Anfrageoptimierung im RDF-Kontext

Samuel Jordan Ouabo

3. August 2025

Inhaltsverzeichnis

1 Einleitung

Das World Wide Web hat sich in den letzten Jahrzehnten grundlegend gewandelt: Von einer rein hypertextbasierten Dokumentensammlung hin zu einem semantisch vernetzten Informationsraum, der nicht nur für Menschen, sondern auch für Maschinen interpretierbar ist. Diese Entwicklung mündet in der Vision des *Semantic Web*, wie sie von Berners-Lee formuliert wurde – ein Web, in dem Daten nicht nur dargestellt, sondern auch verarbeitet und logisch verknüpft werden können [2].

Ein zentrales Element des Semantic Web ist das *Resource Description Framework (RDF)*, das Daten in Form von sogenannten Tripeln repräsentiert. Jedes Tripel besteht aus Subjekt, Prädikat und Objekt und bildet damit eine atomare Aussage über eine Ressource. In der Praxis entstehen durch RDF große, heterogene Graphstrukturen – sogenannte Wissensgraphen –, die in Bereichen wie Linked Data, Bioinformatik, maschinelles Lernen oder öffentliche Verwaltung eingesetzt werden [6]. Zur Abfrage solcher Graphen dient die standardisierte Sprache *SPARQL*, die auf dem Prinzip des *Graph Pattern Matching* basiert.

Mit der wachsenden Verbreitung von RDF-basierten Datenbanken (auch als Triple Stores bezeichnet) stellt sich zunehmend die Frage, wie solche Systeme performant und skalierbar Anfragen verarbeiten können. Anders als relationale Datenbanksysteme verfügen RDF-Stores jedoch über keine feste Schemadefinition, sondern arbeiten unter der *Open World Assumption* (OWA). Diese Eigenschaften erschweren die Anwendung klassischer Optimierungskonzepte erheblich: Typische Verfahren wie Kostenmodelle, Kardinalitätsschätzungen oder regelbasierte Transformationsstrategien stoßen im RDF-Kontext häufig an ihre Grenzen [14, 16].

Die klassische Datenbankforschung – etwa wie sie von Kemper und Eickler beschrieben wurde – hat über Jahrzehnte eine Vielzahl bewährter Methoden für die relationale Anfrageverarbeitung entwickelt [7]. Zu diesen zählen algebraische Anfragepläne, dynamische Join-Reihenfolgenoptimierung und komplexe Indexierungsstrategien. Diese Techniken bilden einen wichtigen Referenzrahmen für die Frage, inwiefern und unter welchen Bedingungen sie sich auch im semantischen Kontext übertragen oder adaptieren lassen.

Ziel dieser Seminararbeit ist es, einen systematischen Überblick über Optimierungsstrategien im RDF-Kontext zu geben. Dazu werden zunächst die Grundlagen der relationalen Anfrageverarbeitung (Kapitel **??**) vorgestellt und mit dem RDF-Datenmodell und SPARQL verglichen (Kapitel 3). Dar-

auf aufbauend werden die spezifischen Optimierungsprobleme und -ansätze in RDF-Datenbanken analysiert (Kapitel 4 und 5). Abschließend erfolgt eine vergleichende Einordnung klassischer und semantischer Optimierungsprinzipien (Kapitel **??**) sowie eine kritische Diskussion aktueller Limitationen und zukünftiger Forschungsfragen (Kapitel **??** und **??**).

2 Grundlagen relationaler Anfrageverarbeitung

Die relationale Anfrageverarbeitung bildet das Rückgrat klassischer Datenbanksysteme. Ziel ist es, Anfragen – typischerweise in der Sprache SQL formuliert – unabhängig von der physischen Speicherung effizient auszuführen. Grundlage hierfür ist eine mehrstufige Verarbeitungspipeline, die unter anderem durch Kemper und Eickler systematisch dargestellt wird [7].

2.1 Anfragebearbeitungsprozess

Die Bearbeitung einer SQL-Anfrage erfolgt üblicherweise in drei Hauptphasen:

1. **Anfrageübersetzung:** Die SQL-Anfrage wird in einen sogenannten *algebraischen Anfragebaum* übersetzt. Dieser dient als interne Repräsentation und basiert auf Operatoren der relationalen Algebra (z. B. σ für Selektion, π für Projektion, $\bowtie$ für Join).

