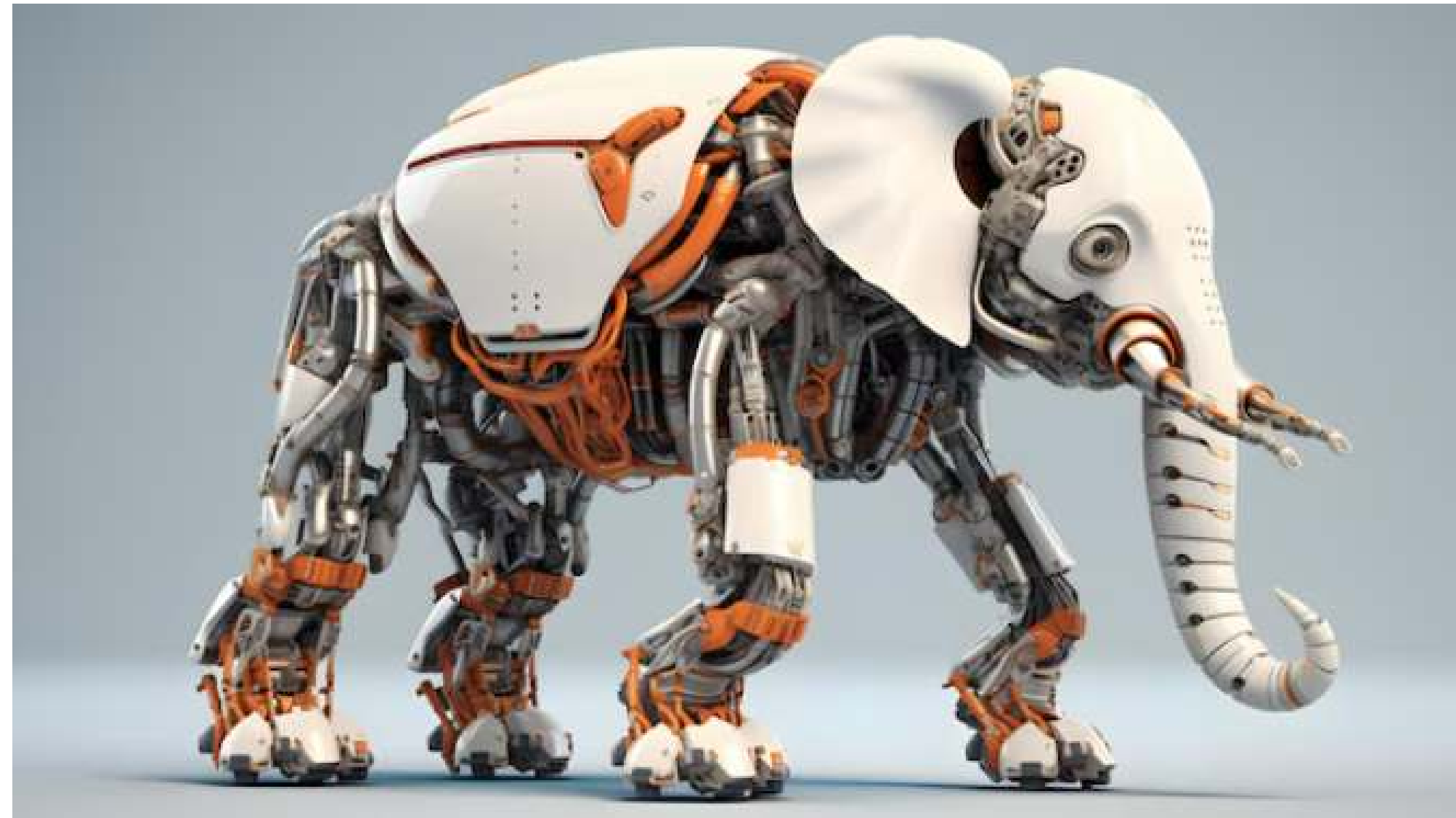


# PostGIS automatisieren

Jörg Thomsen, WhereGroup GmbH



# Themen

- Views
- Generated Columns
- Trigger / Triggerfunktionen
- pgAgent

# Views

```
-- Basis: country-Tabelle aus dem NaturalEarth-Daten

-- erst mal ein wenig übersichtlicher machen:
CREATE TABLE public.countries AS SELECT sovereignt, sov_a3, geom FROM public.ne_10m_admin_0_countries;
ALTER TABLE public.countries ADD COLUMN gid serial primary key;
CREATE INDEX IF NOT EXISTS countries_geom_idx ON public.countries USING gist (geom);
-- DROP TABLE public.countries;

-- #####
-- 1. Berechnung der Fläche der Länder --
-- #####

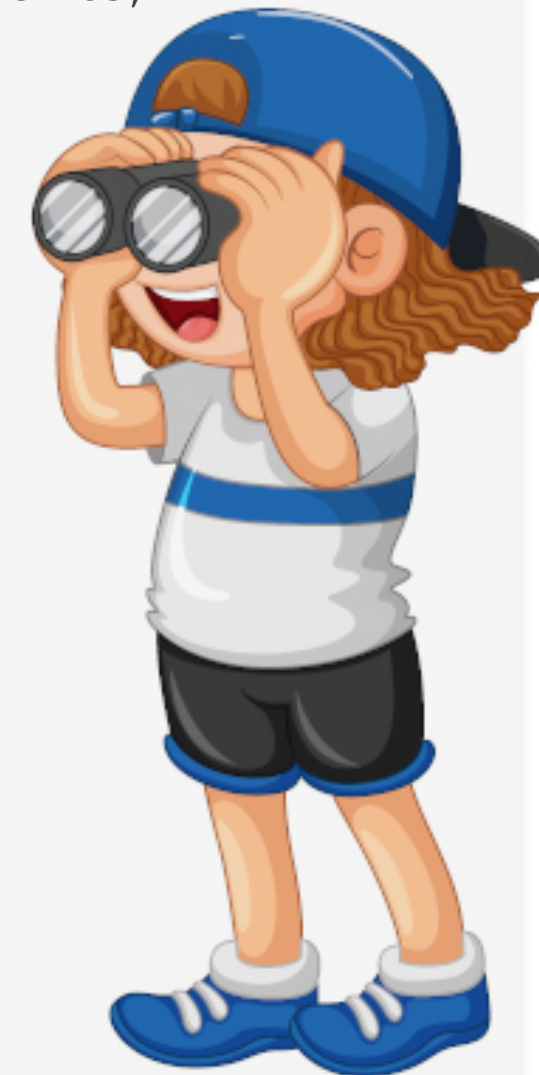
-- 1.1 in SQL --
-----

--EXPLAIN ANALYZE
SELECT gid, sovereignt, sov_a3, trunc(st_area(geom, true)/1000000) as area, geom
      FROM public.countries
-- Execution Time: 684.702 ms

-- 1.2 als view --
-----

CREATE VIEW public.v_countries_with_area AS
SELECT gid, sovereignt, sov_a3, trunc(st_area(geom, true)/1000000) as area, geom
      FROM public.countries;

--EXPLAIN ANALYZE
SELECT * FROM public.v_countries_with_area;
-- Execution Time: 674.531 ms
```



# Generated Column

```
-- 1.3 als generated column --
-----

/*
A virtual generated column is like a view, whereas a stored generated column is similar to a materialized view.
Unlike a material view, PostgreSQL automatically updates data for stored generated columns.
PostgreSQL currently implements only stored generated columns.
*/

ALTER TABLE public.countries
ADD COLUMN area integer GENERATED ALWAYS AS (trunc(st_area(geom, true)/1000000)) STORED;

-- EXPLAIN ANALYZE
SELECT gid, sovereignt, sov_a3, area, geom
      FROM public.countries;
-- Execution Time: 0.202 ms

INSERT INTO public.countries (sovereight, sov_a3, geom) VALUES ('WhereGroup', 'WG', '0106000020E610...876500C0');

-- EXPLAIN ANALYZE
SELECT gid, sovereignt, sov_a3, area, geom
      FROM public.countries;
-- "Execution Time: 0.278 ms"

-- gid | sovereignt | sov_a3 | area | geom
-- ----+-----+-----+-----+-----
-- 264 | WhereGroup | WG     | 585702 | 01060..
```

