SA_week1_OSM_import

May 29, 2025

1 Import with tag filters and parallelism

None of these parallelisms and flat-nodes stuff (and actually separate columns extracts) really make sense for small imports

1.1 Flex config for Cyprus

cyprus.lua:

```
-- Cyprus road extract - Flex backend, string maxspeed
local roads = osm2pgsql.define_way_table('roads', {
  { column = 'highway',  type = 'text'      },
  { column = 'name',     type = 'text'      },
  { column = 'maxspeed', type = 'text'      },
  { column = 'geom',     type = 'linestring', projection = 3857 }
})

function osm2pgsql.process_way(object)
  local hw = object.tags.highway
  if not hw or hw:match('^(footway|path|track)$') then return end

  local geom = object:as_linestring()
  if not geom then return end

  roads:insert{
    highway  = hw,
    name     = object.tags.name,
    maxspeed = object.tags.maxspeed,
    geom     = geom
  }
end
```

1.2 Tag-filter tips (ingestion triage)

Goal	Flex-Lua snippet
Drop service/driveway tracks	`if object.tags.highway == 'service' then return end`
Keep only named POIs	`if not object.tags.name then return end`

Goal	Flex-Lua snippet
Store full `addr:*` tags in one JSONB column	`{column='addr', type='jsonb', not_null=false}`

Community Q&A on tag filters

1.3 Import

```
osm2pgsql \
  --create --slim \
  --number-processes 4 \
  --cache 2000 \
  --output flex \
  --style "$HOME/geo/data/cyprus.lua" \
  --flat-nodes "$HOME/geo/flat-nodes.bin" \
  --host=127.0.0.1 --port=5432 \
  --user postgres --database osm \
  "$HOME/geo/data/cyprus-latest.osm.pbf"
```

Clean osm db should look like:

```
ostgres@127:osm> \d
+--------+----------------------+-------+----------+
| Schema | Name                 | Type  | Owner    |
|--------+----------------------+-------+----------/
| public | geography_columns    | view  | postgres |
| public | geometry_columns     | view  | postgres |
| public | osm2pgsql_properties | table | postgres |
| public | planet_osm_rels      | table | postgres |
| public | planet_osm_ways      | table | postgres |
| public | roads                | table | postgres |
| public | spatial_ref_sys      | table | postgres |
+--------+----------------------+-------+----------+
SELECT 7
postgres@127:osm> \d roads
+----------+--------------------------+-----------+
| Column   | Type                     | Modifiers |
|----------+--------------------------+-----------/
| way_id   | bigint                   |  not null |
| highway  | text                     |           |
| name     | text                     |           |
| maxspeed | text                     |           |
| geom     | geometry(LineString,3857) |           |
+----------+--------------------------+-----------+
Indexes:
    "roads_geom_idx" gist (geom)
    "roads_way_id_idx" btree (way_id)
Triggers:
```

```
    roads_osm2pgsql_valid BEFORE INSERT OR UPDATE ON roads FOR EACH ROW EXECUTE FUNCTION roads_
postgres@127:osm>
```

2 Indexes

2.1 Simple / classic (highway example)

Without index:

```
postgres@127:osm> EXPLAIN ANALYZE
 SELECT * FROM roads WHERE highway IN ('primary','secondary') LIMIT 2000;
...
| QUERY PLAN
...
| Limit  (cost=0.00..1503.54 rows=2000 width=240)
    (actual time=4.161..43.076 rows=2000 loops=1)
|   ->  Seq Scan on roads  (cost=0.00..7921.38 rows=10537 width=240)
    (actual time=4.159..42.929 rows=2000 loops=1)
...
|         Rows Removed by Filter: 38443
| Planning Time: 7.192 ms
| Execution Time: 43.186 ms
...
Time: 0.058s
postgres@127:osm>
```

Adding index:

```
CREATE INDEX roads_highway_idx ON roads (highway);
ANALYZE roads;          -- refresh planner stats
```

Result:

```
postgres@127:osm> EXPLAIN ANALYZE
 SELECT * FROM roads WHERE highway IN ('primary','secondary') LIMIT 2000;
...
| QUERY PLAN
| Limit  (cost=120.97..1258.28 rows=2000 width=240)
    (actual time=1.245..7.939 rows=2000 loops=1)
|   ->  Bitmap Heap Scan on roads  (cost=120.97..6018.49 rows=10371 width=240)
    (actual time=1.244..7.671 rows=2000 loops=1)    ...
|         ->  Bitmap Index Scan on roads_highway_idx  (cost=0.00..118.38 rows=10371 width=0)
    (actual time=0.718..0.719 rows=10466 loops=1)
...
| Planning Time: 0.438 ms
Time: 0.015s
postgres@127:osm>
```

