

# Geodatenverarbeitung mit SQL

GIS in der Datenbank

# Überblick

PostGIS ermöglicht Verarbeitung von Geodaten in PostgreSQL-Datenbanken

- Wie kommen SQL und GIS zusammen?
- Was sind die Rahmenbedingungen?
- Was für Funktionen gibt es?
- Wie kann ich die Funktionen nutzen?



# Räumliches SQL

- Standards im Hintergrund
- Überblick räumliche SQL-Funktionen
- Rahmenbedingungen

# Standards für Räumliches SQL

- Standards im Hintergrund
  - Baut auf dem Simple Feature Datenmodell auf
  - ISO 19125-1 zur Beschreibung der Architektur zur Speicherung von räumlichen Funktionen
  - "Simple Feature Access – Part 2: SQL Option" ist frei zugänglich
  - ISO 13249-3 mit SQL/MM als Standard beschreibt räumliche SQL-Funktionen
- PostGIS implementiert die entsprechenden Funktionen in PostgreSQL
- Nicht immer sind alle Funktionen implementiert, teilweise sind weitere nicht standardisierte Funktionen in PostGIS verfügbar

# Typen von SQL Funktionen

Beispiel für verschiedene Arten von Funktionen im SQL:

Gruppe	Standard SQL	Räumliches SQL	Erklärung
Skalarfunktionen	LENGTH()	ST_Area(), ST_Length()	Rückgabe eines Wertes für jede Eingabe
Aggregatfunktionen	SUM()	ST_Collect(), ST_Union(), ST_Extent()	Rückgabe eines Wertes für eine Gruppe von Eingaben, Verwendung in Kombination mit GROUP BY
Fensterfunktionen	ROW_NUMBER()	ST_ClusterDBSCAN(), ST_ClusterKMeans()	Rückgabe eines Wertes für eine Gruppe pro Eingabe, Kombination mit PARTITION BY möglich
Set Returning Functions	UNNEST()	ST_Dump(), ST_DumpPoints(), ST_DumpRings()	Rückgabe von mehr als einem Werte pro Eingabezeile

# Zugriff auf Geometrien

- Well-Known Text (WKT) als Austauschformat für Geometrien, Beschreibt Punkt, Linien und Flächen in standardisierter Weise
- Erstellung von Geometrien  
`ST_GeomFromText()`
- Ausgabe als WKT `ST_AsWkt()`,  
`ST_AsText()`
- Zugriff auf einzelne Werte einer Geometrie  
`ST_X()`, `ST_Y()`

```
POINT (30 10)
```

```
LINESTRING (30 10, 10 30, 40 40)
```

```
POLYGON ((35 10, 45 45, 15 40, 10 20, 35 10),  
(20 30, 35 35, 30 20, 20 30))
```

# PostGIS Dokumentation

Vollständige Hilfe für PostGIS-Funktionen im Web verfügbar

Beispiel für `ST_Intersects()` [https://postgis.net/docs/ST\\_Intersects.html](https://postgis.net/docs/ST_Intersects.html)

## ST\_Intersects

### Synopsis

```
boolean ST_Intersects( geometry geomA , geometry geomB );
```

```
boolean ST_Intersects( geography geogA , geography geogB );
```

### Description

Returns **true** if two geometries intersect. Geometries intersect if they have any point in common.

For geography, a distance tolerance of 0.00001 meters is used (so points that are very close are considered to intersect).

In mathematical terms:  $ST\_Intersects(A, B) = A \cap B \neq \emptyset$

# Zugriff und Performance

- Index
  - Nutzung von Indexen dient dem schnellen Zugriff auf die Daten
  - Index entspricht dem Inhaltsverzeichnis eines Buches
  - Für Geometriespalten immer einen Index erstellen
- Primärschlüssel
  - Für die eindeutige Identifikation von Datensätzen ist ein Primärschlüssel notwendig
  - Ohne Primärschlüssel keine Bearbeitung möglich
  - Verwendung von Zahlen oder ein UUID