# Trigger und Funktionen

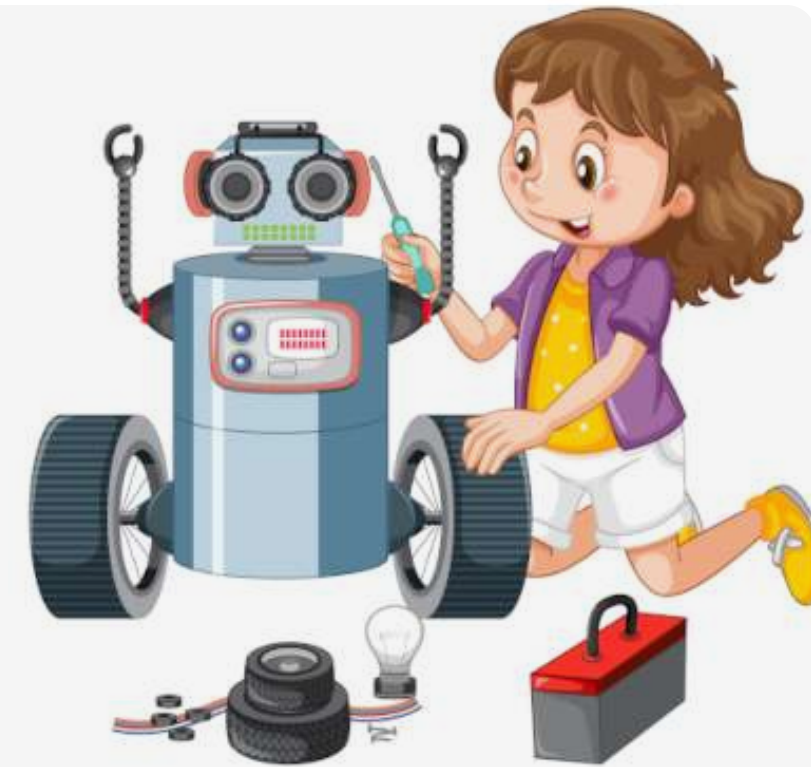
```
-- #####
-- 2. Trigger und Funktionen --
-- #####

-- aufräumen, löschen der generated column
ALTER TABLE public.countries
DROP COLUMN area;

ALTER TABLE public.countries
ADD COLUMN area integer;

-- 2.1 ein einfaches Beispiel --
-----

-- Trigger-Funktion, die die Flächen einer neu hinzugefügten oder geänderten
-- Geometrie berechnet:
CREATE OR REPLACE FUNCTION public.calculate_qkm()
    RETURNS TRIGGER
    LANGUAGE PLPGSQL
    AS
    $$
    BEGIN
        IF st_geometrytype(NEW.geom) in ('ST_MultiPolygon', 'ST_POLYGON') THEN
            NEW.area = trunc(st_area(NEW.geom, true)/1000000);
        ELSE
            RAISE NOTICE 'neue Geometrie ist kein Polygon';
        END IF;
        RETURN NEW;
    END;
    $$
```



```
-- der zugehörige Trigger:
CREATE TRIGGER trg_calculate_area
BEFORE INSERT OR UPDATE
ON public.countries
FOR EACH ROW
EXECUTE PROCEDURE public.calculate_qkm();

-- testen:
INSERT INTO public.countries (sovereignty, sov_a3, geom)
VALUES
('WhereGroup', 'WG', '0106000020E6100000020000000....100');

-- EXPLAIN ANALYZE
SELECT gid, sovereignty, sov_a3, area, geom FROM public.countries where sov_a3 = 'WG';
-- Execution Time: 0.186 ms

-- gid | sovereignty | sov_a3 | area | geom
-- ----+-----+-----+-----+-----
-- 264 | WhereGroup | WG     | 585702 | 01060..

-- etwas Performance:
CREATE INDEX IF NOT EXISTS countries_sov_a3_idx ON public.countries(sov_a3);
-- Wenn nach Größe selektiert werden soll, sollte auch auf der Spalte area ein index liegen
-- (hier und bei generated columns möglich, bei views nicht möglich)
-- -> Execution Time: 0.091 ms
```

```
-- 2.2 ein etwas komplexeres Beispiel --
-----
-- man muss nicht in einer Tabelle bleiben

ALTER TABLE ne_10m_populated_places ADD COLUMN countryname varchar;

CREATE TRIGGER placesOnInsertOrUpdateGetInfo
  BEFORE INSERT OR UPDATE
  ON ne_10m_populated_places
  FOR EACH ROW
  EXECUTE PROCEDURE placesGetAdditionalInfo();

-- DROP FUNCTION placesGetAdditionalInfo() CASCADE;
CREATE OR REPLACE FUNCTION placesGetAdditionalInfo()
  RETURNS trigger AS
  $BODY$
  DECLARE
    r RECORD;
  BEGIN

    SELECT c.name INTO r from ne_10m_admin_1_states_provinces c
    WHERE ST_Distance(c.geom , NEW.geom)= 0;

    RAISE NOTICE 'Operation: %' , TG_OP;
    RAISE NOTICE 'When: %' , TG_WHEN;
    RAISE NOTICE 'Countryname: %' , r.name;

    IF r.name IS NOT NULL THEN
      NEW.countryname = r.name;
    END IF;
    RETURN NEW;
  END
  $BODY$
LANGUAGE plpgsql VOLATILE COST 100;
```

```
-- testen:  
  
INSERT INTO ne_10m_populated_places (geom, name) VALUES (ST_GeomFromText('POINT(7.08 50.72)', 4326), 'Pusemuckel');  
  
-- Messages:  
-- HINWEIS: Operation: INSERT  
-- HINWEIS: When: BEFORE  
-- HINWEIS: Countryname: Nordrhein-Westfalen  
--  
-- Query returned successfully in 101 msec.  
  
-- Man kann Trigger deaktivieren:  
ALTER TABLE public.ne_10m_populated_places DISABLE TRIGGER placesoninsertorupdategetinfo;  
  
-- -> TG_OP etc: https://www.postgresql.org/docs/current/plpgsql-trigger.html  
-- -> event trigger: https://www.postgresql.org/docs/current/event-trigger-definition.html
```

