NEXTWIND

Development of a PostGIS-Based Geospatial Infrastructure for Wind Farm Planning

Nils Hendrischke

Internship Report

December 2025

Supervisor at University of Potsdam: Gerold Zeilinger
Supervisor at Nextwind Management GmbH: Behrooz Rostami

I hereby declare that I have written this internship report independently and without any unauthorized assistance. All sources and materials used have been properly acknowledged. Artificial intelligence tools were used exclusively for language refinement and formatting purposes. The content, analysis, and conclusions presented in this report are entirely my own work.

Berlin, 11.12.2025 Hendrischke

Date Signature

Abstract

As NeXtwind continues to grow rapidly, the company faces several challenges in its technical infrastructure and data management. In the field of geospatial data, our goal is to provide colleagues with a synchronized, user-friendly system for accessing and managing spatial information.

Previously, wind farm planning relied heavily on printed maps and shapefiles. Data was stored in shared SharePoint folders, and simultaneous file access frequently led to errors and inconsistencies. The goal of this report is to design and implement a PostGIS-based database that can be accessed through both common SQL query tools and desktop GIS, while also providing a guided introduction on how the system works and how it can be used. The report explains the modeling and implementation processes, outlines possible technical applications, and discusses how the database structure aligns with the company's practical requirements.

The presented data model and database together form a digital twin of NeXtWind's wind and energy parks, representing elements such as planned and existing wind turbines, cable routes, relevant land parcels, and administrative boundaries. The model is designed to be scalable, with clearly defined attributes to support flexible analysis, planning, and reporting. Finally, the report demonstrates examples of automation through database triggers and outlines potential directions for future development and system evolution.

Contents

Chapter 1

Introduction

NeXtWind is a renewable energy company founded in 2020, specializing in the repowering of onshore wind farms. Its core strategy involves replacing outdated wind turbines with modern, higher-capacity models to significantly increase energy output on existing sites. Developing a wind farm involves multiple interdependent steps: identifying suitable locations based on wind and grid conditions, negotiating with landowners and municipalities, conducting environmental assessments, obtaining regulatory approval (e.g. under the Federal Immission Control Act (Bundes-Immissionsschutzgesetz, BImSchG)), and managing detailed design, construction, and commissioning. The company also plans to expand into solar energy, battery storage, and the construction of its own substations to play a more integrated and profitable role in Germany's energy sector.

As NeXtWind continues to expand its portfolio, the need for a coherent digital infrastructure becomes increasingly important. A central part of this digital transformation is the creation of a digital twin of the company's assets and planning data — a system that reflects the current and planned state of each energy park in real time. Visualization of this information is essential for efficient decision-making across teams.

Within this digital twin, the geospatial component plays a key role. Being able to visualize and query the current park layout at the push of a button is as crucial as checking the current Stage-Gate status of a project — a system that was recently introduced to track the development phases of the energy parks. Tasks such as potential analysis, park layout optimization, environmental planning, or the calculation of land lease payments all depend on an accurate and flexible representation of real-world features such as land parcels, turbines, access roads, and grid connections.

To establish a system for categorizing all relevant features and depict their relationships we introduce the concept of Entity-Relationship Models (ERM) [2, 7]. The ERM proposed in this report has been developed primarily from a project development and construction perspective, reflecting the author's position within that department. Nonetheless, it is designed to remain extensible so that, if the concept is pursued further, future adaptations can incorporate additional aspects of asset management, operations, or maintenance workflows as the company's digital infrastructure evolves.

While various database-like tools are already in use across the company — such as Airtable, Monday.com, or Excel — these systems are not designed to handle geospatial relationships or spatial analysis. They are valuable for workflow and metadata manage-

ment but lack the functionality to represent and process spatial geometries. For this reason, a dedicated geospatial database system is required. The proposed solution — based on PostgreSQL with the PostGIS extension — adds full geospatial capabilities to a robust relational database system. Its open architecture and API accessibility make it possible to connect or synchronize with other company systems, such as Airtable, in a future integration phase, provided that NeXtWind decides to adopt and further develop this solution in coordination with management and the IT team.

This report is structured as follows: Chapter 2 discusses the motivation for establishing a geospatial database at NeXtWind, outlining current challenges and the opportunities that arise from adopting a PostGIS-based solution. Chapter 3 presents the conceptual data model, defining the core entities and relationships required to represent the company's assets in a geospatial context. Chapter 4 describes the implementation of this model in PostgreSQL/PostGIS, including example queries and automation through triggers. The report concludes with Chapter 5, a discussion of potential next steps toward system expansion and integration within the company's digital infrastructure.

Chapter 2

Current Status and Motivation

At present, most geospatial work at NeXtWind is carried out manually and managed in a decentralized way. Each business unit (BU) maintains its own desktop GIS project files, which contain collections of visualization templates, and links to various geospatial data files. These files exist in multiple formats — including Shapefiles (e.g., SharePoint/WP_Altmark/09_Planning/03_QGIS/Standort_E-82.shp), CAD files (SharePoint/GIS/Projects/Kirchdorf_Ni/WorkingFiles/Kirchdorf_WEA.dxf), GeoPackages, or KML files — and often contain similar information, such as wind turbine locations (WEA – Windenergieanlagen). While such formats are convenient for local use and data exchange, they are not suitable for centralized storage. Combined with SharePoint synchronization, they frequently lead to duplicated datasets, conflicting versions, and lack true multi-user editing.