# Hilfreiche SQL-Konstrukte

- Unterabfragen und Common Table Expressions
- Spatial Join

# Unterabfragen

- Unterabfragen, englisch "Subquery", sind Abfragen eingebettet in größere Abfragen
- Vorteile:
  - Mehre Abfragen in einer Abfrage
  - Sehr flexibel Konstrukt
  - Abfrage-Schnipsel lassen sich wiederverwenden
- Nachteile:
  - Wiederholte Ausführung verlangsamt Abfrage,
  - Viele verschachtelte Unterabfragen sind schwer lesbar

```
-- Informationen zusammenstellen
SELECT fid, "name",
  (SELECT TO_CHAR(date_import, 'DD.MM.YYYY')
   FROM imports WHERE table_name = 'forest') AS date_import
FROM forest f

-- Unterabfrage, die eine Tabelle zurückgibt
SELECT ROW_NUMBER() OVER(), fid, name, geom
FROM
  (SELECT fid, "name", (ST_Dump(geom)).geom
   FROM forest f) subquery;

-- Beispiel für eine Unterabfrage in WHERE-Klausel
SELECT fid, "name"
FROM forest f
WHERE fid IN (SELECT id FROM forest_selection);
```

# Common Table Expressions

- Common Table Expressions (CTEs) funktionieren wie temporäre Tabellen innerhalb einer Abfrage
- Vorteile:
  - Bessere Lesbarkeit als Unterabfragen
  - Wiederverwendung in Abfrage möglich
- Nachteile:
  - Performancevorteile durch Indexe in der Datenverarbeitung gehen verloren

```
-- Unterabfrage umgeschrieben als CTE

WITH
single_part AS
(
  SELECT fid, "name", (ST_Dump(geom)).geom
  FROM forest f
)
SELECT ROW_NUMBER() OVER(), fid, name, geom
FROM single_part;
```

## Spatial Join

- Ein Join ist die Verknüpfung von Daten aus zwei Tabellen über einen gemeinsamen Wert
- PostGIS ermöglicht Verknüpfung über räumliche Beziehungen (Overlaps, Contains, Within, Intersect)
- Für eine gute Performance ist ein Index auf der Geometriespalte notwendig!

```
-- Abfrage aller Punkte die Geometrien
-- in der Tabelle polygons schneiden

SELECT *
FROM points a
JOIN polygons b
  ON ST_Intersects(a.geom, b.geom)
```

## Spatial Antijoin

- Ziel eines Antijoin ist es Daten zu finden, die in einer anderen Tabelle nicht existieren
- Räumliches Beispiel: Daten finden die nicht in einem Polygon liegen
- Ausführliche Erläuterungen wie man es implementiert von [crunchydata](#)

```
-- Abfrage aller Punkte die Geometrien  
-- in der Tabelle polygons nicht schneiden
```

```
SELECT *  
FROM points a  
WHERE NOT EXISTS  
  (SELECT 1 FROM polygons b  
   WHERE ST_Intersects(a.geom, b.geom))
```

# Anwendung von Skalarfunktionen

- Abfrage von Eigenschaften
- Zugriff auf Geometrieteile
- Umformung der Geometrien

# Umformung der Geometrien

- Zugriff auf die Geometrie und ihre Eigenschaften
- Eigenschaften: `ST_Area()`, `ST_Length()`, `ST_NumPoints()`
- Teile der Geometrie: `ST_StartPoint()`, `ST_PointN()`

```
-- Ausgabe der Fläche
```

```
SELECT fid, name, ST_Area(geom) AS flaeche  
FROM osm.forest f
```

```
-- Zugriff auf den ersten Punkt der Geometrie
```

```
SELECT fid, name, ST_StartPoint(geom) AS geom  
FROM line
```

# Umformung der Geometrien

- Funktionen ermöglichen die Verarbeitung der Geometrien mit PostGIS
- Änderung des Geometrietyps: `ST_Buffer()`, `ST_Centroid()`, `ST_PointOnSurface()`
- Anpassung der Geometrie: `ST_Simplify()`, `ST_SimplifyVW()`, `ST_ChaikinSmoothing()`

```
-- Umwandlung in Punktgeometrie
```

```
SELECT fid, name, ST_PointOnSurface(geom) AS geom  
FROM osm.forest f
```

```
-- Verringerung der Stützpunktzahl
```

```
SELECT fid, name, ST_Simplify(geom, 4) AS geom  
FROM osm.forest f
```