2. **Anfrageoptimierung:** Der Anfragebaum wird durch Transformationen in eine äquivalente, aber effizienter auszuführende Variante überführt. Dies kann regelbasiert (durch algebraische Umformungsregeln) oder kostenbasiert (unter Zuhilfenahme eines Kostenmodells) erfolgen.

3. **Anfrageausführung:** Der optimierte Anfrageplan wird anschließend vom DBMS durch konkrete Zugriffe auf Tabellen, Indizes und Pufferspeicherstrukturen ausgeführt.

2.2 Relationale Algebra

Die relationale Algebra stellt eine formale Grundlage für Anfragen dar. Sie umfasst zentrale Operatoren:

- **Selektion (σ):** Filtert Tupel auf Basis eines Prädikats, z. B. $\sigma_{note>2.0}$(Klausuren).
- **Projektion (π):** Reduziert die Attributmenge, z. B. $\pi_{\text{name, note}}$(Klausuren).
- **Join ($\bowtie$):** Verknüpft zwei Relationen über gemeinsame Attribute, z. B. Studenten $\bowtie_{\text{matrNr}}$ Klausuren.
- **Mengenoperationen:** Union, Differenz und kartesisches Produkt.

Die Kombination dieser Operatoren in einem Baum hat großen Einfluss auf die Effizienz – insbesondere, wenn redundante Zwischenresultate vermieden werden.

2.3 Optimierungsstrategien

Ziel der Anfrageoptimierung ist es, einen möglichst kostengünstigen Ausführungsplan zu generieren. Dazu kommen zwei Hauptstrategien zum Einsatz:

- **Regelbasierte Optimierung:** Transformation des Anfragebaums anhand äquivalenter Umschreibungen. Beispiele sind das Vorziehen selektiver Operationen ("Selektion vor Join") oder Projektionsreduktionen zur Verkleinerung der Zwischenergebnisse.
- **Kostenbasierte Optimierung:** Hierbei werden Statistiken über Tupelanzahl, Kardinalitäten und Indizes genutzt, um die geschätzten Kosten alternativer Pläne zu bewerten. Das Ziel ist, den optimalen Plan mit minimalem Ressourcenverbrauch auszuwählen.

Ein klassisches Beispiel für kostenbasierte Optimierung ist der *System R Optimizer*, der mittels dynamischer Programmierung die optimale Join-Reihenfolge bestimmt [9].

2.4 Einflussfaktoren auf die Performanz

Die Ausführungszeit einer Anfrage hängt von zahlreichen Systemaspekten ab:

- **Indexierung:** Beschleunigt den Zugriff auf einzelne Tupel oder Wertebereiche.

- **Buffer Management:** Vermeidet unnötigen Festplattenzugriff durch intelligenten Umgang mit dem Hauptspeicher.
- **Parallelisierung:** Nutzt mehrere CPU-Kerne zur gleichzeitigen Ausführung von Teilplänen.
- **Materialisierung vs. Pipelinierung:** Unterschiedliche Strategien zur Auswertung mehrstufiger Operatorbäume.

2.5 IDB und EDB: Relevanz für Optimierung

In der Datalog-Theorie wird zwischen der *Extensional Database (EDB)* – also den gespeicherten Basisdaten – und der *Intensional Database (IDB)* – abgeleiteten Daten aus Regeln – unterschieden. Diese Konzepte sind insbesondere bei rekursiven Abfragen relevant.

Beispiel: In einem Regelwerk könnte die transitiv abgeschlossene "Vorgesetztenbeziehung" durch rekursive Regeln aus einfachen "Chef von"-Tripeln berechnet werden – ähnlich wie SPARQL `PROPERTY PATHS` dies im RDF-Kontext ermöglichen. Optimierungsstrategien müssen in solchen Fällen den Fixpunktalgorithmus berücksichtigen.

3 RDF & SPARQL – Datenmodell und Anfrageverarbeitung

Das *Resource Description Framework (RDF)* ist ein standardisiertes Datenmodell zur Repräsentation strukturierter Informationen im Web. Es bildet die Grundlage des Semantic Web und ermöglicht die Modellierung von Aussagen über Ressourcen in Form von Tripeln. Die zugehörige Anfragesprache *SPARQL* erlaubt das gezielte Durchsuchen solcher RDF-Graphen und stellt somit das Gegenstück zu SQL im relationalen Kontext dar.