Recognizing these limitations, some datasets have already been migrated to QGIS Cloud (https://qgiscloud.com/), a data hosting and Web-GIS service. However, the QGIS Cloud database has several drawbacks: it offers limited configurability, is disproportionately expensive (approximately 20€ per month per GB), and restricts connections to a maximum of 20 users. Moreover, the current setup was established without a predefined data model or structural framework. It also lacks version control and change-tracking functionality, creating risks of unnoticed data loss and making it difficult to reconstruct previous data states.

The current, mainly decentralized approach makes cross-project collaboration difficult. When data from multiple projects must be combined, each individual file has to be located, verified, and merged manually. This process increases the risk that important datasets are overlooked, duplicated, or become inconsistent between projects. To avoid fragmentation and loss of oversight, a centralized geospatial infrastructure is required. The key principle is to establish a single source of truth — a unified database where all relevant spatial entities (e.g., wind turbines, parcels, infrastructure elements) are stored and managed consistently. The centralized storage does not prevent decentralized use. Subsets of the data can still be filtered, exported, or referenced in individual project files, but the underlying database ensures that the authoritative information remains consistent and complete. A relational database management system with geospatial extensions, like PostGIS, provides such a foundation. It combines structured storage, performance, and integration capabilities with mechanisms for multi-user access, data

security and data versioning if needed. File-based formats will still play a role for exchange and local work, but we suggest that the core planning data is better maintained in a database environment.

2.1 Relational Databases

Relational databases are a proven standard across many industries, including energy, logistics, finance, and urban planning. They are used to manage interdependent datasets where data integrity is critical. The same principles that enable management of millions of transactions or utility assets can be applied to renewable energy project development. A relational database structures data in tables with clearly defined relationships, avoiding redundancy and inconsistency. Consider the example of land parcels (Flurstücke) and landowners. A naive approach would store all parcel information in one table, with additional columns for owner details such as name, address, or age. If one owner holds 100 parcels, their information is repeated 100 times. This is inefficient and makes updates error-prone. In a relational model, owners and parcels are stored in separate tables. A third table defines the relationships between them. This "many-to-many relation" allows multiple owners per parcel and multiple parcels per owner without duplication. If an address changes, it only needs to be updated once. This principle of separating entities and linking them through relations ensures non-redundant storage and consistent and reliable data across projects [2, 1]. The same logic applies to other planning-relevant data, such as wind turbines and their technical specifications, grid connections, or environmental constraints.

2.2 Geospatial Relations

Traditionally, spatial checks (e.g., whether a planned turbine lies within a certain zoning area) are performed manually through GIS interfaces. This is time-consuming and prone to oversight. A database with geospatial capabilities allows such tasks to be automated and scaled. By extending structured query language (SQL) with spatial functions, it becomes possible to run geometric analyses directly within the database environment [1, 5]. This allows users to perform operations such as buffering, overlaying, and distance calculations on geometries. Furthermore, spatial joins can be used to relate datasets based on location — for instance, linking wind turbines to land-use categories or assigning parcels to municipalities. In addition, combining and filtering datasets through SQL queries allows for reproducible analyses. This shifts the workflow from manual checks towards standardized queries, enabling faster analysis, consistent results and better traceability.

Chapter 3

Data Model

To design a robust geospatial infrastructure, it is first necessary to define what information the system should represent and how different elements relate to each other. The data model forms the conceptual foundation for subsequent database and interface developments. The level of detail represented in a model is always a trade-off between completeness and maintainability. A more detailed model allows for higher analytical accuracy but increases the long-term effort for data maintenance and documentation. Conversely, a simplified model reduces complexity and upkeep but may generalize certain real-world aspects. For the current stage of work, the chosen model aims to capture only essential entities and relationships with a minimal level of detail, acknowledging that future refinement may be needed as workflows and requirements evolve.

The data model is structured into two groups: *Business Unit (BU) Data*, representing company-owned or project-related assets and *Authority Data*, representing external or official datasets from public authorities. The model was created using Draw.io (https://draw.io/), and the associated XML file can be found in the Appendix A.1 for future adaptation or extension. In the following sections, we first introduce the notation and structure of Entity–Relationship Models, then describe the content of both model groups, and conclude with an overview of attribute assignment.

