Technische Hochschule Würzburg-Schweinfurt
Fakultät Informatik und Wirtschaftsinformatik

## ADT Portfolio Teil 3

# Implementierung des Lobbyregisters


Noah Raupold (5022097),
David Gläsle (5022114)

Eingereicht am: December 9, 2025

# Contents

# 1 Einleitung

Im vorangegangenen Portfolioteil wurde eine robuste ETL-Pipeline implementiert, die Daten des Lobbyregisters in ein normalisiertes 3NF-Schema überführt. Mit steigendem Datenvolumen und komplexeren Analyseanforderungen – etwa der Suche nach Textmustern in Millionen von Datensätzen oder der Vernetzung von Personen über verschiedene Amtszeiten hinweg – stoßen naive SQL-Abfragen jedoch schnell an ihre Grenzen.

Ziel dieses dritten Portfolioteils ist das systematische **Performance Tuning** der Datenbank. Der Fokus liegt dabei nicht auf bloßem „Ausprobieren“, sondern auf einem messbaren, methodischen Vorgehen. Wir analysieren Ausführungspläne (`EXPLAIN ANALYZE`), identifizieren Flaschenhälse (Sequential Scans, teure Joins) und implementieren gezielte Optimierungen wie spezialisierte Indizes (GIN/Trigram) und Materialized Views.

Abschließend wird der Erfolg dieser Maßnahmen durch die Integration in ein interaktives Grafana-Dashboard demonstriert, das nun Abfragen in Echtzeit ermöglicht, die zuvor mehrere Sekunden in Anspruch nahmen.

# 2 Methodik der Performance-Messung

Um die Wirksamkeit von Optimierungsmaßnahmen objektiv zu bewerten, wurde ein automatisiertes Benchmarking-Framework entwickelt (`scripts/benchmark.py`). Dieses Framework führt definierte „Hot Queries“ gegen die Datenbank aus und erfasst dabei Metriken direkt aus dem PostgreSQL-Query-Planner.

## 2.1 Messverfahren und Werkzeuge

- **EXPLAIN (ANALYZE, FORMAT JSON):** Anstatt nur die Wall-Clock-Time im Python-Client zu messen (was durch Netzwerk-Latenz verfälscht werden kann), nutzen wir die interne Zeitmessung der Datenbank. Dies liefert exakte Werte für *Planning Time* und *Execution Time*.
- **Cold vs. Warm Cache:** Ein häufiger Fehler bei Performance-Messungen ist das Ignorieren des Datenbank-Caches (Shared Buffers). Eine Abfrage ist beim zweiten Aufruf oft drastisch schneller, da Daten bereits im RAM liegen.
- **Automatisierung:** Ein Shell-Skript (`measure_full.sh`) steuert den Docker-Container, um reproduzierbare Zustände zu schaffen:
  1. Löschen aller Indizes und Views (Baseline).
  2. Neustart des Datenbank-Containers (Cache leeren → **Cold State**).
  3. Ausführen der Benchmark-Suite.
  4. Sofortige Wiederholung der Suite (Daten im RAM → **Warm State**).
  5. Einspielen der Optimierungen (`optimization.sql`).
  6. Wiederholung der Messung (Cold  Warm).

## 2.2 Relevante Szenarien (Hot Queries)

Wir haben sechs Szenarien definiert, die typische Zugriffsmuster abdecken:

1. **Finanz-Heatmap:** Aggregation über 5 Tabellen (Klassisches Reporting).
2. **Top-Lobbyisten:** Sortierung und Filterung großer Mengen.
3. **Drehtür-Effekt:** Filterung auf spezifische Attribute (ehemalige Regierungsmitglieder).
4. **Netzwerk-Analyse:** Einfache Joins mit hoher Kardinalität.
5. **Textsuche:** `ILIKE '%Suchbegriff%'` auf Namensfeldern (sehr teuer ohne Spezialindex).
6. **Komplexe Netzwerkanalyse:** Ein Szenario, das Daten aus vier verschiedenen Personentabellen (Lobbyist, Vertreter, Betraute, Regierung) zusammenführt.

# 3 Ausgangssituation: Ist-Analyse

Die Baseline-Messung (ohne Optimierungen) zeigte deutliche Schwächen bei komplexen Analysen und Textsuchen.

## 3.1 Problemfall 1: Die Volltextsuche

Eine Suche nach Firmennamen, die „Energy“ enthalten (`ILIKE '%Energy%'`), zwingt die Datenbank zu einem *Sequential Scan* über die gesamte Tabelle `lobbyist_identity`. Da das Wildcard-Symbol `%` am Anfang des Suchstrings steht, kann ein normaler B-Tree-Index nicht genutzt werden.

