Vorbereitung

Neue PostGIS Datenbank mit Docker run Befehl angelegt und in QGIS eingebunden:

docker run

```
docker run --name col_dd_db_02_analytics -d -e
POSTGRES_PASSWORD=postgres -p 5403:5432 -v ~/col-
dd/col_dd_db_02_analytics:/var/lib/postgresql/data mdillon/postgis
```

CREATE SCHEMA data;

CREATE EXTENSION postgis;

CREATE EXTENSION hstore;

Schema „data“ bereits angelegt mit folgenden Tabellen

- **edits**
- **buildings**: (gleich als Centroide / Punktgeometrien)

CREATE SCHEMA grids;

- **districs**: Open Data Dresden → WFS Dresden Stadtteile import in DB per Drag and Drop
- **inspire_1km:** INSPIRE 1x1km Gitter → Drag and Drop in DB ziehen

Layer in QGIS visualisiert zeigen (edits, buildings, districts und inspire_1km)

Datenimport

```
CREATE SCHEMA results;

CREATE SCHEMA mapping;


CREATE TABLE mapping.mapping_feature_category (
      id SERIAL PRIMARY KEY,
      feature_de VARCHAR,
      feature_en VARCHAR,
      category_de VARCHAR,
      category_en VARCHAR
);
```

Datenzeilen für Zuweisung von Gebäudemerkmal auf Kategorie importieren

(Beispielzeile):

```
INSERT INTO mapping.mapping_feature_category (feature_de, feature_en,
category_de, category_en) VALUES ('Name des Gebäudes', 'location_name',
'Standort', 'location');



SELECT * FROM data_prep.mapping_feature_category;
```

Spatial Index

Spatial Index erzeugen mit GIST für die Tabellen: buildings und edits

```
CREATE INDEX edits_geom_idx
  ON data.edits
  USING GIST (geom);


CREATE INDEX buildings_geom_idx
  ON data.buildings
  USING GIST (geom);
```

Daten sichten und prüfen

Ersten zehn Zeilen einer Tabelle sichten

```
SELECT * FROM data.edits ORDER BY log_timestamp DESC LIMIT 10;

SELECT * FROM data.edits ORDER BY log_timestamp ASC LIMIT 10;
```

Koordinatenreferenzsystem der Geometrien prüfen:

```
SELECT ST_Srid(geom) AS "SRID", COUNT(*) AS "Anzahl"
FROM data.edits
GROUP BY ST_Srid(geom)
ORDER BY "Anzahl" DESC;
```

Geometrietypen prüfen

```
SELECT ST_GeometryType(geom) AS "GeometryType", COUNT(*) AS "Anzahl"
FROM data.buildings
GROUP BY ST_GeometryType(geom)
ORDER BY "Anzahl" DESC;
```

Validität prüfen

```
SELECT ST_IsValid(geom) AS "IsValid", COUNT(*) AS "Anzahl"
FROM grids.districts
GROUP BY ST_Isvalid(geom)
ORDER BY "Anzahl" DESC;
```

Ist die Geometrie simple und valide gemäß OGC? zB keine Self-Intersection

```
SELECT ST_IsSimple(geom) AS "IsSimple", COUNT(*) AS "Anzahl"
FROM grids.inspire_1km
GROUP BY ST_IsSimple(geom)
ORDER BY "Anzahl" DESC;
```

Ausgabe von Speicherplatz pro Tabelle und Counts

```
SELECT
      t.schemaname AS schema_name,
      t.tablename AS table_name,
      pg_total_relation_size(format('%I.%I', t.schemaname,
t.tablename)) AS total_size_bytes,
      pg_table_size(format('%I.%I', t.schemaname, t.tablename)) AS
table_size_bytes,
      (SELECT n_live_tup FROM pg_stat_user_tables WHERE relname =
t.tablename) AS row_count
FROM
      pg_tables t
WHERE
      t.schemaname = 'data'
UNION ALL
SELECT
      t.schemaname AS schema_name,
      t.tablename AS table_name,
      pg_total_relation_size(format('%I.%I', t.schemaname,
t.tablename)) AS total_size_bytes,
      pg_table_size(format('%I.%I', t.schemaname, t.tablename)) AS
table_size_bytes,
      (SELECT n_live_tup FROM pg_stat_user_tables WHERE relname =
t.tablename) AS row_count
FROM
      pg_tables t
WHERE
      t.schemaname = 'grids'
UNION ALL
SELECT
      t.schemaname AS schema_name,
      t.tablename AS table_name,
      pg_total_relation_size(format('%I.%I', t.schemaname,
t.tablename)) AS total_size_bytes,
      pg_table_size(format('%I.%I', t.schemaname, t.tablename)) AS
table_size_bytes,
      (SELECT n_live_tup FROM pg_stat_user_tables WHERE relname =
t.tablename) AS row_count
FROM
      pg_tables t
WHERE
      t.schemaname = 'mapping';
```