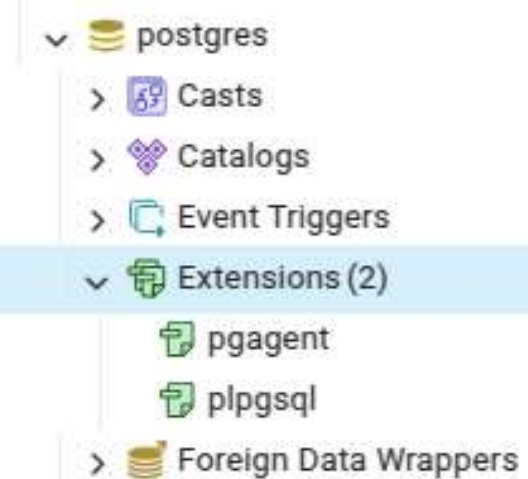
# pgAgent



# pgAgent

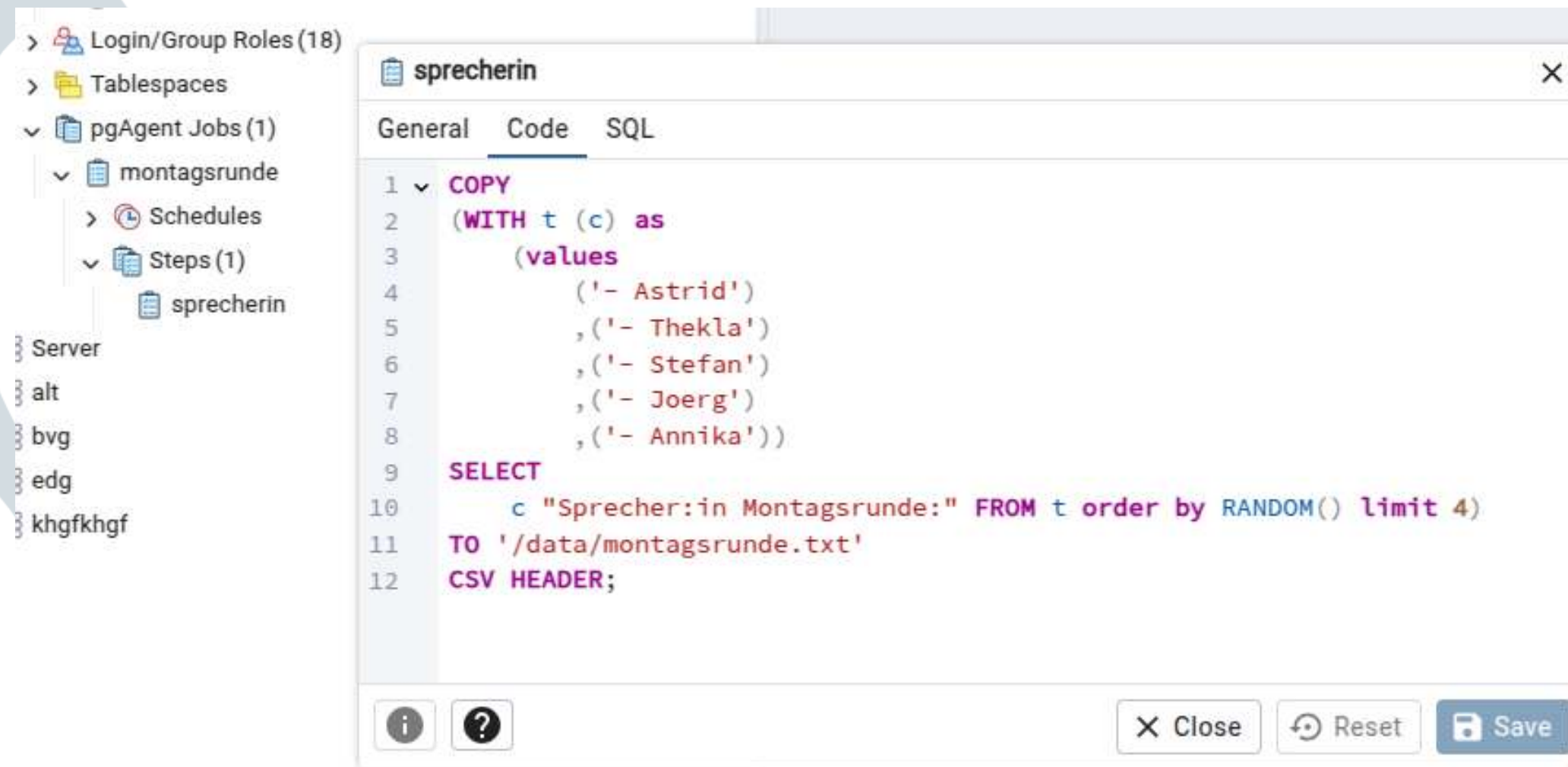
Eine Erweiterung für zeitgesteuerte Aktionen.

Die Erweiterung muss in die Maintenance-DB (meist 'postgres') geladen werden und kann über pgAdmin und DBeaver konfiguriert werden.