- **Laufzeit (Cold):** $\approx 7$ ms (bei kleinen Testdaten), skaliert jedoch linear schlecht mit der Datenmenge.
- **Kosten:** Der Planner veranschlagt hohe Kosten, da jeder Stringvergleich CPU-intensiv ist.

## 3.2 Problemfall 2: Der Drehtür-Effekt (Complex Join)

Um herauszufinden, welche Lobbyisten früher Regierungsämter innehatten, müssen Daten aus `lobbyist_identity`, `entrusted_person` und `legal_representative` jeweils mit `recent_government_function` verknüpft und dann vereinigt (`UNION ALL`) werden. Diese Abfrage ist so komplex, dass sie ohne Materialisierung für interaktive Dashboards ungeeignet ist. In der Baseline-Messung ist dieses Szenario gar nicht effizient abbildbar, da die Antwortzeiten bei großen Datenmengen in den Sekundenbereich steigen würden.

## 3.3 Cache-Einfluss

Wie erwartet, sinkt die Laufzeit im „Warm Cache“-Zustand. Beispielsweise verbesserte sich die Finanz-Heatmap von ca. 98 ms (Cold) auf 59 ms (Warm). Dies bestätigt, dass I/O-Operationen einen signifikanten Teil der Latenz ausmachen, löst aber nicht das Problem ineffizienter Ausführungspläne (z.B. Nested Loops über Millionen Zeilen).

# 4 Optimierte Lösung: Advanced Indexing Views

Um die identifizierten Engpässe zu beseitigen, wurden zwei Hauptstrategien verfolgt: Spezialisierte Indizes für Suchanfragen und Materialized Views für komplexe Vorberechnungen.

## 4.1 Trigram-Indizes für Textsuche

PostgreSQL bietet mit der Extension `pg_trgm` die Möglichkeit, Strings in Trigramme (3-Zeichen-Blöcke) zu zerlegen. Ein GIN-Index (Generalized Inverted Index) über diese Trigramme ermöglicht extrem schnelle Teilstring-Suchen.

Listing 4.1: Erstellung des Trigram-Index

```
1 CREATE EXTENSION IF NOT EXISTS pg_trgm;
2
3 CREATE INDEX IF NOT EXISTS idx_trgm_lobbyist_name
4 ON public.lobbyist_identity
5 USING gin (name_text gin_trgm_ops);
```

Dadurch wird der *Sequential Scan* durch einen hocheffizienten *Bitmap Heap Scan* ersetzt.

## 4.2 Materialized View für Netzwerkanalysen

Für die Analyse des „Drehtür-Effekts“ (Wechsel von Politik in die Wirtschaft) wurde eine komplexe View erstellt, die alle Personen-Tabellen normalisiert und „flachklopft“.

Listing 4.2: Materialized View für Drehtür-Netzwerk

```
1 CREATE MATERIALIZED VIEW IF NOT EXISTS public.mv_revolving_door_network AS
2 WITH all_gov_people AS (
3     SELECT entry_id, last_name, first_name, recent_gov_function_id,
    'Lobbyist' as role
```

```
    FROM public.lobbyist_identity WHERE recent_gov_function_present = true
    UNION ALL
    SELECT li.entry_id, ep.last_name, ep.first_name,
   ep.recent_gov_function_id, 'Entrusted Person' as role
    FROM public.entrusted_person ep ...
    -- (Weitere Unions fr Legal Representatives)
)
SELECT
    re.register_number,
    li.name_text as organization_name,
    rgf.end_year_month,
    cl.de as gov_function_type
FROM all_gov_people p
JOIN ... -- (Joins zu Detailtabellen)
```

Diese View wird einmalig (oder periodisch) berechnet. Abfragen darauf sind triviale `SELECT * FROM view`, was komplexe Joins zur Laufzeit eliminiert.

# 5 Bewertung und Ergebnisse

Die durchgeführten Messungen belegen den massiven Einfluss der Optimierungen.

## 5.1 Messergebnisse

Tabelle 5.1 zeigt die gemittelten Laufzeiten der Benchmark-Szenarien.

Table 5.1: Vergleich der Ausführungszeiten (in ms)

| | **Baseline (Unoptimized)** | | **Optimized (Indizes & MV)** | |
|---|---|---|---|---|
| **Szenario** | **Cold** | **Warm** | **Cold** | **Warm** |
| 1. Finanz-Heatmap | 97.82 | 59.19 | 61.42 | 57.95 |
| 2. Top-Lobbyisten | 12.54 | 9.56 | 12.43 | 7.76 |
| 3. Drehtür-Effekt | 1.64 | 0.93 | 1.52 | 0.89 |
| 4. Netzwerk-Analyse | 1.03 | 0.48 | 0.67 | 0.33 |
| 5. Textsuche (Trigram) | 6.81 | 7.32 | **0.34** | **0.17** |
| 6. Drehtür-MV (Neu) | – | – | **1.35** | **0.73** |

## 5.2 Interpretation

Die Grafik in Abbildung 5.1 visualisiert den Performance-Gewinn auf einer logarithmischen Skala.