→ can see up to ×10 execution speedup, but really hard to reproduce on such low volumes

2.2 maxspeed

Use-case	Recommended index	Example query
Find obvious numeric outliers (200 km/h)	`sql CREATE INDEX roads_speed_num_idx ON roads ((maxspeed::int)) WHERE maxspeed ~ '^\d+$';`	`SELECT * FROM roads WHERE maxspeed::int >= 200;`
Case-insensitive exact text match	`sql CREATE INDEX roads_speed_txt_idx ON roads (maxspeed text_pattern_ops);`	`WHERE maxspeed ILIKE 'signals'`
Fuzzy search / autocomplete	`sql CREATE EXTENSION IF NOT EXISTS pg_trgm; CREATE INDEX roads_speed_trgm_idx ON roads USING gin (maxspeed gin_trgm_ops);`	`WHERE maxspeed % '80'`

(Note on the last row: check the trgm similarity stuff later)

2.3 geom & SP-GiST

Without the SP-GiST index:

```
-- \timing on
postgres@127:osm> EXPLAIN ANALYZE
 SELECT *
 FROM   roads
 WHERE  geom && ST_Expand('SRID=3857;POINT(3670300 4187800)'::geometry, 200);
...
| Gather  (cost=1000.00..7600.79 rows=3 width=239)
    (actual time=22.095..40.347 rows=6 loops=1)
...
|   ->  Parallel Seq Scan on roads  (cost=0.00..6600.49 rows=1 width=239)
    (actual time=14.446..25.135 rows=2 loops=3)
...
|         Rows Removed by Filter: 60155
| Planning Time: 0.106 ms
Time: 0.050s
postgres@127:osm>
```

Adding index:

```
CREATE INDEX roads_geom_spgist
    ON roads USING spgist (geom);
ANALYZE roads;          -- refresh planner stats
```

Result:

```
postgres@127:osm> EXPLAIN ANALYZE
 SELECT *
 FROM   roads
 WHERE  geom && ST_Expand('SRID=3857;POINT(3670300 4187800)'::geometry, 200);
...
```

```
| Bitmap Heap Scan on roads  (cost=4.31..16.14 rows=3 width=243)
    (actual time=0.208..0.214 rows=6 loops=1)
...
|   ->  Bitmap Index Scan on roads_geom_spgist  (cost=0.00..4.30 rows=3 width=0)
    (actual time=0.202..0.202 rows=6 loops=1)
...
| Planning Time: 1.069 ms
Time: 0.012s
postgres@127:osm>
```

→ **×100 execution speedup**

3 A note on coords and conversions

3.1 Some basics

The SRID (Spatial Reference System Identifier) `3857` = "WGS 84 / Web Mercator"—the projection used by most modern slippy-map tile service. Coordinates are in metres east/north of the projection origin, not in lat/lon degrees.

Origin is where the equator meets the prime meridian (0 °, 0 °) after projection → coordinate (0 , 0).

osm2pgsql re-projects each way from its original lon/lat (EPSG 4326) into Web Mercator.

The coordinate differences behave like metres near the equator, but you would not trust `ST_Distance(geom1, geom2)` in SRID 3857 for accurate road-length reports. For metric lengths or areas you would normally re-project on-the-fly, e.g.:

```
-- length in real metres using UTM 36N (EPSG 32636 covers Cyprus)
SELECT ST_Length(
         ST_Transform(geom, 32636)
       ) AS true_length_m
FROM roads
WHERE way_id = 12345;
```

3.2 When is this useful?

- **Tile math** – Web maps cut the square world into 256 × 256 px tiles whose edges are exactly powers of 2 in EPSG 3857 metres.
- **Fast spatial filtering** – bounding boxes index nicely because X/Y grow linearly.
- **Mix-and-match with web imagery** – most satellite and OSM raster layers ship in 3857, so you avoid re-projection CPU when overlaying vector data on tiles.

3.3 Details on powers of 2 in Web-Mercator tiling

The standard slippy-map scheme (OSM, Google, Mapbox, Leaflet, etc.) fixes three things:

Constant	Value
Tile size in pixels	256 px × 256 px
World extent in EPSG 3857 metres	−20 037 508.342 789 2 m … +20 037 508.342 789 2 m on both X and Y axes (40 075 016.685 6 m square)
Zoom levels	non-negative integers $z = 0, 1, 2…$

Because every zoom level doubles the map's resolution, the **number of tiles per axis is** 2^z and each tile's width/height is the world extent divided by that power of two:

```
tile_size_m(z) = 40 075 016.685 6 m / 2^{z}
```

So the **tile borders**—hence the bounding-box edges you see in `roads_geom_idx`—always lie on coordinates that are integer multiples of `tile_size_m(z)`, a power-of-two fraction of the whole world.