3.1 Entity-Relationship Modeling

The Entity–Relationship Model (ERM) [2] formalizes the relevant entities—such as turbines, parcels, landowners—and the relationships between them. Each entity is represented as a rectangular box, contains a primary key, which is a unique identifier for distinguishing real-world features that can later be visualized in GIS, as well as a defined geometry type. The geometry types used in the entity definitions follow the Open Geospatial Consortium (OGC) Simple Feature specification [4]. The German entity name is also included for clarity. Arrow lines indicate specialization, meaning that the child entity inherits both geometry and primary key from its parent. The diamond-shaped symbols represent relationships. A relationship can either be one-to-one (1:1), one-to-many (1:n), or many-to-many (n:m) [7]. No totality constraints were introduced to preserve flexibility during data entry. The ERM only focuses on the non-spatial relationships between entities. Spatial relations (such as containment, overlap, distance)

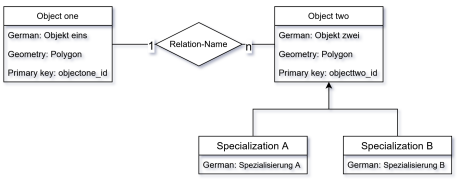


Figure 3.1: Example of ERM notation

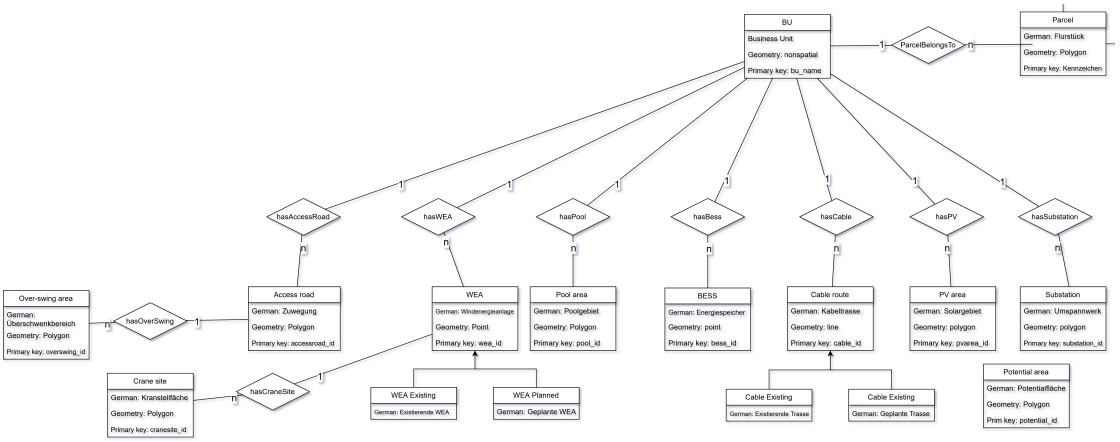


Figure 3.2: ERM of the Business Unit Data

are not stored explicitly and can be computed dynamically through queries with spatial joins. An example of the ERM notation is shown in Figure 3.1.

3.2 Business Unit Data

A Business Unit (BU) is an abstract entity defined by management to group and structure assets. Currently, around 35 BUs exist, with roughly 25 added within the last year. Data within the Business Unit group is mostly maintained by project developers and other internal staff. This group therefore represents the dynamic part of the data model: real-world features that frequently change as planning progresses. In our proposed Model (Figure 3.2), each BU can "own" or be associated with multiple asset types (entities), forming one-to-many relationships with Wind turbines (WEA), Battery energy storage systems (BESS), Cable routes, Photovoltaic areas (PV), Pool areas, Substations and Access roads. Some of these entities hold sub-entities: Access roads can have multiple over-swing areas. A WEA can have a (temporal or permanent) crane site. For both WEAs and cable routes entities, the model distinguishes between existing and planned, as their attributes and data sources typically differ throughout the planning process. Additionally, potential areas that are often not yet assigned to a specific BU are included for completeness. Finally, each BU can be linked to several parcels, connecting the internal and authority domains.

