Martin Stobbelaar
StudentID: 12131617

Exercise 13: Database backend for a WebGIS

Festival selection

While the company *Vudstork Entertainment Ltd* is specialized on festivals for medium sized cities, they got a request from the city of Munich to create a mobile app for the Oktoberfest, as they were amazed by other similar applications done by the company. I got the task to build the spatial database around the app.

Data preparation

To get started, I first look for already existing data. As there is no official spatial data for the Oktoberfest to download, I downloaded the area of the *Theresienwiese* from OSM. A lot of tents and streets are already mapped and have to be only cleaned and converted to the correct geometry type.

Next to already existing I manually mapped everything else in QGIS, with the help of a map from the Oktoberfest that can be seen in figure 1 on the left. The right map is the manually mapped area in QGIS.

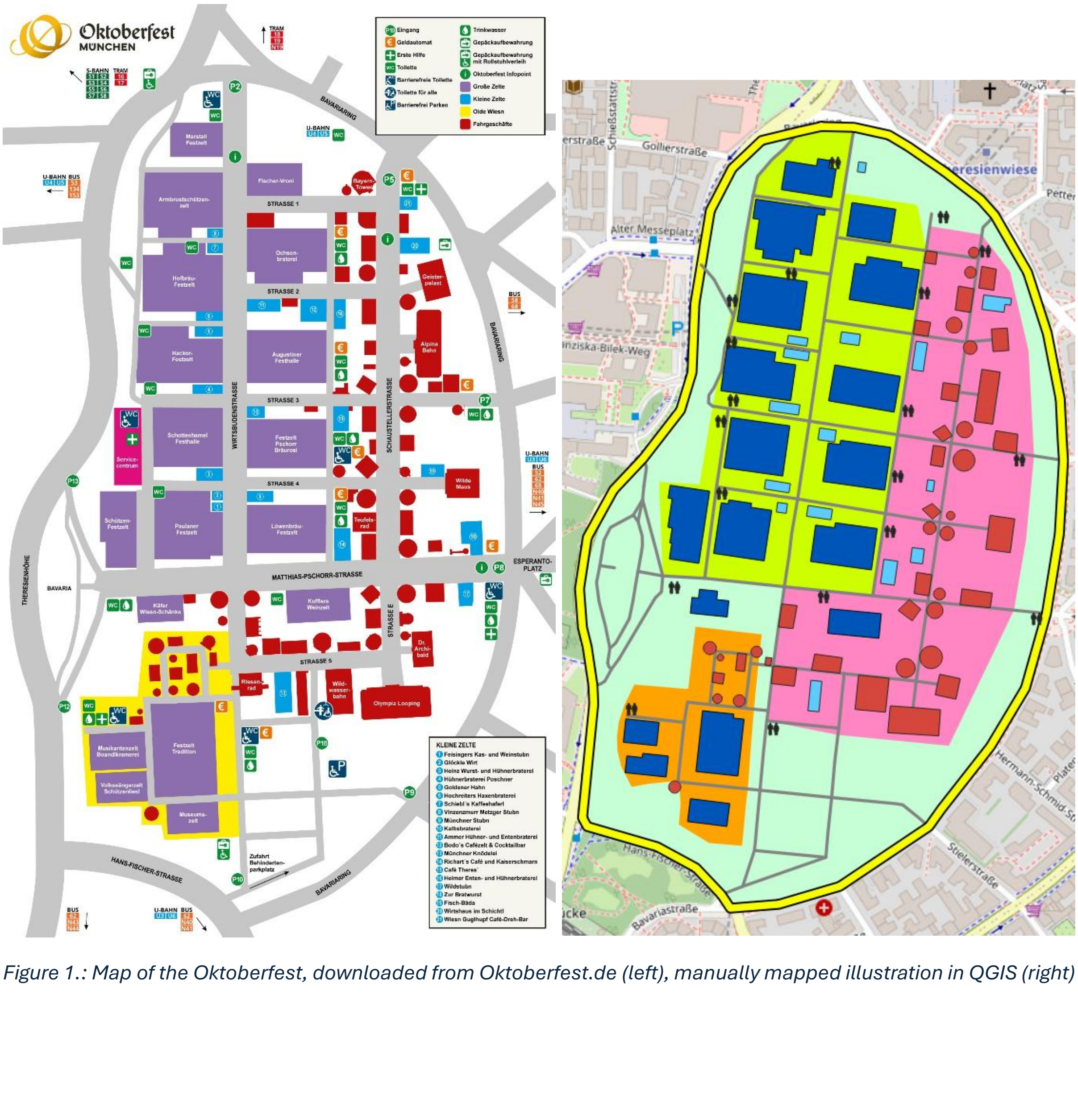


Figure 1.: Map of the Oktoberfest, downloaded from Oktoberfest.de (left), manually mapped illustration in QGIS (right)

