a geometry column geom, EPSG is 4326
3. Add spatial index to speed up updates:

```
CREATE INDEX IF NOT EXISTS <layer>_geom_gix ON <layer> USING GIST (geom);
```

4. Run an UPDATE <prefix>_ways ... to set the corresponding *_weight column to **0/1**.

---

## 8.2 Pattern A — Point layers (distance-based → set 0/1)

**Rule of thumb (buffer / distance):**

For point layers (e.g., **trees**, **street lights**), when relating points to routing edges, a **20 m** distance threshold usually works well (i.e., treat an edge as "affected/covered" if a point is within ~20 m).

Example: **lighting** as environmental factor

(as positive factor → default=1; if no streetlight nearby, discomfort → value = 0)

```
UPDATE <prefix>_ways w
SET light_weight =
  CASE
    WHEN EXISTS (
      SELECT 1
      FROM streetlight s
      WHERE ST_DWithin(w.the_geom::geography, s.geom::geography, 20)
    )
    THEN 1.0
    ELSE 0.0
  END;
```

---

## 8.3 Pattern B — Polygon zones (intersect-based → set 0/1)

**Rule of thumb (two polygon types):**

1. **Scattered patches that rarely touch the road geometry**

   Example: **green infrastructure** areas.

   → Use a **20 m buffer** of polygons and then test intersection.

2. **Full-coverage surfaces over the whole city**

Example: **noise distribution**, **temperature heatmaps**.

→ Prefer **threshold filtering** (e.g., dB_level >= 65, -2< thermal_comfort_level <2) and then set flags based on the thresholded zones.

Example: **noise** zones with attribute dB_level

(negative factor → default=0; if in a high-noise zone → discomfort, value set to 1)

```
UPDATE <prefix>_ways w
SET noise_weight =
  CASE
    WHEN EXISTS (
      SELECT 0
      FROM noise_distribution n
      WHERE ST_Intersects(w.the_geom, n.geom)
        AND n. dB_level >= 65      -- threshold policy (adjust per dataset)
    )
    THEN 1.0
    ELSE 0.0
  END;
```

---

## 8.4 Pattern C — Line layers (buffer then intersect → set 0/1)

**Rule of thumb (buffer):**

For line layers, it's recommended to apply a **20 m buffer** before checking intersection with routing edges (lines often represent a feature corridor rather than an exact road surface match)

Example: **tactile** lines buffered by 10m

(positive factor → default=1 (means tactile support exists); if no tactile support → set to 0)

```
UPDATE <prefix>_ways w
SET tactile_weight =
  CASE
    WHEN EXISTS (
      SELECT 1
      FROM tactile_lines t
      WHERE ST_Intersects(
        w.the_geom,
        ST_Buffer(t.geom::geography, 20)::geometry
      )
    )
    THEN 1.0
    ELSE 0.0
  END;
```

---

## 9 Cleanup (recommended: keep Supabase DB lean)

After you have successfully populated all *_weight flags on <prefix>_ways, **remove the imported environment layer tables** (keep a local backup if needed) to keep the Supabase database tidy and within the **free storage quota**.

**Recommended to keep on Supabase (per city)**

- <prefix>_ways
- <prefix>_ways_vertices_pgr

**Then clean up storage**

- Drop the environment layer tables you no longer need (e.g., noise_distribution, streetlight…).
- Run VACUUM (and optionally ANALYZE) to reclaim space and refresh stats: )

```
VACUUM (ANALYZE) <prefix>_ways;
VACUUM (ANALYZE) <prefix>_ways_vertices_pgr;
```

(If VACUUM is restricted by permissions in your Supabase setup, skip it—dropping tables still saves most space.)

---