# Anwendung von Aggregatfunktionen

- Zusammenfassen von Geometrien

# Zusammenfassen von Geometrien

## ST\_Union()

- Verschmelzen der Geometrien
- Rechenintensiver in der Ausführung

```
SELECT "type", ST_Union(geom) AS geom
FROM osm.grass g
GROUP BY "type"
```

	A-z type	geom
1	heath	MULTIPOLYGON (((449140.67492309713 5590093.976330166,
2	scrub	MULTIPOLYGON (((466846.72121892346 5598677.852452918,
3	park	MULTIPOLYGON (((447670.33690541197 5592730.103953613,
4	grassland	MULTIPOLYGON (((447957.4420541764 5586232.081433068, 4
5	wood	MULTIPOLYGON (((478782.9663571899 5624301.131466, 478;
6	allotments	MULTIPOLYGON (((447582.72609747044 5594297.628909069,
7	grass	MULTIPOLYGON (((447626.813972804 5593076.778559768, 4
8	cemetery	MULTIPOLYGON (((447308.4658236758 5588539.705348146, 4
9	meadow	MULTIPOLYGON (((447147.5645740596 5586109.219438397, 4
10	village_green	MULTIPOLYGON (((472495.8679493349 5604481.265659397, 4
11	wetland	MULTIPOLYGON (((476562.61713729193 5606046.280242416,
12	recreation_grc	MULTIPOLYGON (((448174.34078499617 5613792.725339555,

## ST\_Collect()

- Sammeln der Geometrien ohne Veränderung
- Benötigt wenig Rechenzeit

```
SELECT "type", ST_Collect(geom) AS geom
FROM osm.grass g
GROUP BY "type"
```

	A-z type	geom
1	heath	GEOMETRYCOLLECTION (MULTIPOLYGON (((454656.7222041241
2	scrub	GEOMETRYCOLLECTION (MULTIPOLYGON (((466761.7267416235
3	park	GEOMETRYCOLLECTION (MULTIPOLYGON (((458522.03997099027
4	grassland	GEOMETRYCOLLECTION (MULTIPOLYGON (((458478.34814752743
5	wood	GEOMETRYCOLLECTION (MULTIPOLYGON (((478767.8330117924
6	allotments	GEOMETRYCOLLECTION (MULTIPOLYGON (((458504.4623699151
7	grass	GEOMETRYCOLLECTION (MULTIPOLYGON (((458584.685420511 5
8	cemetery	GEOMETRYCOLLECTION (MULTIPOLYGON (((458625.28198597935
9	meadow	GEOMETRYCOLLECTION (MULTIPOLYGON (((457840.3943330377
10	village_green	GEOMETRYCOLLECTION (MULTIPOLYGON (((487002.19088962703
11	wetland	GEOMETRYCOLLECTION (MULTIPOLYGON (((476519.866978374 5
12	recreation_ground	GEOMETRYCOLLECTION (MULTIPOLYGON (((456103.5706230714

# Anwendung von Set-Returning-Functions

- Auflösung von Multigeometrien

# Auflösung von Multigeometrien

- Auflösen von Multigeometrien in Einzelgeometrien als Beispiel
- Beispiel `ST_Dump()` Aus einer Geometrie mit mehreren Teilen in einer Zeile werden mehrere Zeilen und zwei Spalten mit `path` und `geom`