List of entities

There are several entities on the Oktoberfest, some of them have a physical location e.g. a tent, whilst other are something that happens at a location (e.g. events). In the following table 1 is a list of all entities that exist in my database.

Entities	**Relationship (if existing)**
Tents	Is part of locations
Small tents	Is part of locations
Rides	Is part of locations
Toilets	Is part of locations
ATMs	Is part of locations
Areas	Contain physical entities- not observable
Events	Take place in/at tents/rides or other locations
Visitor	Has certain location at a specific time
Entrances/Exits	Is part of locations
Locations	List of all locations

All entities with geometries and spatial information are uploaded to the new database with the methods learned in one of the first exercises using the Database Manager in QGIS. All other entities are created as tables directly using SQL in pgAdmin.

The "locations" table contains all locations on the festival. Although no tents, rides etc. exist more then once at the same location, this would make that possible. Its also easier to search for events.

Graphical overview of database

In the following figure 2 (also attached in the zip file), the physical data model of the database can be seen. This graphical overview has been created with the ERD tool in pgAdmin.

The database is quite simple as except for the location table there aren't any obvious relations. The columns and their datatypes are set in a way that they allow specific spatial and temporal queries from the visitors perspective. There are multiple visitors in the "visitor" table, allowing queries from varying positions and daytimes. This makes showcasing the database a little bit easier.

As can be seen in the graphical overview; tents, small tents and rides have columns that give information on the opening times. As it is assumed that they open every day to the same hours during the festival, they have open- and closing times with the datatype "time without time zone". Events that take place have open and closing time with the datatype "timestamp without time zone" as they only last a few hours.

Visitors also have a column called timestamp with the datatype "timestamp without time zone", to simulate a visit during a specific time.