# pgAgent in pgAdmin

'Steps' -> auszuführender Code



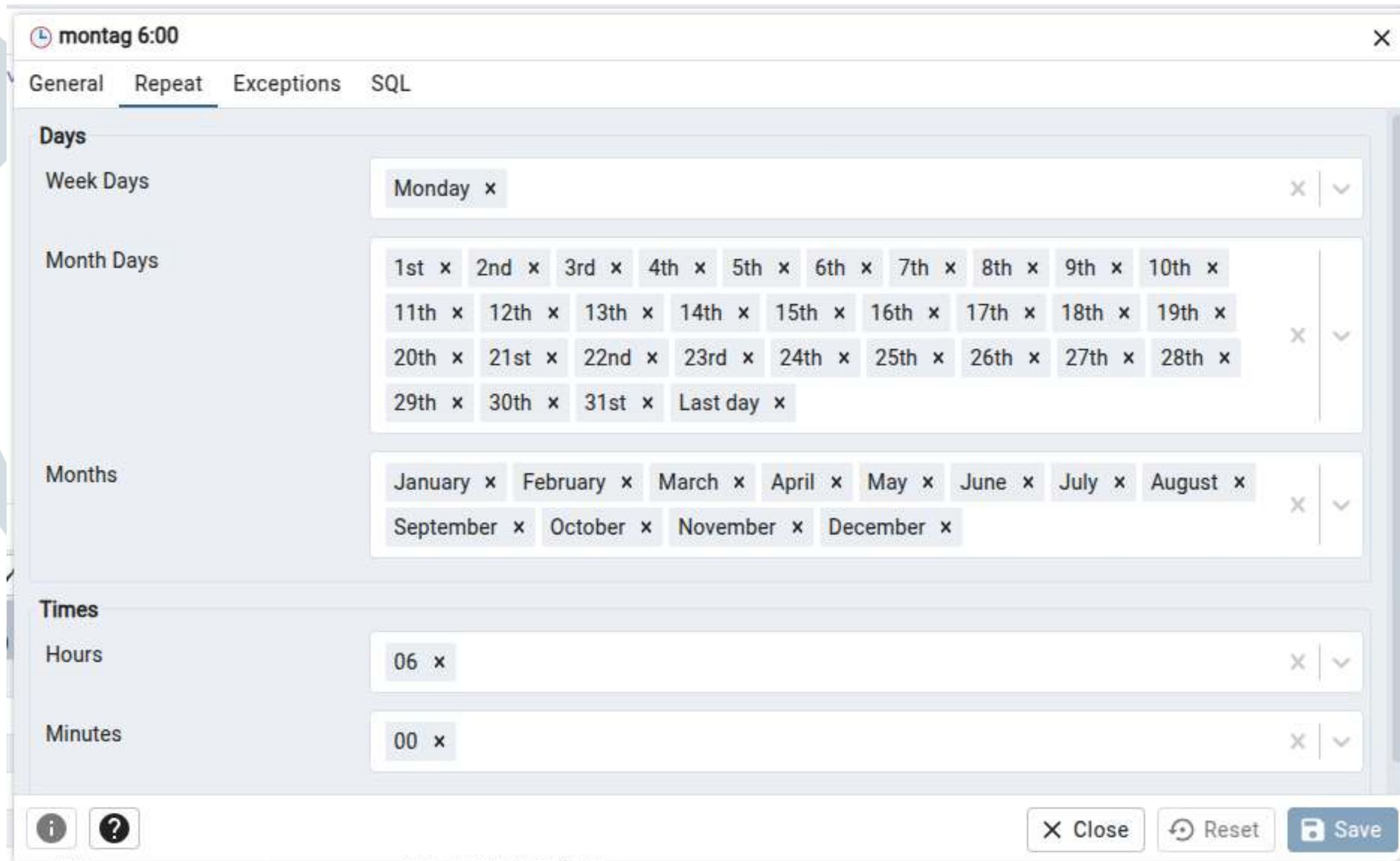
The screenshot shows the pgAdmin interface. On the left, a tree view shows the database structure: Login/Group Roles (18), Tablespaces, pgAgent Jobs (1), montagsrunde, Schedules, and Steps (1). The 'Steps (1)' folder is expanded, showing a step named 'sprecherin'. The main window displays the SQL code for this step:

```
1 COPY
2 (WITH t (c) as
3     (values
4         ('- Astrid')
5         , ('- Thekla')
6         , ('- Stefan')
7         , ('- Joerg')
8         , ('- Annika'))
9 SELECT
10     c "Sprecher:in Montagsrunde:" FROM t order by RANDOM() limit 4)
11 TO '/data/montagsrunde.txt'
12 CSV HEADER;
```

At the bottom of the window, there are buttons for 'Close', 'Reset', and 'Save'.