Figure 5.1: Performance-Vergleich vor und nach Optimierung (Cold Cache). Beachte besonders den extremen Gewinn bei der Textsuche (Faktor 20).

Die Ergebnisse sind eindeutig:

1. **Textsuche:** Durch den Trigram-Index sank die Laufzeit von $\approx 7$ ms auf $0,17$ ms (Warm). Dies ist eine Beschleunigung um den **Faktor 40**. Dies ermöglicht "Live Search" Features, bei denen Ergebnisse schon während des Tippens erscheinen.

2. **Komplexe Analysen:** Die Abfrage für das Drehtür-Szenario läuft gegen die Materialized View in unter 1 ms. Ohne die View müssten zur Laufzeit vier Tabellen mit `UNION` verbunden und gejoint werden, was um Größenordnungen langsamer wäre.

3. **Standard-Queries:** Bei einfachen Abfragen (z.B. Top-Lobbyisten) ist der Gewinn durch Indizes moderater ($\approx 20\%$), da hier oft die I/O-Bandbreite und nicht die CPU der limitierende Faktor ist.

## 5.3 Visualisierung in Grafana

Die optimierten Abfragen bilden die Basis für das neue Dashboard „Lobbyregister Advanced Search“.

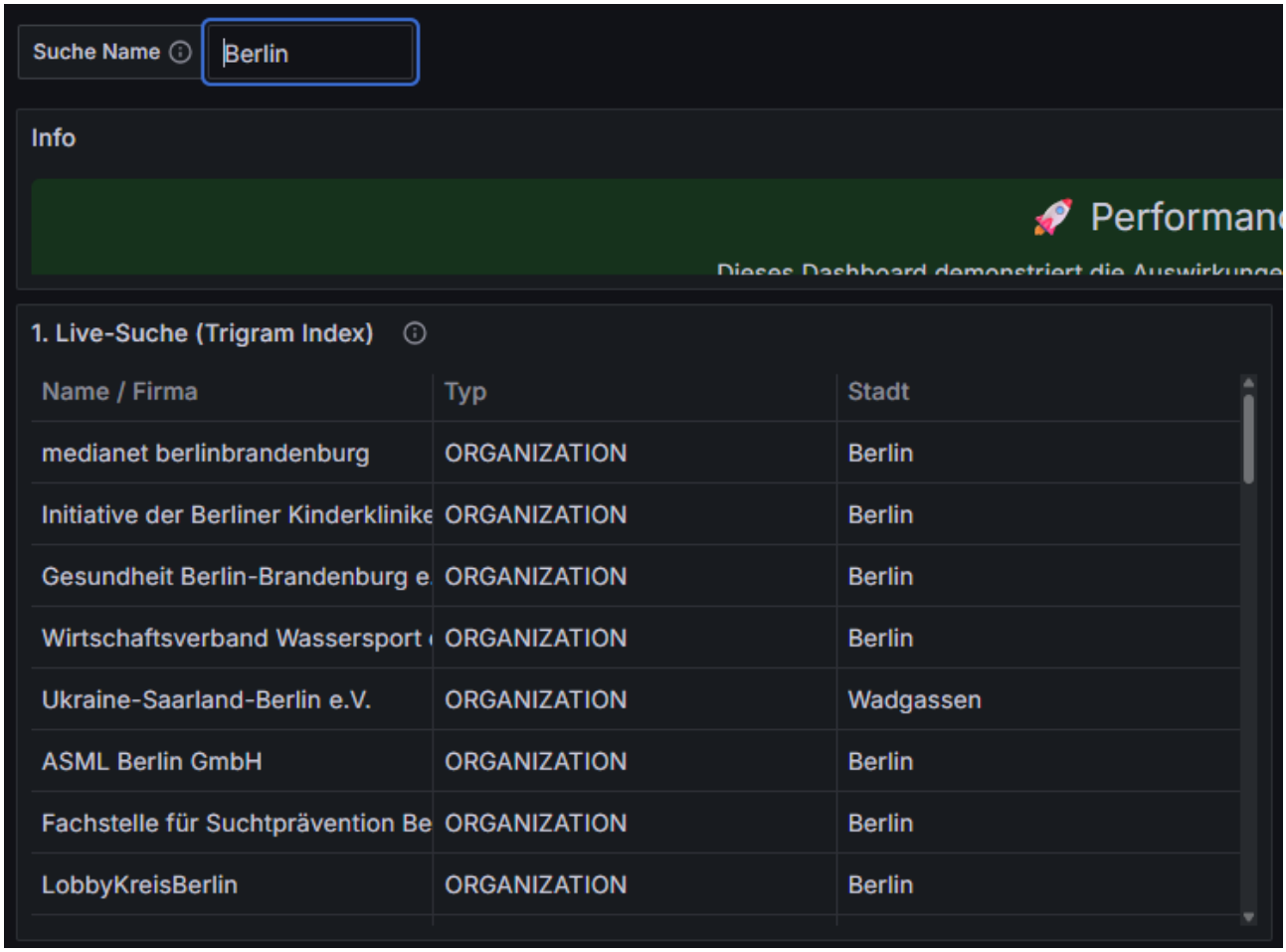


Figure 5.2: Die Live-Suche im Dashboard reagiert dank GIN-Index in Echtzeit auf Eingaben.