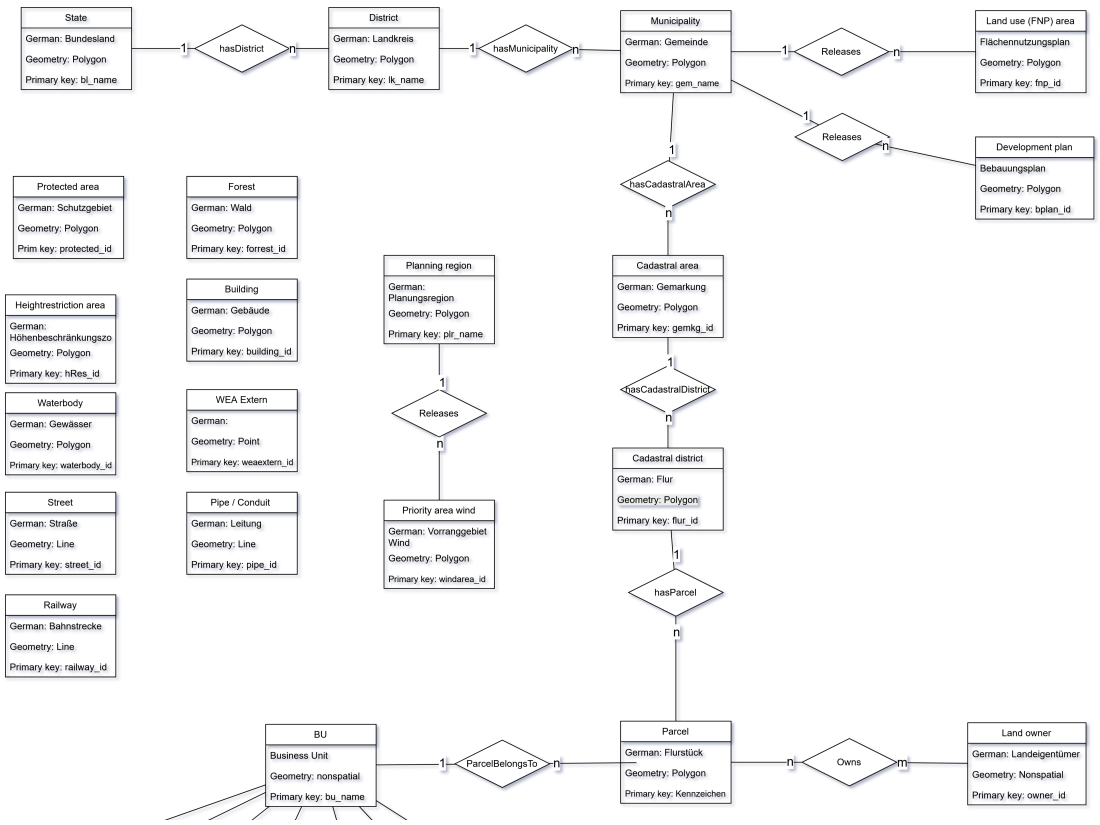


Figure 3.3: ERM of the Authority Data

3.3 Authority Data

This part of the model represents relatively static datasets that are updated only when authorities publish new data or when legal regulations change. Providers such as https://nefino.de/ offer bundled datasets that can be used to populate these entities. The proposed model (Figure 3.3) represents the hierarchical structure of cadastral and administrative units: Flurstücke (parcels) belong to Fluren (cadastral districts), which belong to Gemarkungen (cadastral areas), which belong to Gemeinden (municipalities), which belong to Landkreise (districts), which belong to Bundesländer (states). Since one parcel can have multiple landowners and one owner can hold multiple parcels, the <owns< relationship is modeled as many-to-many. This includes both types of landowners: natural persons and *rechtliche Einheiten* (legal entities, e.g., associations or churches).

The model also displays which authorities may issue land-use plans (FNP), development plans (B-Plans) and priority areas for wind energy (Vorranggebiete Wind). Beyond the hierarchical structure, several thematic or environmental datasets are integrated for contextual analysis: Protected areas, height restriction zones, water bodies, forests, buildings, external WEA, pipes/conduits, streets, and railways. Their detailed administrative dependencies and relations are not explicitly modeled at this stage, given that they are not critical for the current scope of the project.

3.4 Attribute Lists

For each entity represented in the ERM, an attribute list defines the key characteristics that describe the entity. One can distinguish between primary attributes, those that store unique or directly observed information (e.g. the object ID or its name), and secondary attributes which can be calculated from primary attributes of the entity itself or of a relative entity. In addition to that there are generic metadata attributes such as timestamps (`created_at`, `updated_at`), editor information (`created_by`, `updated_by`), and geometry-related metrics such as coordinates (for `Point` or `MultiPoint` entities), line length (for `LineString` or `MultiLineString` entities), or area (for `Polygon` or `MultiPolygon` entities).

Attributes are intentionally omitted from the ERM in Figure 3.3) and Figure 3.3 to keep the diagram clear and focused on conceptual relationships. Table 3.1 shows the suggested attribute list of WEA as an exemplary entity. A set of all entities attribute lists can be found in the Appendix A.2. The set of all attribute lists is a preliminary proposal and the final decisions on attribute selection and standardization should be made in consultation with the responsible teams. An attribute list specifies the attributes names, types, and descriptions. The type indicates the data type but also whether an attribute serves as a primary key (pk) or foreign key (fk), thereby documenting the underlying relationships between entities.

Attribute	**Type**	**Description**
wea_id	integer, pk	ID to uniquely identify a wind turbine (WEA).
bu_name	text, fk of bu	Name of the related Business Unit.
model	text	Manufacturer of the wind turbine (e.g., Vestas, Nordex, Enercon).
capacity	text	Rated power output of the turbine in megawatts (MW).
ground_elev	double	Elevation of the ground where the turbine is installed, measured above sea level (m).
hub_ht	double	Height from the ground to the rotor hub (m).
rot_dia	double	Total rotor diameter, distance across the blades (m).
total_ht	double	Total height from ground to highest blade tip (m).
wea_nr	text	Turbine number. Should be unique within each Business Unit.
fun_dia	double	Diameter of the turbine foundation (m).
con_rad	double	Construction circle radius for assembly, maintenance, and crane operations (m).
rot_wind	double	Main wind direction at the turbine location, expressed in degrees.
status_wea	text	Operational state of the turbine (Planned, Operational, Maintenance, Decommissioned).
nxw_share	text	NextWind's ownership share or percentage.

Table 3.1: Proposed attributes of the wind turbine (WEA) entity

Chapter 4

Database Setup, Utilization and Automation

Once the data model and all attributes have been defined, it can be implemented in a relational database. We use PostgreSQL together with its spatial extension PostGIS [6, 5]. The extension allows us to directly store and process spatial data, enabling spatial queries and geometric operations to be executed efficiently inside the database.