# pgAgent in pgAdmin

'Schedule' -> Zeitplanung



montag 6:00

General Repeat Exceptions SQL

**Days**

Week Days: Monday

Month Days: 1st, 2nd, 3rd, 4th, 5th, 6th, 7th, 8th, 9th, 10th, 11th, 12th, 13th, 14th, 15th, 16th, 17th, 18th, 19th, 20th, 21st, 22nd, 23rd, 24th, 25th, 26th, 27th, 28th, 29th, 30th, 31st, Last day

Months: January, February, March, April, May, June, July, August, September, October, November, December

**Times**

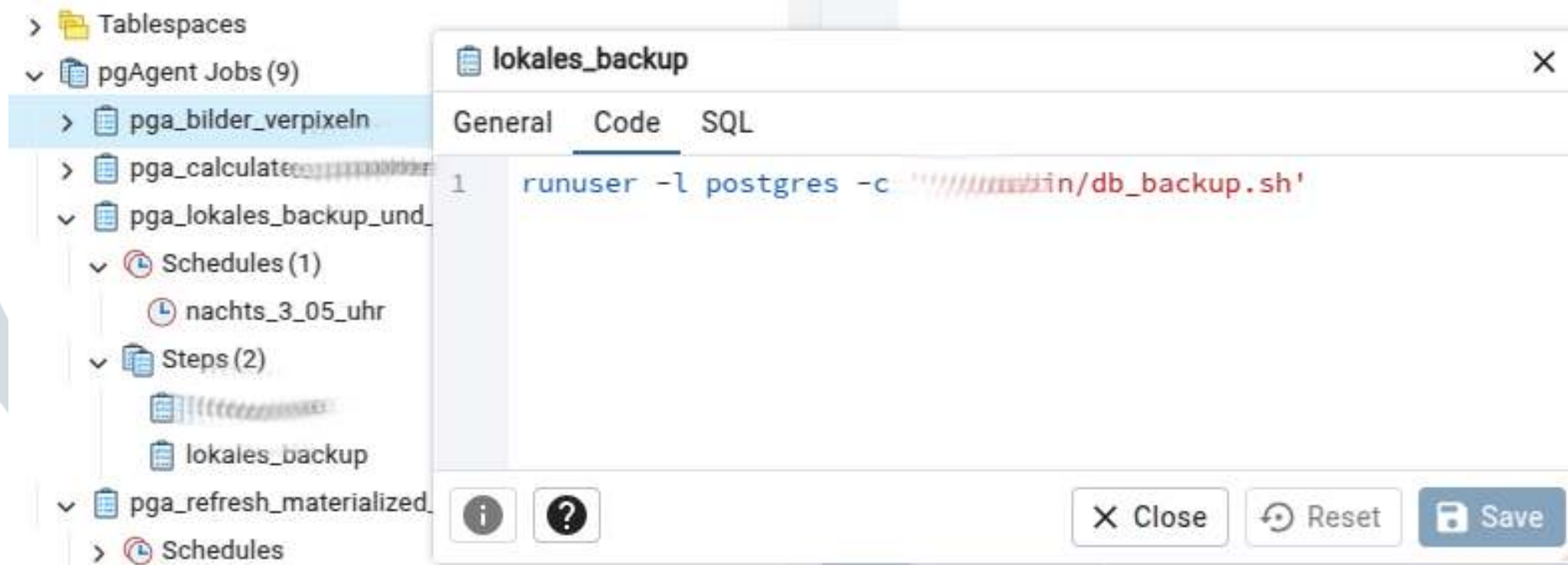
Hours: 06

Minutes: 00

Close Reset Save

# pgAgent in pgAdmin

Aufruf externer Scripts:



The screenshot shows the pgAdmin interface with a tree view on the left and a configuration window for a job named 'lokales\_backup' on the right. The tree view shows the following structure:

- > Tablespaces
- ✓ pgAgent Jobs (9)
  - > pga\_bilder\_verpixeln
  - > pga\_calculate...
  - ✓ pga\_lokales\_backup\_und...
    - ✓ Schedules (1)
      - ↳ nachts\_3\_05\_uhr
    - ✓ Steps (2)
      - ↳ [redacted]
      - ↳ lokales\_backup
  - ✓ pga\_refresh\_materialized...
    - > Schedules

The configuration window for 'lokales\_backup' has the following tabs: General, Code, SQL. The 'Code' tab is active, showing the following command:

```
1 runuser -l postgres -c '#####in/db_backup.sh'
```