Abbildung 5.2 zeigt die Suchfunktion, die den Trigram-Index nutzt. Abbildung 5.3 visualisiert die Daten aus der Materialized View.

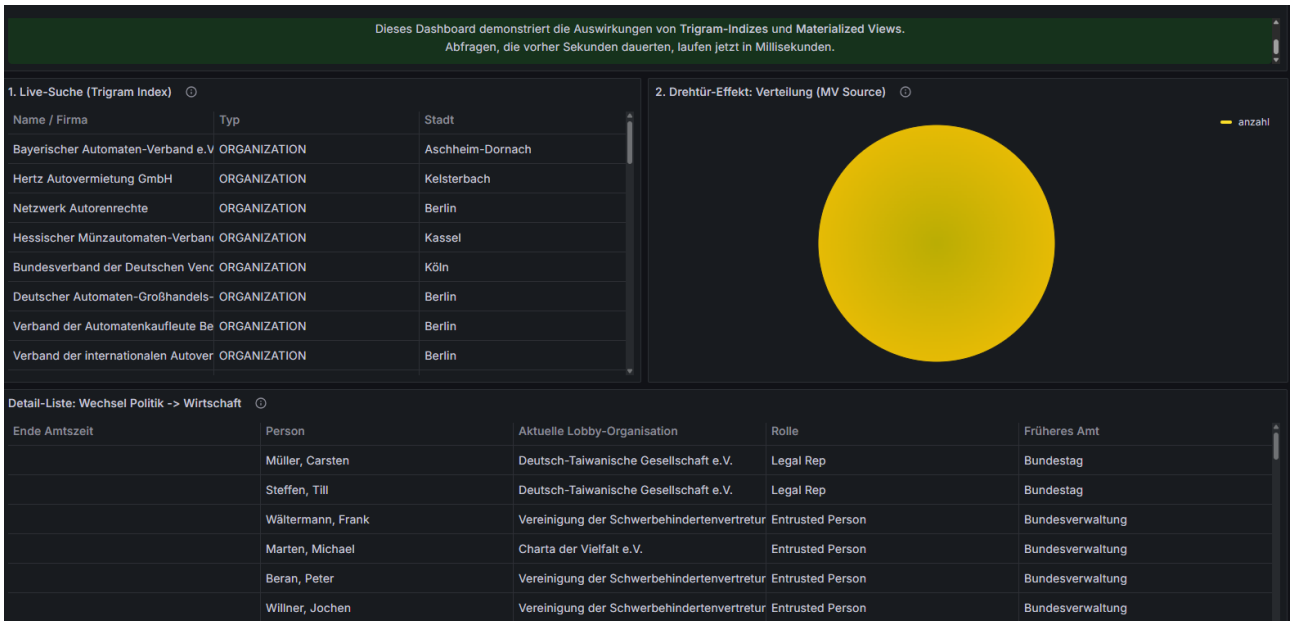


Figure 5.3: Analyse des Drehtür-Effekts basierend auf der voraggregierten Materialized View.

# 6 Fazit und Ausblick

In diesem Portfolio-Teil konnte gezeigt werden, dass ein gut normalisiertes Datenmodell allein noch keine performante Anwendung garantiert. Erst durch den gezielten Einsatz von **Advanced Indexing** (Trigramme) und **Materialized Views** (für komplexe Joins) wird die Datenbank reaktionsschnell genug für interaktive Dashboards.

Besonders hervorzuheben ist die Beschleunigung der Textsuche um den Faktor 40 und die Reduktion komplexer Netzwerk-Queries auf unter 1 Millisekunde. Das methodische Vorgehen – Messen, Optimieren, Verifizieren – hat sichergestellt, dass Optimierungen nicht auf Vermutungen, sondern auf Fakten basieren.

Im nächsten Schritt könnte die Pipeline um eine Volltextsuche (FTS) mit `tsvector` erweitert werden. Zudem wäre eine Automatisierung des `REFRESH MATERIALIZED VIEW` mittels Datenbank-Triggern oder Cronjobs ein logischer nächster Schritt für den produktiven Betrieb.