The database can be either self-hosted (for full control over configuration and access) or operated through a cloud service provider. For this prototype, we use a PostgreSQL instance hosted by Neon (https://neon.com/), which offers the required administrative privileges like implementing the schema, installing extensions, and creating trigger functions. Avoid using connection pooling, as it can cause issues with triggers, and session-specific settings, which may also affect desktop GIS connections.

4.1 Creation of entities and relations

In the first step, the model structure needs to be inserted into the database. All entities and their relationships from the ERM are transferred into database tables as listed in the attribute lists. Each entity becomes a table in the database, while relationships between entities are represented through foreign keys. An exception is the <owns> relationship of the authority data. Since it is the only many-to-many relationship a separate linking table is required to be added to the database.

For entities containing spatial information, the geometry column must be defined with the correct geometry type (e.g. `Point`, `LineString`, `Polygon`, `MultiPolygon`, etc.). We store geometries in WGS 84 / EPSG:4326 for consistency, but use EPSG:25832 for calculations of lengths and areas, since EPSG:4326 is not metric [3]. The `geom` column, which stores the geometries in Well-Known Binary (WKB) format [4], is always defined as NOT NULL.

As an example, Script 4.1 shows the transformation of the WEA entity attribute list (As seen in Table 3.1) into SQL Code. The complete SQL-Script to create all entities and relations can be found in the Appendix B.1.

Script 4.1 SQL Example — Creating the WEA table

```sql
CREATE TABLE wea(
    wea_id serial PRIMARY KEY,
    bu_name text REFERENCES bu(bu_name) ON UPDATE CASCADE,

    model text,
    capacity text,
    ground_elev double precision,
    hub_ht double precision,
    rot_dia double precision,
    total_ht double precision,
    wea_nr text,
    fun_dia double precision,
    con_rad double precision,
    rot_wind double precision,
    status_wea text,
    nxw_share text,

    -- Coordinates
    latitude_dd double precision,
    longitude_dd double precision,
    latitude_dms text,
    longitude_dms text,
    utm_zone text,
    utm_easting double precision,
    utm_northing double precision,

    -- Metadata
    created_at timestamptz DEFAULT now(),
    updated_at timestamptz DEFAULT now(),
    created_by text DEFAULT current_user,
    updated_by text DEFAULT current_user,

    geom geometry(Point, 4326) NOT NULL
);
```

4.2 Querying examples

Querying describes the process of interacting with the database, such as retrieving, adding, modifying, or removing information. Furthermore, queries are the main tool to transform the database from a static data store into a dynamic analytical resource. Operations are performed through SQL syntax and include `SELECT` (to retrieve data or create filtered datasets), `INSERT` (to add new records), `UPDATE` (to modify existing entries), `DELETE` (to remove records) and `JOIN` (to combine information from multiple tables based on relationships), but also many more. In practice, even changes made through desktop GIS interfaces are internally translated into SQL operations. Script 4.2 to Script 4.7 show several example queries illustrating possible use cases in the current setup.

Script 4.2 SQL Example — Insert a new wind turbine

```
INSERT INTO wea (bu_name, model, capacity, ground_elev, hub_ht,
    rot_dia, total_ht, wea_nr, fun_dia, con_rad, rot_wind,
    status_wea, nxw_share, geom)
VALUES ('Altmark', 'N-175', '6.8', '45', '179', '175', '266',
    'W01', '30', '106', '45', 'In␣Planning', '100%',
    ST_SetSRID(ST_MakePoint(11.4512112, 52.7386130), 4326));
```

Script 4.3 SQL Example — Update an attribute

```
UPDATE wea SET status_wea = 'In␣Operation'
WHERE bu_name = 'Altmark' AND wea_nr = 'W01';
```

Script 4.4 SQL Example — Delete an outdated object

```
DELETE FROM potential_area WHERE potential_id = '42';
```

4.3 Views

A view is a saved query that can combine or filter data from multiple tables and presents the result as a single virtual dataset. Views display up-to-date information from the underlying tables without requiring users to understand the internal data structure. A view can be queried via: `SELECT [...] FROM [viewname]`. They are particularly useful for simplifying frequently used or complex queries and can provide ready-to-use data layers for integration into external systems. In the following, two examples illustrate how views can simplify access to relevant project information.

Script 4.5 SQL Example — Average length of all cable routes

```sql
SELECT AVG(length_geom) AS avg_cable_length_m FROM cable_route;
```

Script 4.6 SQL Example — Owner with the most parcels (+ parcel list)

```sql
SELECT o.firstname, o.secondname, COUNT(p.parcel_id) AS
    parcel_count, STRING_AGG(p.parcel_nr, ', ') AS parcel_list
FROM land_owner o
JOIN parcel_owner po ON o.owner_id = po.owner_id
JOIN parcel p ON p.parcel_id = po.parcel_id
GROUP BY o.owner_id
ORDER BY parcel_count DESC
LIMIT 1;
```

Script 4.7 SQL Example — Rank Business Units by distance to Buxtehude

```sql
-- Assume the centroid of all WEA defines the BU location.
SELECT
    bu_name,
    ST_Distance(
        ST_Centroid(ST_Collect(w.geom))::geography,
        ST_SetSRID(ST_MakePoint(9.7036, 53.4763), 4326)::geography
    ) AS distance_m
FROM wea AS w
GROUP BY bu_name
ORDER BY distance_m;
```

4.3.1 Example: Combine parcels with landowner names

In the early development phase of wind farms, negotiations with landowners regarding parcel leasing are frequent. To support this process, maps showing which parcels belong to which owners are required. The database, however, stores parcels and landowners in separate tables. The view shown in Script 4.8 defines a query that joins these tables to create a virtual dataset listing each parcel along with the names of its owners. This allows project teams to directly use the view as a layer in desktop GIS applications. The query of the view can easily be adjusted to include more detailed owner information if needed.