Datenaufbereitung Edits → einzelne Feature

Aus einem Edit einzelne Features extrahieren in neue Tabelle

```
CREATE TABLE data.mapped_features
AS
WITH data1_extracted AS (
    SELECT
        id as "edit_id",
        json_each_text.key AS key1,
        json_each_text.value AS value1,
        ROW_NUMBER() OVER (PARTITION BY id ORDER BY
json_each_text.key) AS rn
    FROM
        data.edits,
        LATERAL json_each_text(forward_patch::json)
),
data2_extracted AS (
    SELECT
        id as "edit_id",
        json_each_text.key AS key2,
        json_each_text.value AS value2,
        ROW_NUMBER() OVER (PARTITION BY id ORDER BY
json_each_text.key) AS rn
    FROM
        data.edits,
        LATERAL json_each_text(reverse_patch::json)
)
SELECT
    ROW_NUMBER() OVER () AS "id",
     d1."edit_id",
    d1.key1 AS "forward_key",
    d1.value1 AS "forward_value",
    d2.key2 AS "reverse_key",
    d2.value2 AS "reverse_value"
FROM
    data1_extracted d1
JOIN
    data2_extracted d2
    ON d1."edit_id" = d2."edit_id" AND d1.rn = d2.rn
ORDER BY
    d1."edit_id", d1.rn;
```

Nun den Edit Type bestimmen, ob das Gebäudemerkmal hinzugefügt (added), geändert (modified) oder entfernt (removed) wurde.

```
ALTER TABLE data.mapped_features
ADD COLUMN edit_type VARCHAR;



UPDATE data.mapped_features
SET "edit_type" = CASE
    WHEN "forward_value" NOT LIKE '%{%}%' AND "reverse_value" IS
NULL AND "forward_value" IS NOT NULL THEN 'added'
    WHEN "reverse_value" NOT LIKE '%{%}%' AND "reverse_value" IS NOT
NULL AND "forward_value" IS NULL THEN 'removed'
    WHEN "reverse_value" NOT LIKE '%{%}%' AND "forward_value" NOT
LIKE '%{%}%' AND "reverse_value" IS NOT NULL AND "forward_value" IS
NOT NULL THEN 'modified'

     WHEN "reverse_value" LIKE '{%}' AND "forward_value" LIKE '{%}'
AND (SELECT COUNT(*) FROM jsonb_object_keys("forward_value"::jsonb))
> (SELECT COUNT(*) FROM jsonb_object_keys("reverse_value"::jsonb))
THEN 'added'
     WHEN "reverse_value" IS NULL AND "forward_value" LIKE '{%}'
THEN 'added'
     WHEN "reverse_value" LIKE '{%}' AND "forward_value" LIKE '{%}'
AND (SELECT COUNT(*) FROM jsonb_object_keys("forward_value"::jsonb))
< (SELECT COUNT(*) FROM jsonb_object_keys("reverse_value"::jsonb))
THEN 'removed'
     WHEN "reverse_value" LIKE '{%}' AND "forward_value" IS NULL
THEN 'removed'
     WHEN "reverse_value" LIKE '{%}' AND "forward_value" LIKE '{%}'
AND (SELECT COUNT(*) FROM jsonb_object_keys("forward_value"::jsonb))
= (SELECT COUNT(*) FROM jsonb_object_keys("reverse_value"::jsonb))
THEN 'modified'

     WHEN "reverse_value" LIKE '[%]' AND "forward_value" LIKE '[%]'
AND (SELECT COUNT(*) FROM
jsonb_array_length("forward_value"::jsonb)) > (SELECT COUNT(*) FROM
jsonb_array_length("reverse_value"::jsonb)) THEN 'added'
     WHEN "reverse_value" IS NULL AND "forward_value" LIKE '[%]'
THEN 'added'
     WHEN "reverse_value" LIKE '[%]' AND "forward_value" LIKE '[%]'
AND (SELECT COUNT(*) FROM
jsonb_array_length("forward_value"::jsonb)) < (SELECT COUNT(*) FROM
jsonb_array_length("reverse_value"::jsonb)) THEN 'removed'
     WHEN "reverse_value" LIKE '[%]' AND "forward_value" IS NULL
THEN 'removed'
     WHEN "reverse_value" LIKE '[%]' AND "forward_value" LIKE '[%]'
AND (SELECT COUNT(*) FROM
```

```
jsonb_array_length("forward_value"::jsonb)) = (SELECT COUNT(*) FROM
jsonb_array_length("reverse_value"::jsonb)) THEN 'modified'
    ELSE "edit_type" -- retain the original value if no conditions
match
END;
```