fid	name	st_dump
		path geom
991	[NULL]	> 1 POLYGON ((479709.4387223957 5604734.468010384, 479729.15155068913
992	[NULL]	> 1 POLYGON ((479715.0553783261 5603792.049102095, 479756.7470967936
993	[NULL]	> 1 POLYGON ((479715.6173731479 5603935.669286173, 479721.6068248832
994	[NULL]	> 1 POLYGON ((479868.28204881295 5610887.359229856, 479926.7014004072
995	[NULL]	> 1 POLYGON ((479189.2205443565 5609518.091538782, 479191.9224702051
996	[NULL]	> 1 POLYGON ((479776.7243854793 5608209.113683368, 479824.5252226993
997	[NULL]	> 1 POLYGON ((479621.8254084057 5608688.009642753, 479624.9952312438
998	[NULL]	> 1 POLYGON ((479547.94916700124 5611489.99516308, 479656.6731150998
1.015	[NULL]	> 1 POLYGON ((479523.37115523295 5623661.636838492, 479529.8722595252
999	[NULL]	> 1 POLYGON ((479284.6218087989 5614691.501179761, 479332.975995183
1.000	Mühlwald	> 1 POLYGON ((479584.3477191891 5613331.637841923, 479584.22969951347
1.001	[NULL]	> 1 POLYGON ((479767.3413467699 5611628.941833304, 479783.37814323086
1.002	[NULL]	> 1 POLYGON ((480220.6347861026 5612789.21348089, 480221.4834203429
1.003	[NULL]	> 1 POLYGON ((480120.7821362401 5613620.936190671, 480121.1476619751

```
-- ST_Dump als Set-Returning-Funktion
-- .geom greift auf die Spalte aus dem
-- Ergebnis von ST_Dump zu

SELECT fid, name, (ST_dump(geom)).geom
FROM osm.forest f
```

# Anwendung von Fensterfunktionen

- Entfernung von Duplikaten
- Räumliches Clustering

# Entfernung von Duplikaten

Kein räumliches Problem, aber oft benötigt:

```
-- Finden und Entfernen von Duplikaten
-- Beispiel: Abfrage von Bäckereien mit gleichem Namen
```

```
WITH
zaehlen AS
(
  SELECT fid, name, geom,
  row_number() over(PARTITION BY name) AS name_id
  FROM osm.poi
  WHERE shop = 'bakery'
)
SELECT fid, name, geom
FROM zaehlen
WHERE name_id < 2
```

	123 fid	A-z name	geom	123 name_id
6	137.658	Bäckerei Barth	POINT (484599.4950028013 5612917.573460091)	2
7	35.382	Bäckerei Bender	POINT (475894.90590100107 5602238.643973055)	1
8	78.487	Bäckerei Bierau	POINT (480206.2233145794 5616863.012568284)	1
9	78.488	Bäckerei Bierau	POINT (480208.51114308625 5616865.741249427)	2
10	8.696	Bäckerei Bölzer	POINT (450867.83229056525 5597829.220018)	1
11	40.636	Bäckerei Braun	POINT (475836.4284113894 5607799.058340555)	1
12	32.379	Bäckerei Braun	POINT (477115.5285972258 5603131.637123538)	2
13	32.378	Bäckerei Braun	POINT (477112.56336266216 5603122.754240264)	3
14	121.180	Bäckerei Dickel	POINT (486689.28361785685 5620330.566494028)	1
15	124.122	Bäckerei Dickel	POINT (487393.42365416995 5608327.397926537)	2
16	172.723	Bäckerei Drescher	POINT (468041.3891488685 5603878.901235447)	1
17	172.722	Bäckerei Drescher	POINT (468044.679355062 5603875.131589718)	2
18	120.387	Bäckerei Eckhardt	POINT (488286.9129089796 5621812.239043679)	1

Duplikate werden hochgezählt

	123 fid	A-z name	geom
34	118.970	Bäckerei Lukasch	POINT (488123.8545002105 5604999.74005725)
35	12.194	Bäckerei Milch	POINT (452622.3911872718 5611400.929549826)
36	173.704	Bäckerei Moos	POINT (467672.7167564713 5622842.765842241)
37	183.237	Bäckerei Moos GmbH	POINT (463726.46832624194 5603141.736844034)
38	198.900	Bäckerei Moos - Verkaufs	POINT (462141.80189701746 5603860.402848501)
39	85.660	Bäckerei Müller	POINT (481430.4909135901 5611826.829470234)
40	6.264	Bäckerei Pfister	POINT (456834.95508695854 5596410.13215212)
41	192.411	Bäckerei Redhardt	POINT (459326.71001214586 5600159.222948123)
42	255.885	Bäckerei Richter	POINT (502327.7373974789 5610511.959400178)
43	124.936	Bäckerei Röhm	POINT (487227.9964977561 5596504.36690126)
44	141.901	Bäckerei Seidl	POINT (474177.73371561285 5608360.923854067)
45	180.881	Bäckerei Steinmüller	POINT (464760.2155857746 5602392.005201658)
46	181.844	Bäckerei Steinmüller	POINT (464419.2668801582 5598028.767823395)