Script 4.8 SQL Example — Create a view to lists parcels with landowner names

```sql
CREATE OR REPLACE VIEW gis_views.parcels_with_ownernames AS
SELECT
    p.geom,
    p.parcel_id,
    p.bu_name,
    p.kennz,
    STRING_AGG(
        TRIM(
            COALESCE(o.secondname, '') || ', ' || COALESCE(o.firstname, '')
        ),
        ', ' ORDER BY o.firstname, o.secondname
    ) AS owners
FROM parcel p
LEFT JOIN parcel_owner po ON po.parcel_id = p.parcel_id
LEFT JOIN land_owner o ON o.owner_id = po.owner_id
GROUP BY p.parcel_id, p.bu_name, p.kennz
ORDER BY p.parcel_id;
```

4.3.2 Example: Table for WEA approval Bundeswehr

The Bundeswehr (German Armed Forces) are involved in the approval process for wind turbines under the Federal Immission Control Act (Bundes-Immisionsschutzgesetzt or BImSchG). Its role is to ensure that military interests (such as radar installations or flight paths) are not affected by any wind turbines with a total height of more than 50 meters. Information about each turbine must be submitted to the Bundeswehr in a specific tabular format (see Figure 4.1). The view shown in Script 4.9 defines a query that joins the tables WEA, Gemarkung, Flur, and Parcel based on their spatial overlaps to produce a virtual dataset matching this format. Note that the view's substring parser which splits the coordinates only works if the entries contain all values in the correct form (which gets ensured through geometry-based attribute recalculation in Section 4.4.2). The dataset can then be directly exported as an Excel file, eliminating the need for manual data entry and ensuring that the information provided is consistent and up to date.

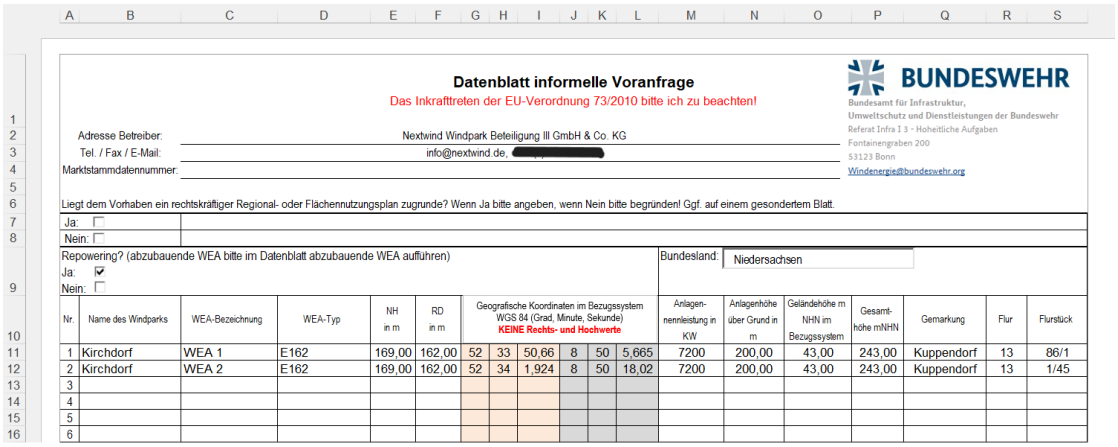
Script 4.9 SQL Example — Create a view to output WEA information in the required Bundeswehr format

```
CREATE OR REPLACE VIEW gis_views.bundeswehr_approval AS
SELECT
  w.wea_id,
  w.bu_name                    AS "Name des Windparks",
  w.wea_nr                     AS "WEA-Bezeichnung",
  w.model                      AS "WEA-Typ",
  w.hub_ht                     AS "NH in m",
  w.rot_dia                    AS "RD in m",

  (substring(w.latitude_dms FROM $$([0-9]+)°$$))::INT
    AS "Nord Grad",
  (substring(w.latitude_dms FROM $$([0-9]+)min$$))::INT
    AS "Nord Minute",
  (substring(w.latitude_dms FROM $$([0-9]+(?:\.[0-9]+)?)sek$$))::NUMERIC
    AS "Nord Sekunde",
  (substring(w.longitude_dms FROM $$([0-9]+)°$$))::INT
    AS "Ost Grad",
  (substring(w.longitude_dms FROM $$([0-9]+)min$$))::INT
    AS "Ost Minute",
  (substring(w.longitude_dms FROM $$([0-9]+(?:\.[0-9]+)?)sek$$))::NUMERIC
    AS "Ost Sekunde",

  w.capacity                   AS "Anlagennennleistung in KW",
  w.total_ht                   AS "Anlagenhöhe über Grund in m",
  w.ground_elev                AS "Geländehöhe m NHN im Bezugssystem",
  (COALESCE(w.total_ht,0) + COALESCE(w.ground_elev,0))
    AS "Gesamt-höhe mNHN",
  gm.gem_name                  AS "Gemarkung",
  fl.flur_nr                   AS "Flur",
  pa.parcel_nr                 AS "Flurstück",
  w.geom::geometry(Point,4326) AS geom

FROM  public.wea   w
LEFT JOIN LATERAL (
  SELECT g.gem_name
  FROM public.gemarkung g
  WHERE ST_Contains(ST_Transform(g.geom,4326), w.geom)
  LIMIT 1
) gm ON true
LEFT JOIN LATERAL (
  SELECT f.flur_nr
  FROM public.flur f
  WHERE ST_Contains(ST_Transform(f.geom,4326), w.geom)
  LIMIT 1
) fl ON true
LEFT JOIN LATERAL (
  SELECT p.parcel_nr
  FROM public.parcel p
  WHERE ST_Contains(ST_Transform(p.geom,4326), w.geom)
  LIMIT 1
) pa ON true;
```