Kontrolle mit Group By und Count

```
SELECT "edit_type", COUNT(*) AS "count" FROM data.mapped_features
GROUP BY "edit_type" ORDER BY "count" DESC;
```

```
SELECT * FROM data.mapped_features LIMIT 5;
```

Daten explorieren

COUNT Gebäude in Stadtteil Dresden Altstadt, mit WITH Statement

```
WITH district_altstadt AS (
      SELECT * FROM grids.districts WHERE bez = 'Innere Altstadt'
      )
SELECT COUNT (blds.*)
FROM data.buildings blds, district_altstadt dstr
WHERE ST_Intersects(blds.geom, dstr.geom);
```

Anz. Gebäudemerkmale nach Monat mit GROUP BY

```
SELECT DATE_TRUNC('month', log_timestamp) AS "Monat",
      COUNT(*) AS "Anzahl"
FROM data.edits
GROUP BY "Monat"
ORDER BY "Monat";
```

Zähle alle hinzugefügten Gebäudemerkmale nach Kategorie:

```
WITH mapped_features_with_category AS (
      SELECT ft.*, ct.category_de
      FROM data.mapped_features ft, mapping.mapping_feature_category
ct
      WHERE ft.forward_key = ct.feature_en
)
SELECT mft.category_de,
      COUNT(*) AS "Anzahl"
FROM mapped_features_with_category mft
WHERE edit_type='added'
GROUP BY mft.category_de
ORDER BY "Anzahl" DESC;
```

Anzahl hinzugefügter Gebäudemerkmale pro Anzahl Gebäude innerhalb Rasterzelle

Karten: dauert 1min15sec

- COUNT je Gitterzelle mit Edits
- COUNT je Stadtteil nach Anz Gebäude
- COUNT Anz. added features pro 100 Gebäude
    - Mit CASE für Division by Zero

```
CREATE TABLE results."number_added_features_by_number_buildings_within_gridcell" AS
WITH added_features AS (
	SELECT f.*,
		ST_Transform(e.geom, 25833) as "centroid",
		DATE_TRUNC ('day', e.log_timestamp) AS "date",
		m.category_de AS "category_de"
	FROM data.mapped_features f,
	data.edits e,
	mapping.mapping_feature_category m
	WHERE e.id = f.edit_id AND f.edit_type='added' AND f.forward_key=m.feature_en
	ORDER BY m.category_de DESC
	),
	grid AS (
			SELECT id,
				geom
			 FROM grids.inspire_1km),
	building_centroids AS (
			SELECT fid,
					"OBJEKT_ID",
					ST_Transform(geom, 25833) AS "geometry"
			FROM data.buildings),
	grid_joined_with_added_features AS (
					SELECT
						g.*,
						COUNT(p1.*) AS cnt_added_features
					FROM
						grid g
					LEFT JOIN
						added_features p1
					ON
						ST_Contains(g.geom, p1.centroid)
					GROUP BY
						g.id, g.geom
			),
	grid_joined_with_building_centroids AS (
					SELECT
						g.*,
						COUNT(p2.*) AS cnt_building_centroids
					FROM
						grid g
					LEFT JOIN
						building_centroids p2
					ON
						ST_Contains(g.geom, p2.geometry)
					GROUP BY
						g.id, g.geom
	)
SELECT
    g.*,
    COALESCE(pc1.cnt_added_features, 0) AS cnt_added_features,
    COALESCE(pc2.cnt_building_centroids, 0) AS cnt_building_centroids,
      CASE
       WHEN COALESCE(pc2.cnt_building_centroids, 0) = 0 THEN NULL  -- Avoid
division by zero
```

```
        ELSE COALESCE(100 * pc1.cnt_added_features, 0)::numeric /
COALESCE(pc2.cnt_building_centroids, 0)::numeric
    END AS ratio_added_ft_by_100_blds
FROM
    grid g
LEFT JOIN
    grid_joined_with_added_features pc1 ON g.id = pc1.id
LEFT JOIN
    grid_joined_with_building_centroids pc2 ON g.id = pc2.id;
```