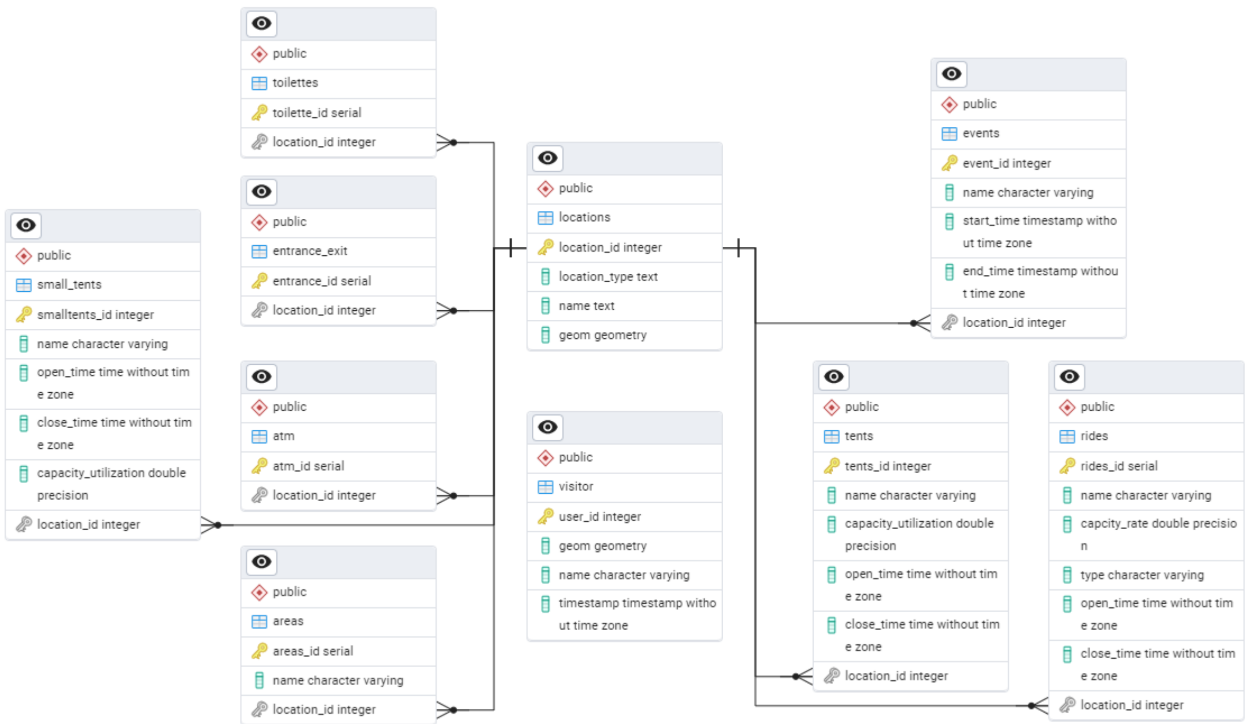


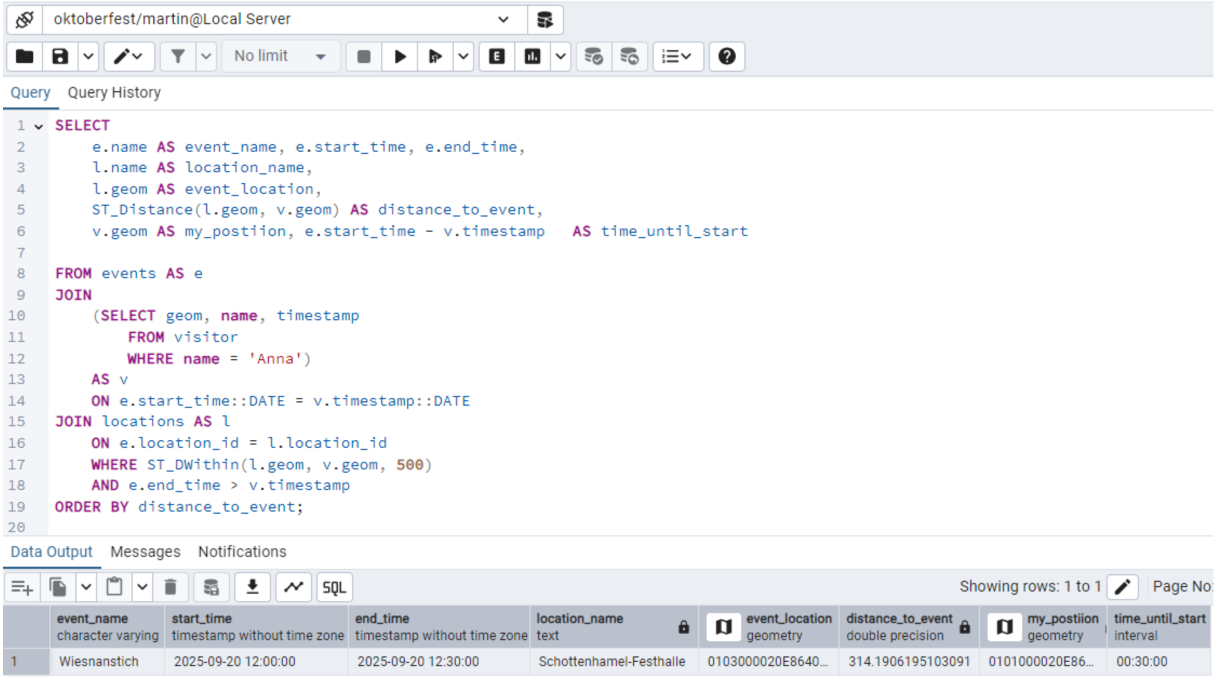
Figure 2.: Physical data model of the database

Example queries

All queries are also stored as SQL files and contained in the ZIP file. Following are some scenarios of visitors needing specific information and the query that retrieves the information.