3.1 RDF: Struktur und Semantik

RDF beschreibt Informationen durch atomare Aussagen in der Form eines Tripels: *(Subjekt, Prädikat, Objekt)*. Diese Tripel lassen sich als gerichtete, beschriftete Kanten in einem Graphen interpretieren. Subjekt und Objekt sind Ressourcen (identifiziert durch URIs) oder Literale; das Prädikat beschreibt die Beziehung zwischen ihnen.

Listing 1: Beispielhafte RDF-Repräsentation einer Einwohnerzahl

```
dbr:Erlangen dbo:populationTotal "114257"^^xsd:
   nonNegativeInteger .
```

RDF ist schemalos, kann aber mithilfe von *RDF Schema (RDFS)* und der *Web Ontology Language (OWL)* um Typinformationen, Klassenhierarchien und logische Einschränkungen ergänzt werden. Diese ermöglichen auch Inferenzregeln, wie z. B. das automatische Ableiten transitiver Beziehungen [1].

3.2 Open World Assumption (OWA)

Ein zentrales semantisches Prinzip von RDF ist die *Open World Assumption* (OWA): Das Fehlen einer Aussage bedeutet nicht, dass sie falsch ist – sondern lediglich, dass nichts darüber bekannt ist. Dieses Prinzip steht im Gegensatz zur *Closed World Assumption (CWA)* relationaler Systeme, wo nicht gespeicherte Fakten implizit als falsch betrachtet werden. Die OWA wirkt sich direkt auf Optimierungsstrategien aus, da beispielsweise Filterelimination oder Join-Elimination nicht ohne Weiteres möglich sind.

3.3 SPARQL: Abfragesprache für RDF

SPARQL (SPARQL Protocol and RDF Query Language) ist die standardisierte Sprache zur Abfrage von RDF-Daten. SPARQL-Anfragen bestehen aus *Graph Patterns*, die über Variablen definiert werden und mit Tripeln in der Datenbasis abgeglichen werden.

Listing 2: SPARQL-Abfrage nach Städten mit mehr als 100.000 Einwohnern

```
SELECT ?city ?population WHERE {
  ?city rdf:type dbo:City .
  ?city dbo:country dbr:Germany .
  ?city dbo:populationTotal ?population .
  FILTER (?population > 100000)
}
ORDER BY DESC(?population)
```

SPARQL unterstützt vier Haupttypen von Anfragen:

- **SELECT:** Liefert Bindungen für Variablen.

- **ASK:** Gibt einen Wahrheitswert zurück.
- **CONSTRUCT:** Generiert neue RDF-Tripel.
- **DESCRIBE:** Liefert eine Beschreibung einer Ressource.

Darüber hinaus erlaubt SPARQL komplexe Konstrukte wie `OPTIONAL` (äquivalent zu *left outer joins*), `UNION`, `FILTER` sowie `PROPERTY PATHS` zur Navigation in RDF-Graphen.

3.4 Anfrageverarbeitung in RDF-Datenbanken

RDF-DBMS (z. B. Apache Jena, Virtuoso, Blazegraph) verarbeiten SPARQL-Anfragen in mehreren Schritten:

1. **Parsing:** Die SPARQL-Anfrage wird in eine interne Repräsentation überführt.
2. **Optimierung:** Der Anfragebaum wird hinsichtlich Filterplatzierung, Join-Reihenfolge und Pattern-Gruppe analysiert.
3. **Ausführung:** Die optimierte Anfrage wird über spezialisierte Speicherstrukturen (z. B. Triple Tables, Hexastore) ausgeführt.

Ein zentrales Problem besteht in der **hohen Joindichte**: Selbst einfache Abfragen erfordern mehrere Joins, da RDF keine Attributtabellen, sondern atomare Aussagen nutzt. Dadurch entstehen viele Kanten im Anfragegraphen – mit teils erheblichem Einfluss auf die Laufzeit.

3.5 Technische Besonderheiten RDF-basierter Systeme

Im Gegensatz zu relationalen Datenbanken setzen RDF-Systeme auf spezielle Speicherstrukturen:

- **Triple Table:** Speicherung aller Tripel in einer dreispaltigen Tabelle (Subjekt, Prädikat, Objekt).
- **Vertical Partitioning:** Separate Tabellen je Prädikat zur Effizienzsteigerung.