Wie viele Edits pro Tag?

Edits nach Tag mit LEFT JOIN und COALESCE

Nutzung von series für Datumsstrahl

```
SELECT to_char(t.day::date, 'YYYY-MM-DD') AS "date",
     to_char(t.day::date, 'DD.MM.YYYY') AS "label",
     COALESCE(requested_data.count,0) AS "value"
FROM generate_series(timestamp '2023-03-06'
                     , timestamp '2023-06-01'
                     , interval  '1 day') AS t(day)
LEFT JOIN
     (SELECT COUNT(*) AS "count", DATE_TRUNC ('day', log_timestamp)
AS "date"
     FROM data.edits
     GROUP BY DATE_TRUNC ('day', log_timestamp)) AS requested_data
ON t.day::date=requested_data.date;
```

Konvexe Hüllen

Erzeugen von konvexen Hüllen über alle Edits eines Users

```
--SELECT some tables a priori for further usage
CREATE TABLE results.convex_hull_for_all_edits_of_each_user AS
WITH src_edits AS (
      SELECT *
      FROM data.edits),
      src_edits_groupby_count_user AS (
      SELECT DISTINCT user_id,
            COUNT(*) AS "number_edits"
            FROM src_edits
            GROUP BY user_id
      )
SELECT joined.*
FROM (
        SELECT DISTINCT user_id
        FROM src_edits
    ) src_edits_unique_users
    JOIN LATERAL (
        SELECT src_edits.user_id, ST_ConvexHull(ST_Union(geom)) as
geom, src_edits_groupby_count_user.number_edits
        FROM src_edits, src_edits_groupby_count_user
        WHERE src_edits.user_id = src_edits_unique_users.user_id
          AND src_edits.user_id =
src_edits_groupby_count_user.user_id
          GROUP BY src_edits.user_id,
src_edits_groupby_count_user.number_edits
    ) joined ON true;
```

In QGIS visualisieren mit

- **Füllfarbe: dunkelblau**
- **Transparenz: 70%**
- **Objekt: multiplizieren**

Dynamische Pivot-Tabelle mit PostGIS: eine Herausforderung

Anzahl added features nach Gebäudemerkmal und Tag in einer großen Tabelle, jeweils eine Tabelle für „added“, „modified“ und „removed“ Gebäudemerkmale.

Problem: PostGIS kann keine dynamischen Pivot-tabellen (hier ist Python mit Pandas und Pivot-Tabelle im Vorteil). PostGIS muss vorher wissen, wie viele Spalten die Zieltabelle aufweisen wird. Es gibt eine Lösung mit einem COMPOSITE TYPE / ENUMERATION, oder man schreibt alle Spalten statisch in die SQL-Abfrage und nutzt die Erweiterung „tablefunc“ mit crosstab und pivot-table.

Wir benötigen tablefunc Extension für crosstab function

```
CREATE EXTENSION tablefunc;
```

An die mapped features nun fest das Datum aus dem Edit speichern

```
CREATE TABLE data.mapped_features_with_date AS SELECT f.*,
     e.log_timestamp,
     DATE_TRUNC ('day', e.log_timestamp) AS "date"
FROM data."mapped_features" f,
data.edits e
WHERE e.id = f.edit_id
```

```
SELECT * FROM data.mapped_features_with_date LIMIT 10;
```

Nun nach edity-type, Gebäudemerkmal und Datum gruppierenen, mit jeweils der Anzahl.