At the bottom of the window, there are buttons for 'Close', 'Reset', and 'Save'.

```
#!/bin/bash

# Listen durch pipe separiert angeben:
rotate_list="ne2|freiburg|mapbender"
ignore_list="template0|template1|postgres|template_postgis"
...

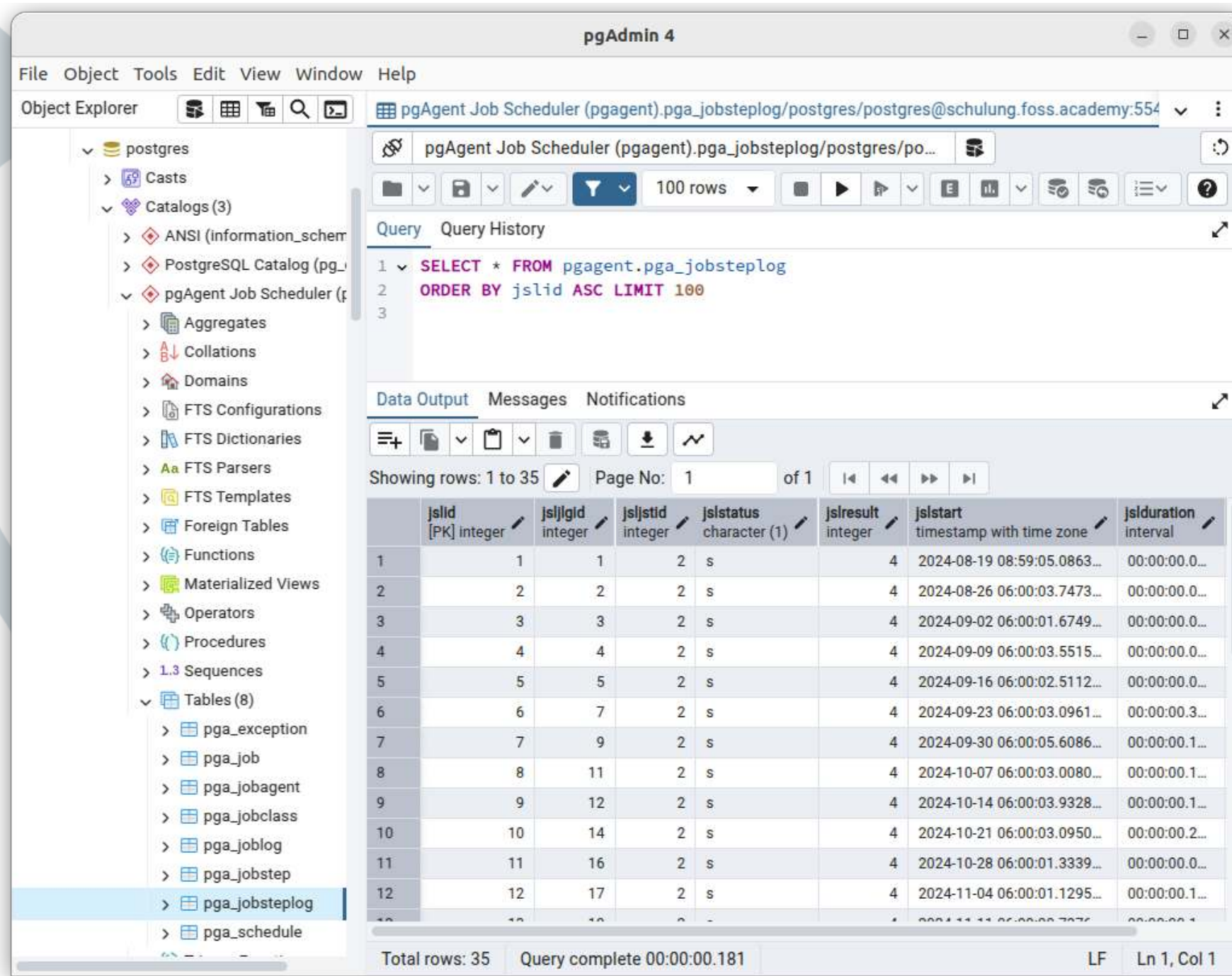
# Alle Datenbanken ermitteln
db_list=$(psql -U postgres -t -c "select datname from pg_database")

for current_db in $db_list; do
...
    /usr/bin/pg_dump -U postgres --format=p $current_db | gzip -c > $dmp_file
...
done

echo "Dumps fertig: $(date +"%a, %d %b %Y %H:%M:%S")" >> $log_file
```

# pgAgent in pgAdmin

Tabellen und LOGs



The screenshot shows the pgAdmin 4 interface. The left pane displays the Object Explorer with the 'pgAgent Job Scheduler (pgagent)' database selected. The 'pgAgent Job Scheduler (pgagent)' database is expanded to show the 'pgAgent Job Scheduler (pgagent)' schema, which contains several tables, including 'pga\_jobsteplog'. The main pane shows a query window with the following SQL query:

```
1 SELECT * FROM pgagent.pga_jobsteplog
2 ORDER BY jslid ASC LIMIT 100
3
```