Scenario 1:

Anna likes to see the festive start of the Oktoberfest and is wondering in which tent it takes place, however she does not want to walk more than 500m from her current position. Also has she forgotten when exactly the event starts. The following query (figure 3) lists all events near her, that take place on the same day as her current time and that have not ended yet.




```sql
SELECT
    e.name AS event_name, e.start_time, e.end_time,
    l.name AS location_name,
    l.geom AS event_location,
    ST_Distance(l.geom, v.geom) AS distance_to_event,
    v.geom AS my_postiion, e.start_time - v.timestamp   AS time_until_start

FROM events AS e
JOIN
    (SELECT geom, name, timestamp
        FROM visitor
        WHERE name = 'Anna')
    AS v
    ON e.start_time::DATE = v.timestamp::DATE
JOIN locations AS l
    ON e.location_id = l.location_id
    WHERE ST_DWithin(l.geom, v.geom, 500)
    AND e.end_time > v.timestamp
ORDER BY distance_to_event;
```

	event_name character varying	start_time timestamp without time zone	end_time timestamp without time zone	location_name text	event_location geometry	distance_to_event double precision	my_postiion geometry	time_until_start interval
1	Wiesnanstich	2025-09-20 12:00:00	2025-09-20 12:30:00	Schottenhamel-Festhalle	0103000020E8640...	314.1906195103091	0101000020E86...	00:30:00

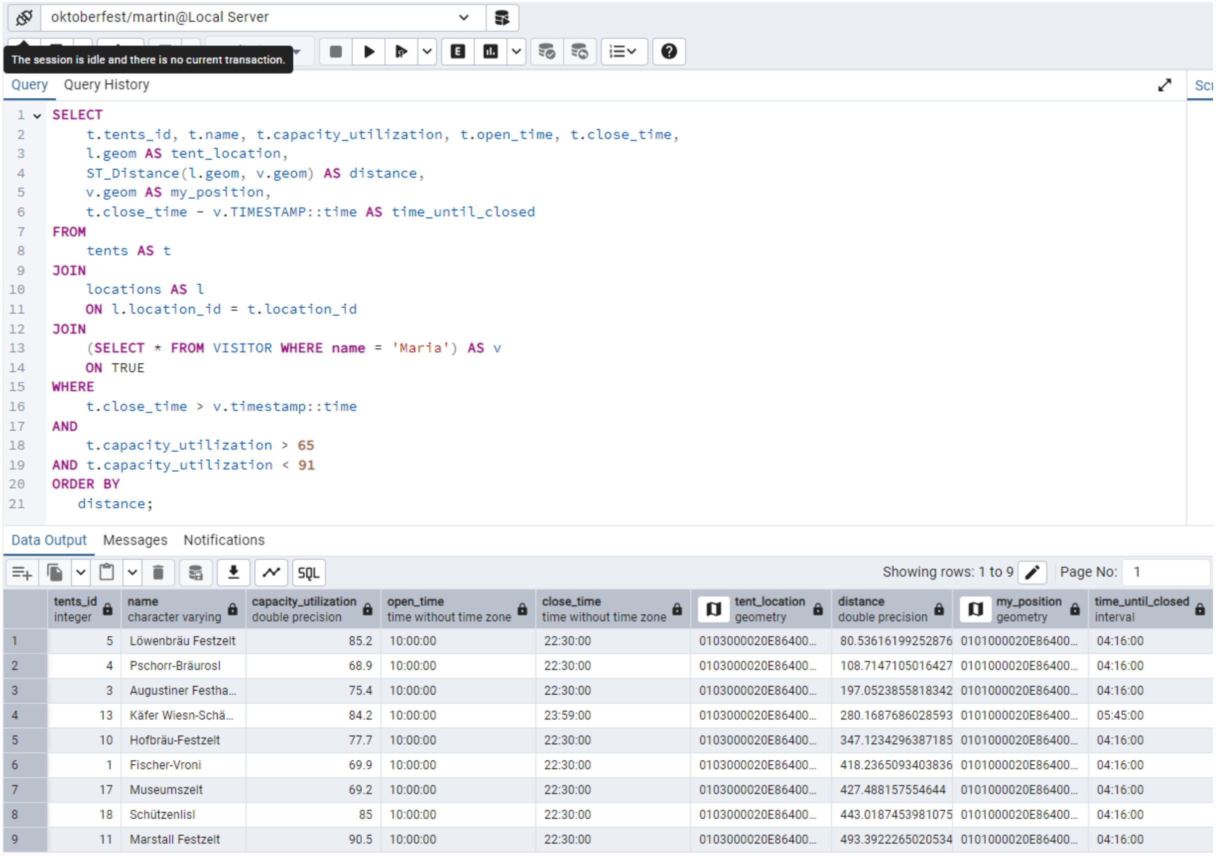
Figure 3.: Query to find events within 500m

The query result is tailored towards the person needing the information.