(nennt sich „fact table“ im sog. Star-Schema eines Datenwürfels)

```
CREATE TABLE data.features_grouped_by_key_date_edittype
AS SELECT forward_key, edit_type, date, COUNT(*)
FROM data.mapped_features_with_date
GROUP BY forward_key, edit_type, date
ORDER BY date, forward_key, edit_type;

SELECT * FROM data.features_grouped_by_key_date_edittype LIMIT 10;
```

Und nun die Pivot Tabelle aufspannen. Dabei diejenigen Zellen mit Nullen auffüllen, wo keine Daten / Anzahl vorliegen.

```
WITH crosstab_number_features_by_day_but_missing_dates AS (
	SELECT *
	FROM crosstab(
		$$
		SELECT
			date,
			forward_key,
			count
		FROM data.features_grouped_by_key_date_edittype
		WHERE edit_type = 'modified'
		ORDER BY date

		$$,
		$$ SELECT DISTINCT forward_key FROM
data.features_grouped_by_key_date_edittype ORDER BY forward_key $$  -- Get distinct
keys for column names
	) AS pivot_table(
		date DATE,  -- Row identifier
		architectural_style INT,
		architectural_style_source INT,
		basement_percentage INT,
		basement_type INT,
		basement_use INT,
		basement_use_source INT,
		building_attachment_form INT,
		building_owner INT,
		building_owner_source INT,
		building_status INT,
		building_status_source INT,
		community_activities INT,
		community_activities_always INT,
		community_activities_current INT,
		community_expected_planning_application_total INT,
		community_local_significance_total INT,
		community_public_ownership INT,
		community_type_worth_keeping_total INT,
		construction_core_material INT,
		construction_roof_covering INT,
		construction_system_type INT,
		construction_system_type_source INT,
		date_link INT,
		date_lower INT,
		date_source INT,
		date_upper INT,
		date_year INT,
		designers INT,
		designers_source_link INT,
		developer_name INT,
		developer_type INT,
		dynamics_has_demolished_buildings INT,
		facade_year INT,
		ground_storey_use INT,
		ground_storey_use_source INT,
		has_extension INT,
		is_domestic INT,
		landowner INT,
		last_renovation INT,
		last_renovation_source INT,
		lead_designer_type INT,
		location_latitude INT,
		location_line_two INT,
		location_longitude INT,
		location_name INT,
```

```
            location_number INT,
            location_postcode INT,
            location_street INT,
            location_town INT,
            size_floor_area_ground INT,
            size_height_apex INT,
            size_roof_shape INT,
            size_roof_shape_source INT,
            size_storeys_attic INT,
            size_storeys_basement INT,
            size_storeys_core INT,
            size_width_frontage INT,
            upper_storeys_use INT,
            upper_storeys_use_source INT,
            use_building_current INT,
            use_building_current_text INT,
            use_building_origin INT,
            use_building_origin_text INT,
            use_number_businesses INT,
            use_number_residential_units INT
      )
)
SELECT to_char(t.day::date, 'YYYY-MM-DD') AS "date",
      to_char(t.day::date, 'DD.MM.YYYY') AS "label",
      COALESCE(requested_data.architectural_style,0) AS "architectural_style",
      COALESCE(requested_data.architectural_style_source,0) AS
"architectural_style_source",
      COALESCE(requested_data.basement_percentage,0) AS "basement_percentage",
      COALESCE(requested_data.basement_type,0) AS "basement_type",
      COALESCE(requested_data.basement_use,0) AS "basement_use",
      COALESCE(requested_data.basement_use_source,0) AS "basement_use_source",
      COALESCE(requested_data.building_attachment_form,0) AS
"building_attachment_form",
      COALESCE(requested_data.building_owner,0) AS "building_owner",
      COALESCE(requested_data.building_owner_source,0) AS "building_owner_source",
      COALESCE(requested_data.building_status,0) AS "building_status",
      COALESCE(requested_data.building_status_source,0) AS
"building_status_source",
      COALESCE(requested_data.community_activities,0) AS "community_activities",
      COALESCE(requested_data.community_activities_always,0) AS
"community_activities_always",
      COALESCE(requested_data.community_activities_current,0) AS
"community_activities_current",
      COALESCE(requested_data.community_expected_planning_application_total,0) AS
"community_expected_planning_application_total",