3.3.1 Concrete example

Zoom z	Tiles across	Tile width	X/Y edge positions (m)
0	1	40 075 016 m	−20 037 508 , **+20 037 508**
1	2	20 037 508 m	−20 037 508 , **0** , +20 037 508
2	4	10 018 754 m	−20 037 508 , −10 018 754 , **0** , +10 018 754 , +20 037 508
3	8	5 009 377 m	… every 5 009 377 m …

All those tile-edge coordinates—0 m, ±10 018 754 m, ±5 009 377 m, …—are **power-of-two divisors** of the full 40 075 016 m extent.

3.3.2 Why osm2pgsql cares

- When it builds a **GiST index** on your `geom` column, it packs bounding boxes that line up neatly with this grid, so PostGIS can reject most off-tile features with a single integer comparison.
- Tile renderers (Mapnik, Mapbox GL, etc.) request "tile X,Y at zoom Z" and know the exact EPSG 3857 range to query because it's just *`tile_size_m(z)` · `X` … `tile_size_m(z)` · `(X+1)`*.

This lets the world-wide Web-Mercator tiling stack stay simple, cacheable, and lightning fast.

4 Diff workflow

4.1 Init

4.1.1 Run once after the fresh import

```
PGPASSWORD=geo osm2pgsql-replication init \
  -H 127.0.0.1 -P 5432 -U postgres -d osm \
  --osm-file "$HOME/geo/data/cyprus-latest.osm.pbf"
```

4.2 Run regularly

> **Notabene:** After each append, update materialised metrics if needed (example: re-compute road length only for cells that changed).

```
PGPASSWORD=geo osm2pgsql-replication update \
  -H 127.0.0.1 -P 5432 -U postgres -d osm \
  -- \
  --output=flex \
  --style="$HOME/geo/data/cyprus.lua" \
  --flat-nodes="$HOME/geo/flat-nodes.bin" \
  --number-processes=4 \
  --cache=2000
```

Sample output:

```
papavova@Thinker:/mnt/c/Users/papa.vova$ PGPASSWORD=geo osm2pgsql-replication update \
>   -H 127.0.0.1 -P 5432 -U postgres -d osm \
>   -- \
>   --output=flex \
>   --style="$HOME/geo/data/cyprus.lua" \
>   --flat-nodes="$HOME/geo/flat-nodes.bin" \
>   --number-processes=4 \
>   --cache=2000
2025-05-29 11:49:30 [INFO]: Using replication service 'https://download.geofabrik.de/europe/cy
2025-05-29 11:49:33  osm2pgsql version 1.11.0
2025-05-29 11:49:33  Database version: 15.8 (Debian 15.8-1.pgdg110+1)
2025-05-29 11:49:33  PostGIS version: 3.4
2025-05-29 11:49:33  Loading properties from table '"public"."osm2pgsql_properties"'.
2025-05-29 11:49:33  Using flat node file '/home/papavova/geo/flat-nodes.bin' (same as on impo
2025-05-29 11:49:33  Using style file '/home/papavova/geo/data/cyprus.lua' (same as on import)
2025-05-29 11:49:35  Reading input files done in 1s.
2025-05-29 11:49:35    Processed 7172 nodes in 0s - 7k/s
2025-05-29 11:49:35    Processed 1747 ways in 1s - 2k/s
2025-05-29 11:49:35    Processed 51 relations in 0s - 51/s
2025-05-29 11:49:35  Going over 829 pending ways (using 4 threads)
Left to process: 0.....
2025-05-29 11:49:37  Processing 829 pending ways took 2s at a rate of 414.50/s
2025-05-29 11:49:37  Going over 242 pending relations (using 4 threads)
Left to process: 0...
2025-05-29 11:49:38  Processing 242 pending relations took 1s at a rate of 242.00/s
2025-05-29 11:49:38  Skipping stage 1c (no marked ways).
2025-05-29 11:49:38  No marked ways (Skipping stage 2).
2025-05-29 11:49:38  Done postprocessing on table 'planet_osm_nodes' in 0s
2025-05-29 11:49:38  Done postprocessing on table 'planet_osm_ways' in 0s
2025-05-29 11:49:38  Done postprocessing on table 'planet_osm_rels' in 0s
2025-05-29 11:49:38  All postprocessing on table 'roads' done in 0s.
2025-05-29 11:49:38  osm2pgsql took 5s overall.
2025-05-29 11:49:39 [INFO]: Data imported until 2025-05-28T20:21:00Z. Backlog remaining: 12 hou
```

```
papavova@Thinker:/mnt/c/Users/papa.vova$
```

(Geofabrik regional feed lags behind the global planet feed, that's why there's still backlog.)