Datenblatt informelle Voranfrage

Das Inkrafttreten der EU-Verordnung 73/2010 bitte ich zu beachten!

BUNDESWEHR

Bundesamt für Infrastruktur, Umweltschutz und Dienstleistungen der Bundeswehr
Referat Infra I 3 - Hoheitliche Aufgaben
Fontainengraben 200
53123 Bonn
Windenergie@bundeswehr.org

Adresse Betreiber: Nextwind Windpark Beteiligung III GmbH & Co. KG
Tel. / Fax / E-Mail: info@nextwind.de, [redacted]
Marktstammdatennummer:

Liegt dem Vorhaben ein rechtskräftiger Regional- oder Flächennutzungsplan zugrunde? Wenn Ja bitte angeben, wenn Nein bitte begründen! Ggf. auf einem gesondertem Blatt.

Ja: ☐
Nein: ☐

Repowering? (abzubauende WEA bitte im Datenblatt abzubauende WEA aufführen)
Ja: ☑
Nein: ☐

Bundesland: Niedersachsen

Nr.	Name des Windparks	WEA-Bezeichnung	WEA-Typ	NH in m	RD in m	Geografische Koordinaten im Bezugssystem WGS 84 (Grad, Minute, Sekunde) KEINE Rechts- und Hochwerte						Anlagen-nennleistung in KW	Anlagenhöhe über Grund in m	Geländehöhe m NHN im Bezugssystem	Gesamt-höhe mNHN	Gemarkung	Flur	Flurstück
1	Kirchdorf	WEA 1	E162	169,00	162,00	52	33	50,66	8	50	5,665	7200	200,00	43,00	243,00	Kuppendorf	13	86/1
2	Kirchdorf	WEA 2	E162	169,00	162,00	52	34	1,924	8	50	18,02	7200	200,00	43,00	243,00	Kuppendorf	13	1/45
3																		
4																		
5																		
6																		

Figure 4.1: Excel template used for Bundeswehr requests in the BImSchG approval process

4.4 Automation with Trigger Functions

To ensure data consistency and reduce manual maintenance, the database makes use of trigger functions that are automated routines which execute whenever certain database events occur. A trigger is a database mechanism that "listens" for changes to a table (e.g. an `INSERT`, `UPDATE`, or `DELETE` operation) and executes a predefined function in response. Triggers do not distinguish between operations originating from SQL query tools or from desktop GIS software. The underlying trigger functions are written in PL/pgSQL (Procedural Language / PostgreSQL), which allows procedural logic such as conditional statements, loops, and arithmetic operations to be combined with standard SQL. Triggers are a powerful way to automate recurring tasks or maintain derived attributes without requiring user intervention. The proposed triggers for the database cover two routines, as described in Sections 4.4.1 and 4.4.2. Script 4.10 illustrates the implementation of the two routines for entities of geometry type `Polygon` or `MultiPolygon`. The complete SQL-Script for creating and linking all triggers is provided in the Appendix B.2.

4.4.1 Automatic metadata tracking

Whenever an entry in any main table is inserted or modified, the trigger automatically fills the `updated_at` attribute with the timestamp of the moment of the entry modification. Furthermore, the `updated_by` attribute gets filled with the username through which the user established the database connection. This ensures a transparent edit history without requiring manual input from users.

4.4.2 Geometry-based attribute calculation

For spatial tables, geometric metadata attributes are automatically (re-)calculated whenever the value of the `geom` column changes or a new entry gets inserted. This is useful for maintaining correct values for derived metrics, e.g. when turbine locations are moved or polygon boundaries are corrected. Depending on geometry type the generic metadata attributes `area_geom` (for `Polygon` or `MultiPolygon`), the `length_geom` (for `LineString` or `MultiLineString`), or the coordinate representations (for `Point` or `MultiPoint`) get (re-)calculated. Coordinate representations include the attributes: `latitude_dd` and `longitude_dd` (for decimal degrees representation), `latitude_dms` and `longitude_dms` (for representation in degrees–minutes–seconds) and `utm_zone`, `utm_northing` and `utm_easting` (for representation within their Universal Transverse Mercator Zone).