Scenario 2:

Maria wants to enjoy some beers in of the tents at the Oktoberfest. She likes it crowded, but not so crowded that finding a place is impossible (hence the capacity utilization between 65 and 91). She does not mind walking a little bit, however its always nice to know if nearby options are meeting her criteria. Figure 4 shows the query as well as the output table.

In an actual application, this result should also be available as a map based output, where the event location and the visitors location is highlighted, while all other info should be visible as well. An example how that should look like, can be seen in figure 5: a styled map, created with QGIS after loading the query result as a new layer. Tents are styled differently, depending of the distance towards the visitor.




```sql
SELECT
    t.tents_id, t.name, t.capacity_utilization, t.open_time, t.close_time,
    l.geom AS tent_location,
    ST_Distance(l.geom, v.geom) AS distance,
    v.geom AS my_position,
    t.close_time - v.TIMESTAMP::time AS time_until_closed
FROM
    tents AS t
JOIN
    locations AS l
    ON l.location_id = t.location_id
JOIN
    (SELECT * FROM VISITOR WHERE name = 'Maria') AS v
    ON TRUE
WHERE
    t.close_time > v.timestamp::time
AND
    t.capacity_utilization > 65
AND t.capacity_utilization < 91
ORDER BY
   distance;
```

	tents_id integer	name character varying	capacity_utilization double precision	open_time time without time zone	close_time time without time zone	tent_location geometry	distance double precision	my_position geometry	time_until_closed interval
1	5	Löwenbräu Festzelt	85.2	10:00:00	22:30:00	0103000020E86400...	80.53616199252876	0101000020E86400...	04:16:00
2	4	Pschorr-Bräurosl	68.9	10:00:00	22:30:00	0103000020E86400...	108.7147105016427	0101000020E86400...	04:16:00
3	3	Augustiner Festha...	75.4	10:00:00	22:30:00	0103000020E86400...	197.0523855818342	0101000020E86400...	04:16:00
4	13	Käfer Wiesn-Schä...	84.2	10:00:00	23:59:00	0103000020E86400...	280.1687686028593	0101000020E86400...	05:45:00
5	10	Hofbräu-Festzelt	77.7	10:00:00	22:30:00	0103000020E86400...	347.1234296387185	0101000020E86400...	04:16:00
6	1	Fischer-Vroni	69.9	10:00:00	22:30:00	0103000020E86400...	418.2365093403836	0101000020E86400...	04:16:00
7	17	Museumszelt	69.2	10:00:00	22:30:00	0103000020E86400...	427.488157554644	0101000020E86400...	04:16:00
8	18	Schützenlisl	85	10:00:00	22:30:00	0103000020E86400...	443.0187453981075	0101000020E86400...	04:16:00
9	11	Marstall Festzelt	90.5	10:00:00	22:30:00	0103000020E86400...	493.3922265020534	0101000020E86400...	04:16:00