4.3 Check the replication state

4.3.1 In general

```
SELECT property, value
FROM   osm2pgsql_properties
WHERE  property LIKE 'replication_%';
```

Should look like:

```
postgres@127:osm> SELECT property, value FROM osm2pgsql_properties WHERE property LIKE 'replica
+-----------------------------+-------------------------------------------------------+
| property                    | value                                                 |
|-----------------------------+-------------------------------------------------------/
| replication_base_url        | https://download.geofabrik.de/europe/cyprus-updates |
| replication_sequence_number | 4432                                                  |
| replication_timestamp       | 2025-05-24T20:21:15Z                                  |
+-----------------------------+-------------------------------------------------------+
SELECT 3
Time: 0.022s
postgres@127:osm>
```

`replication_timestamp` and `replication_sequence_number` should be advancing.

4.3.2 Where we are at any moment

```
SELECT
  property as timestamp,
  value::timestamptz AS utc_time
FROM osm2pgsql_properties
WHERE property LIKE '%timestamp%';
```

Should look like:

```
postgres@127:osm> SELECT
   property as timestamp,
     value::timestamptz AS utc_time
     FROM osm2pgsql_properties
     WHERE property LIKE '%timestamp%';
+----------------------+-----------------------+
| timestamp            | utc_time              |
|----------------------+-----------------------/
| import_timestamp     | 2025-05-24 18:24:09+00 |
| current_timestamp    | 2025-05-28 19:30:00+00 |
| replication_timestamp | 2025-05-28 20:21:00+00 |
+----------------------+-----------------------+
SELECT 3
```

```
Time: 0.009s
postgres@127:osm>
```

4.4 Setup cron or systemd — not done

TODO, does not seem that important

5 pg_stat stuff for observability

5.1 One-off setup

```
# Stop the container, if running
docker stop pg

# Start container with the -c flag for stat_statements
docker run -d --name pg \
  -e POSTGRES_PASSWORD=geo \
  -v $HOME/geo/pgdata:/var/lib/postgresql/data \
  -v $HOME/geo/data:/imports \
  -v $HOME/geo:/geo \
  -p 5432:5432 \
  postgis/postgis:15-3.4 \
  -c shared_preload_libraries=pg_stat_statements

# Create & verify the extension
PGPASSWORD=geo psql -h 127.0.0.1 -U postgres -d osm -c "CREATE EXTENSION IF NOT EXISTS pg_stat
PGPASSWORD=geo psql -h 127.0.0.1 -U postgres -d osm -c "SELECT pg_stat_statements_reset();"
```

5.2 Profiling example

- Reset the counters:

```
PGPASSWORD=geo psql -h 127.0.0.1 -U postgres -d osm \
  -c "SELECT pg_stat_statements_reset();"
```

- Run stuff, e.g. the update procedure
- Check the top statements:

```
PGPASSWORD=geo psql -h 127.0.0.1 -U postgres -d osm -c "
SELECT query,
       calls,

       -- total seconds, 2 dp
       round(((total_plan_time + total_exec_time) / 1000)::numeric, 2)
         AS total_s,

       -- average ms per call, 2 dp
       round(((mean_plan_time  + mean_exec_time )      )::numeric, 2)
         AS ms_per_call,
```

```
       rows
FROM   pg_stat_statements
ORDER  BY (total_plan_time + total_exec_time) DESC
LIMIT  8;
" > top_statements_total.log

PGPASSWORD=geo psql -h 127.0.0.1 -U postgres -d osm -c "
SELECT
    query,
    calls,

    -- cumulative time spent just parsing & planning (seconds, 2 dp)
    round((total_plan_time  / 1000)::numeric, 2) AS plan_s,

    -- cumulative time spent actually running the statement (seconds, 2 dp)
    round((total_exec_time  / 1000)::numeric, 2) AS exec_s,

    -- average per call (milliseconds, 2 dp)
    round(mean_plan_time::numeric, 2)  AS mean_plan_ms,
    round(mean_exec_time::numeric, 2)  AS mean_exec_ms,

    rows
FROM   pg_stat_statements
ORDER  BY total_exec_time DESC          -- heaviest execution cost first
LIMIT  8;
" > top_statements_plan-exec.log
```

6 Check-list to show in the interview

Demonstrable item	How you'll show it
Custom import config	Open `cyprus.lua`, point to table + tag logic
Selective tags saved	`\d roads` in psql (only 4 columns)
Diff update working	Show `osm_state` timestamp advancing & row counts growing
Index / plan proof	`EXPLAIN ANALYZE` before & after SP-GIST
Trade-off commentary	Why flex-config beats style file; why 3857 for length