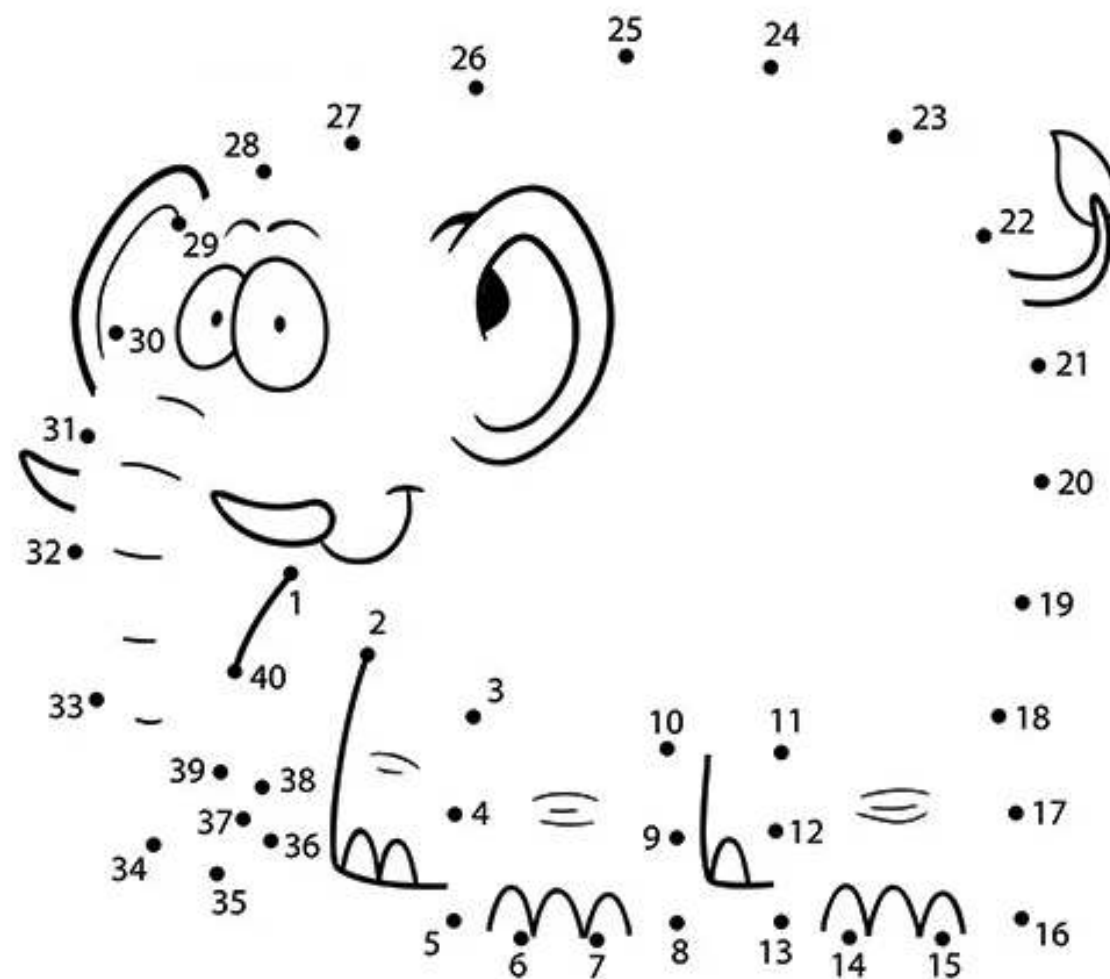
The query results are displayed in a table with the following columns: jslid [PK] integer, jsljgid integer, jsljtid integer, jslstatus character (1), jsresult integer, jsistart timestamp with time zone, and jslduration interval. The table shows 12 rows of data, with the first row having jslid 1 and jsistart 2024-08-19 08:59:05.0863... The status for all rows is 's'.

jslid [PK] integer	jsljgid integer	jsljtid integer	jslstatus character (1)	jsresult integer	jsistart timestamp with time zone	jslduration interval
1	1	1	s	4	2024-08-19 08:59:05.0863...	00:00:00.0...
2	2	2	s	4	2024-08-26 06:00:03.7473...	00:00:00.0...
3	3	3	s	4	2024-09-02 06:00:01.6749...	00:00:00.0...
4	4	4	s	4	2024-09-09 06:00:03.5515...	00:00:00.0...
5	5	5	s	4	2024-09-16 06:00:02.5112...	00:00:00.0...
6	6	7	s	4	2024-09-23 06:00:03.0961...	00:00:00.3...
7	7	9	s	4	2024-09-30 06:00:05.6086...	00:00:00.1...
8	8	11	s	4	2024-10-07 06:00:03.0080...	00:00:00.1...
9	9	12	s	4	2024-10-14 06:00:03.9328...	00:00:00.1...
10	10	14	s	4	2024-10-21 06:00:03.0950...	00:00:00.2...
11	11	16	s	4	2024-10-28 06:00:01.3339...	00:00:00.0...
12	12	17	s	4	2024-11-04 06:00:01.1295...	00:00:00.1...

The status bar at the bottom indicates 'Total rows: 35', 'Query complete 00:00:00.181', and 'LF Ln 1, Col 1'.


# PostGIS automatisieren

Interesse geweckt? – Fragen Sie gerne!



 Webseite

<https://wherogroup.com>

 E-Mail

[joerg.thomsen@wherogroup.com](mailto:joerg.thomsen@wherogroup.com)

 Telefon

+49 228 / 90 90 38 - 0

 Adresse

WhereGroup GmbH

Bundesallee 23

10717 Berlin

 WhereGroup Shorts

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