Figure 4.: Query to find tents with a capacity utilization between 65 and 91

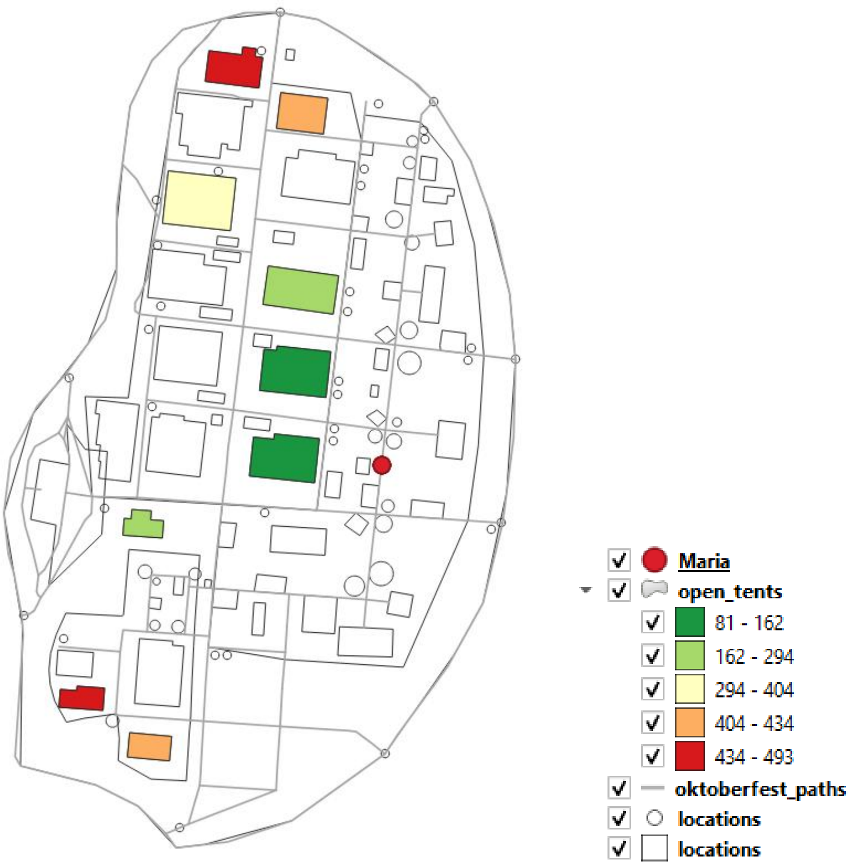
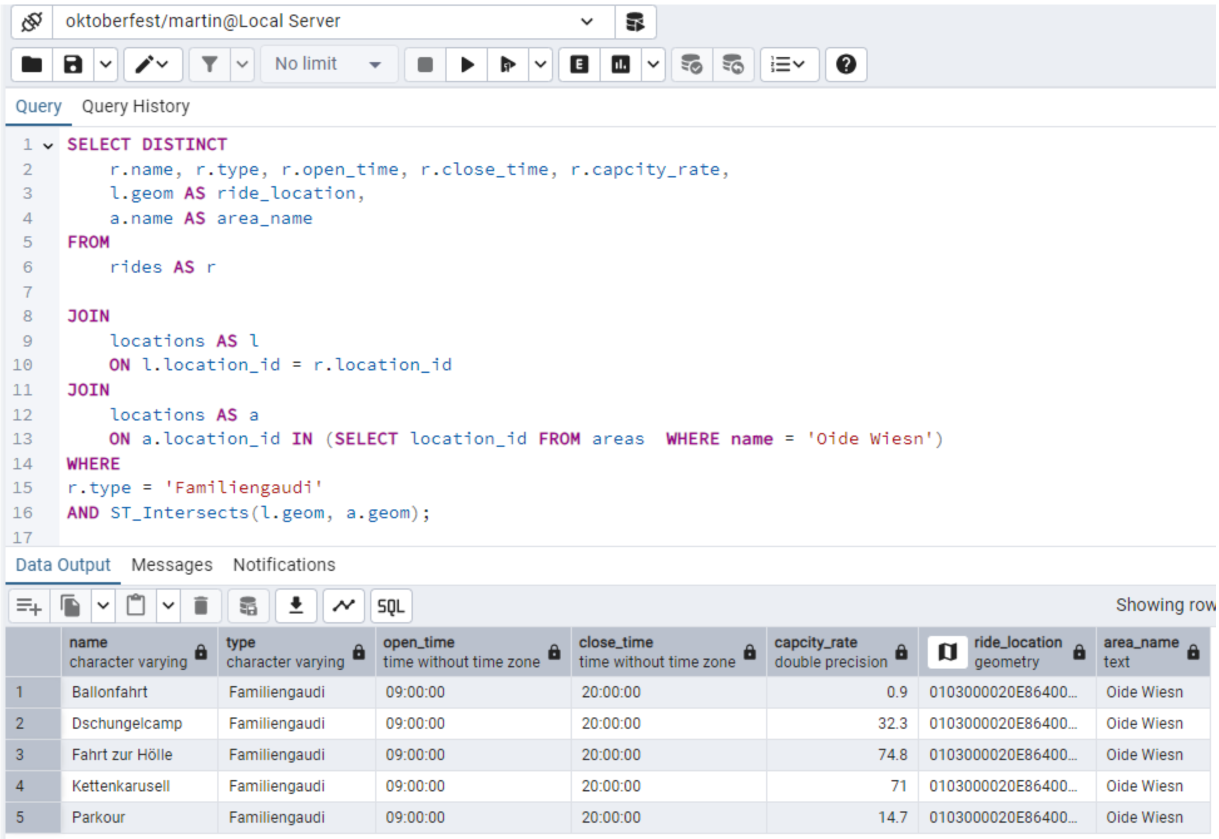


Figure 5.: Example map created in QGIS. Showing Marias postion and highlighted tents

Scenario 3:

Josef wants to get information on where to go with his grandkids before going to the Oktoberfest so he is well prepared. He likes the sound of the "Oide Wiesn", the more traditional part of the Oktoberfest. He wants to know which family friendly rides (ride type: "Familiengaudi") are available in this area of the festival.