Script 4.10 SQL Example — Metadata tracking and area calculation for polygons

```
CREATE OR REPLACE FUNCTION set_metadata_polygon()
RETURNS TRIGGER AS $$
BEGIN
    NEW.updated_at := now();
    NEW.updated_by := current_user;
    IF NEW.geom IS DISTINCT FROM OLD.geom THEN
        NEW.area_geom := ST_Area(ST_Transform(NEW.geom, 25832));
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Chapter 5

Conclusion

The goal of this report was to design a data model that captures the most essential geospatial aspects of NeXtWind's digital twin and aligns them with the company's internal structure and strategic needs. To support this model, a PostgreSQL/PostGIS database was proposed as the central geospatial infrastructure, together with a set of functionalities such as views and trigger functions, that optimize data access, automation, and consistency. Throughout the report, short explanations of ERM modeling principles and relevant SQL concepts were provided to enable readers to understand the design decisions and functionality.

The proposed model intentionally includes only a minimal set of entities and attributes. This reflects the current level of available domain knowledge and acknowledges the complexity of wind farm planning. However, the model is deliberately structured in an open and extensible way so that additional attributes, entities, or constraints can be added as workflows develop and more detailed requirements become clear. Potential extensions include: (1) defining more attributes and allowed value constraints to improve data quality and consistency, (2) expanding the integration of environmental entities into the administrative authority structure to support more advanced site assessments, and (3) introducing a contract entity to enable closer alignment with the Commercial Asset Management Team, allowing more precise management and visualization of different contractual rights tied to land parcels, for example separate contracts for turbine locations, cable routes, or substations.

From a technical perspective, the report demonstrated a possible implementation pathway: defining and creating tables based on the data model, designing views for simplified and standardized access to datasets, and implementing trigger functions to automate updates and calculations on metadata attributes. Additional database administration topics such as user and user group creation or privilege management could be addressed in future work to support a secure multi-user environment. Further enhancements might include using PostgreSQL extensions like `temporal_tables` or `pg_partman` to introduce system versioning through historical tables.

Looking ahead, several development directions are possible. A WebGIS interface or a GeoServer-based service layer could be built on top of the database, exposing only prepared views as Web Feature Services or Web Map Services to ensure that users work with clean, standardized datasets rather than raw tables. Interfaces to other tools,

such as Airtable, could be established through API-based synchronization. Automated map-generation tools, producing contract maps for specific Business Units or landowners could further streamline planning workflows.

By identifying the challenges in the current workflow, introducing a coherent data model, and outlining an implementation based on PostgreSQL/PostGIS, this report proposes a more organized and reliable approach to managing spatial information. We aim to provide a clear overview of the energy sector context, data modeling principles, and relational database concepts. Ultimately, the report is intended to offer practical value to the organization and support informed decision-making as NeXtwind's digital strategy continues to evolve.

Bibliography

[1] Thomas Brinkhoff. *Geodatenbanksysteme in Theorie und Praxis*. VDE Verlag, Berlin, 4th revised and expanded edition edition, 2022. ISBN 978-3-87907-694-9.

[2] Peter Pin-Shan Chen. The entity–relationship model — toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976. doi: 10.1145/320434.320440.

[3] International Association of Oil & Gas Producers. Epsg geodetic parameter dataset. Geodetic reference dataset, 2025. URL https://epsg.org.

[4] Open Geospatial Consortium. Opengis implementation standard for geographic information — simple feature access. Implementation Standard OGC 06-103r4, OGC, 2011. URL https://www.ogc.org/standards/sfa.

[5] PostGIS Project. Postgis documentation. Online documentation, 2025. URL https://postgis.net/documentation/.

[6] PostgreSQL Global Development Group. Postgresql documentation. Online documentation, 2025. URL https://www.postgresql.org/docs/.

[7] Technische Universität Berlin. Informationssysteme und datenanalyse. Online lecture material, 2025. URL https://isda-sose25-9dee2e.gitlab-pages.tu-berlin.de. Institute of Software and Database Technology.

Appendix

All materials, diagrams, SQL scripts and datasets used in this report are publicly available in the associated GitHub repository:
https://github.com/hendrini/Intern_Report

Appendix Overview

ID	Material Description	Source / GitHub Path
A.1	ERM diagram file for import into https://draw.io/	https://github.com/hendrini/Intern_Report/blob/main/Datamodel/ERM.drawio
A.2	Attribute Lists (complete attribute specifications for all entities)	https://github.com/hendrini/Intern_Report/blob/main/Datamodel/Attribute_Lists.xlsx
B.1	SQL script — creation of all entity and relationship tables + views	https://github.com/hendrini/Intern_Report/blob/main/SQL/create_tables_and_views.sql
B.2	SQL script — implementation of automated trigger routines (metadata + geometry calculations)	https://github.com/hendrini/Intern_Report/blob/main/SQL/trigger_functions.sql

Table 5.1: List of all appendix materials and their storage locations

Acknowledgments

Special thanks to Behrooz Rostami, Victor Garzón and Kamen Radew for their valuable input and support throughout and beyond the development of this report.