      COALESCE(requested_data.community_local_significance_total,0) AS
"community_local_significance_total",
      COALESCE(requested_data.community_public_ownership,0) AS
"community_public_ownership",
      COALESCE(requested_data.community_type_worth_keeping_total,0) AS
"community_type_worth_keeping_total",
      COALESCE(requested_data.construction_core_material,0) AS
"construction_core_material",
      COALESCE(requested_data.construction_roof_covering,0) AS
"construction_roof_covering",
      COALESCE(requested_data.construction_system_type,0) AS
"construction_system_type",
      COALESCE(requested_data.construction_system_type_source,0) AS
"construction_system_type_source",
      COALESCE(requested_data.date_link,0) AS "date_link",
      COALESCE(requested_data.date_lower,0) AS "date_lower",
      COALESCE(requested_data.date_source,0) AS "date_source",
      COALESCE(requested_data.date_upper,0) AS "date_upper",
      COALESCE(requested_data.date_year,0) AS "date_year",
```

```
       COALESCE(requested_data.designers,0) AS "designers",
       COALESCE(requested_data.designers_source_link,0) AS "designers_source_link",
       COALESCE(requested_data.developer_name,0) AS "developer_name",
       COALESCE(requested_data.developer_type,0) AS "developer_type",
       COALESCE(requested_data.dynamics_has_demolished_buildings,0) AS
"dynamics_has_demolished_buildings",
       COALESCE(requested_data.facade_year,0) AS "facade_year",
       COALESCE(requested_data.ground_storey_use,0) AS "ground_storey_use",
       COALESCE(requested_data.ground_storey_use_source,0) AS
"ground_storey_use_source",
       COALESCE(requested_data.has_extension,0) AS "has_extension",
       COALESCE(requested_data.is_domestic,0) AS "is_domestic",
       COALESCE(requested_data.landowner,0) AS "landowner",
       COALESCE(requested_data.last_renovation,0) AS "last_renovation",
       COALESCE(requested_data.last_renovation_source,0) AS
"last_renovation_source",
       COALESCE(requested_data.lead_designer_type,0) AS "lead_designer_type",
       COALESCE(requested_data.location_latitude,0) AS "location_latitude",
       COALESCE(requested_data.location_line_two,0) AS "location_line_two",
       COALESCE(requested_data.location_longitude,0) AS "location_longitude",
       COALESCE(requested_data.location_name,0) AS "location_name",
       COALESCE(requested_data.location_number,0) AS "location_number",
       COALESCE(requested_data.location_postcode,0) AS "location_postcode",
       COALESCE(requested_data.location_street,0) AS "location_street",
       COALESCE(requested_data.location_town,0) AS "location_town",
       COALESCE(requested_data.size_floor_area_ground,0) AS
"size_floor_area_ground",
       COALESCE(requested_data.size_height_apex,0) AS "size_height_apex",
       COALESCE(requested_data.size_roof_shape,0) AS "size_roof_shape",
       COALESCE(requested_data.size_roof_shape_source,0) AS
"size_roof_shape_source",
       COALESCE(requested_data.size_storeys_attic,0) AS "size_storeys_attic",
       COALESCE(requested_data.size_storeys_basement,0) AS "size_storeys_basement",
       COALESCE(requested_data.size_storeys_core,0) AS "size_storeys_core",
       COALESCE(requested_data.size_width_frontage,0) AS "size_width_frontage",
       COALESCE(requested_data.upper_storeys_use,0) AS "upper_storeys_use",
       COALESCE(requested_data.upper_storeys_use_source,0) AS
"upper_storeys_use_source",
       COALESCE(requested_data.use_building_current,0) AS "use_building_current",
       COALESCE(requested_data.use_building_current_text,0) AS
"use_building_current_text",
       COALESCE(requested_data.use_building_origin,0) AS "use_building_origin",
       COALESCE(requested_data.use_building_origin_text,0) AS
"use_building_origin_text",
       COALESCE(requested_data.use_number_businesses,0) AS "use_number_businesses",
       COALESCE(requested_data.use_number_residential_units,0) AS
"use_number_residential_units"
FROM generate_series(timestamp '2023-03-06'
                           , timestamp '2023-10-01'
                           , interval  '1 day') AS t(day)
LEFT JOIN
       (SELECT * FROM crosstab_number_features_by_day_but_missing_dates)
       AS requested_data
ON t.day::date=requested_data.date;
```