Figure 6 shows the query which gives Josef the results he wants.




```sql
SELECT DISTINCT
    r.name, r.type, r.open_time, r.close_time, r.capcity_rate,
    l.geom AS ride_location,
    a.name AS area_name
FROM
    rides AS r

JOIN
    locations AS l
    ON l.location_id = r.location_id
JOIN
    locations AS a
    ON a.location_id IN (SELECT location_id FROM areas  WHERE name = 'Oide Wiesn')
WHERE
r.type = 'Familiengaudi'
AND ST_Intersects(l.geom, a.geom);
```

	name character varying	type character varying	open_time time without time zone	close_time time without time zone	capcity_rate double precision	ride_location geometry	area_name text
1	Ballonfahrt	Familiengaudi	09:00:00	20:00:00	0.9	0103000020E86400...	Oide Wiesn
2	Dschungelcamp	Familiengaudi	09:00:00	20:00:00	32.3	0103000020E86400...	Oide Wiesn
3	Fahrt zur Hölle	Familiengaudi	09:00:00	20:00:00	74.8	0103000020E86400...	Oide Wiesn
4	Kettenkarusell	Familiengaudi	09:00:00	20:00:00	71	0103000020E86400...	Oide Wiesn
5	Parkour	Familiengaudi	09:00:00	20:00:00	14.7	0103000020E86400...	Oide Wiesn

Figure 6.: Retrieving specific rides in specific area

Other possible queries:

Figure 7 and 8 show further possible queries that could be useful for visitors. Of course a lot of different queries can be executed, the possibilities are endless. These queries are just a selection of things that might be useful for gathering information about the festival.

All queries can be altered dynamically towards other visitors or locations to retrieve the desired output.

oktoberfest/martin@Local Server

```sql
SELECT
    l.*,
    ST_DISTANCE(l.geom,v.geom) as DISTANCE
FROM
    locations as l
JOIN
    (SELECT * FROM visitor WHERE name = 'Herbert') as v
    ON TRUE
WHERE st_distance (l.geom,v.geom) < 100
AND NOT l.location_type = 'areas'
ORDER by DISTANCE;
```

Data Output

	location_id [PK] integer	location_type text	name text	geom geometry	distance double precision
1	105	toilettes	[null]	0101000020E864...	24.66270626915486
2	68	rides	Hangover - The Tower	0103000020E864...	29.66482071178158
3	14	tents	Fischer-Vroni	0103000020E864...	48.04792960404956
4	15	tents	Ochsenbraterei	0103000020E864...	56.28462682895253
5	75	rides	Bayern Tower	0103000020E864...	60.0051645035312
6	112	atm	[null]	0101000020E864...	64.07852875323925
7	64	rides	Flohzirkus	0103000020E864...	72.04658466557359
8	113	atm	[null]	0101000020E864...	76.12209371521847
9	107	toilettes	[null]	0101000020E864...	79.87311722258224
10	106	toilettes	[null]	0101000020E864...	85.16077172037225
11	67	rides	Breakdancer	0103000020E864...	85.48057846049545
12	71	rides	Big Pictures 2.0	0103000020E864...	85.67782552644063
13	6	entrance_exit	[null]	0101000020E864...	88.64814027069541

Figure 7.:Query to find all physical locations within certain distance of another location (in this case a visitor)

oktoberfest/martin@Local Server

```sql
SELECT
    l.geom as location,l.location_type,
    t.geom as tent_location,
    ST_DISTANCE(l.geom,t.geom) as dist_from_tent
FROM
    locations as l
JOIN
    locations as t
    ON t.location_id IN (SELECT location_id FROM tents WHERE name = 'Ochsenbraterei')
WHERE st_distance (l.geom,t.geom) < 50
AND l.location_type IN ('toilettes','atm')
ORDER by dist_from_tent;
```

Data Output

	location geometry	location_type text	tent_location geometry	dist_from_tent double precision
1	0101000020E864...	toilettes	0103000020E86400...	10.58613437527116
2	0101000020E864...	atm	0103000020E86400...	11.49172582121331

Figure 8.: Query to find all physical locations within certain distance of another location (in this case a visitor)