Nur der jeweils erste Name wurde ausgewählt

# Räumliches Clustering

- Anwendung einer Fensterfunktion zum Finden von räumlichen Gruppen
- Rückgabewert der Funktion ist eine Nummer des jeweiligen Clusters
- Beispielfunktion aus PostGIS:  
ST\_ClusterDBSCAN(geometrie, Abstand zwischen Punkten, minimale Clustergröße)
- Weitere Verarbeitung über die Nummern der Cluster möglich, wie zum Beispiel das Gruppieren und finden eines Schwerpunkts für jeden Cluster

```
-- Finden von räumlichen Clustern
```

```
SELECT fid, name, geom,  
ST_ClusterDBSCAN(geom, 20, 1) OVER() AS cluster_id  
FROM osm.poi  
WHERE shop = 'bakery'
```

	fid	name	geom	cluster_id
1	801	Schäfers Backstuben	POINT (457929.4209171566 5616447.011027505)	0
2	1.074	Hermann Redhardt	POINT (456849.4578085863 5605156.129031248)	1
3	2.110	[NULL]	POINT (456159.7256865764 5605801.6721369)	2
4	2.117	Bäckerei Künkel	POINT (456163.03303688584 5605819.73717656)	2
5	2.118	Bäckerei Künkel	POINT (456160.6739729091 5605814.2472236585)	2
6	2.506	[NULL]	POINT (456409.86632517475 5605609.7985781105)	3
7	2.556	Schäfers Backstuben	POINT (456244.05683140375 5605754.107382123)	4
8	2.557	Schäfers Backstuben	POINT (456244.2757515742 5605761.01126124)	4
9	2.614	Redhard	POINT (456271.0850680146 5605140.099465715)	5
10	4.019	Bäckerei und Cafe Schuppner & Koschare	POINT (454692.5088980669 5599619.47565175)	6
11	4.566	[NULL]	POINT (455440.99218775873 5586941.749205483)	7
12	4.931	Die Mühlenbäcker	POINT (456062.68896213965 5576302.618119211)	8
13	6.157	Brot- und Feinbäckerei Siegfried Pfister	POINT (456813.79065596376 5596060.901762701)	9

# Beispiel Räumliches Clustering

```
-- Komplettes Beispiel für finden von Clustern
-- und Brechnung eines Schwerpunkts für jeden Cluster

WITH
clustering AS (
  SELECT fid, name, geom,
  ST_ClusterDBSCAN(geom, 500, 1)
  OVER() AS cluster_id
  FROM osm.poi
  WHERE shop = 'bakery'
)
SELECT cluster_id, ST_Collect(geom) AS points,
ST_Centroid(ST_Collect(geom)) AS centroid
FROM clustering
GROUP BY cluster_id
```



Visualisierung der Punkte und der Clustermittelpunkte (große Punkte)

# Zusammenfassung Geodatenverarbeitung mit SQL

- PostGIS ermöglicht Verarbeitung von Geodaten in PostgreSQL-Datenbanken
- Erweiterung bekannter Funktionen um räumliche Pendanten
- Schwerpunkt liegt auf Vektordaten
- Vorteile von Räumlichen SQL zur Geodatenverarbeitung
  - Kombination aus GIS und SQL
  - Standardisierte Abfragesprache
  - Einfache Automatisierung von Aufgaben
  - Schnelle Verarbeitung großer Datenmengen



## Geodatenverarbeitung mit SQL


Interesse geweckt? – Fragen Sie gerne!




 Website  
<https://wherogroup.com>

 E-Mail  
[mathias.groebe@wherogroup.com](mailto:mathias.groebe@wherogroup.com)

 Telefon  
+49 305 1302 78 81

 Adresse  
WhereGroup GmbH  
Bundesallee 23  
10717 Berlin

 WhereGroup Shorts  
<https://wherogroup.com/shorts/>