- **Property Tables:** Gruppierung häufig gemeinsam auftretender Prädikate.
- **Hexastore:** Speicherung aller 6 Permutationen von S-P-O zur Beschleunigung von Joins.

Diese Strukturen bestimmen maßgeblich die Optimierbarkeit von SPARQL-Anfragen und beeinflussen sowohl den Planungsaufwand als auch die Ausführungsgeschwindigkeit.

4 Optimierungsprobleme in RDF-Datenbanken

Im Vergleich zu klassischen relationalen Datenbanksystemen stellt das RDF-Datenmodell die Anfrageoptimierung vor neuartige Herausforderungen. Diese resultieren sowohl aus der schemalosen und semistrukturierten Natur von RDF als auch aus der offenen Semantik, wie sie durch die Open World Assumption (OWA) definiert wird. In diesem Kapitel werden zentrale Optimierungsprobleme analysiert, die bei der Ausführung von SPARQL-Anfragen in RDF-Datenbanken typischerweise auftreten.

4.1 Fehlende Schemata und begrenzte Ontologien

RDF-Datenbanken verfügen typischerweise über kein fixes, global definiertes Schema. Im Gegensatz zu relationalen Datenbanken, in denen die Struktur der Daten über Tabellen- und Attributdefinitionen klar vorgegeben ist, liegt RDF-Daten meist keine explizite Schemadefinition zugrunde. Zwar können optionale semantische Erweiterungen durch Ontologien in RDFS oder OWL eingebunden werden, doch sind diese in der Praxis häufig zu generisch, unvollständig oder nicht konsistent gepflegt. Dadurch fehlen dem Optimierer wesentliche Informationen, etwa über Kardinalitäten, Schlüssel oder Typverteilungen [5].

4.2 Hohe Joindichte bei einfachen Anfragen

Ein charakteristisches Merkmal von SPARQL-Anfragen ist ihre Joindichte: Selbst einfache Informationsabfragen bestehen aus mehreren Tripelmustern, die über Variablen verknüpft sind. So entspricht z. B. eine Abfrage wie „Welche Autoren haben Bücher veröffentlicht, die 2020 erschienen sind?“ bereits einem mehrstufigen Join über die Tripel `(?book dc:creator ?author)`, `(?book`

`dc:date ?year)` und `FILTER (?year = 2020)`. In relationalen Datenbanken könnten solche Informationen in einer einzigen, gut indizierten Tabelle abgefragt werden, während RDF-Systeme mehrere Tripelrelationen verbinden müssen.

Die effiziente Bestimmung der optimalen Join-Reihenfolge ist bei hoher Joindichte besonders anspruchsvoll, da die Anzahl möglicher Join-Bäume exponentiell mit der Anzahl der Tripelpattern wächst [4].

4.3 Fehlende Kardinalitätsabschätzungen

Ein zentrales Problem bei der Optimierung ist die fehlende oder ungenaue Schätzung der Kardinalitäten einzelner Tripelpattern. Während relationale Optimierer häufig auf detaillierte Statistiken über Werteverteilungen, Indizes und Histogramme zurückgreifen können, sind entsprechende Metainformationen in RDF-Systemen selten vorhanden oder ungenau. Dies liegt u. a. an der extrem heterogenen Nutzung von Prädikaten und an der typischen „long tail"-Verteilung von Vokabularen – viele Prädikate treten sehr selten auf, während wenige sehr häufig vorkommen [11].

Falsche Kardinalitätsschätzungen führen in kostenbasierten Optimierern regelmäßig zu suboptimalen Ausführungsplänen, insbesondere bei mehrfach geschachtelten Joins oder OPTIONAL-Konstrukten.

4.4 Einfluss der Open World Assumption (OWA)

Die OWA wirkt sich fundamental auf Optimierungsstrategien aus. Da das Fehlen einer Aussage nicht bedeutet, dass sie falsch ist, sind Optimierungen wie Filter-Pushdown, Join-Eliminierung oder Ergebnisvorhersage nur eingeschränkt anwendbar. Besonders bei OPTIONAL-Blöcken ist die semantische Bedeutung komplex: Das Weglassen eines Tripels kann valide sein, auch wenn dadurch keine vollständige Bindung entsteht. Optimierer müssen also Worst-Case-Szenarien einkalkulieren, was zu konservativen und ineffizienten Plänen führen kann.

4.5 Komplexe SPARQL-Konstrukte: OPTIONAL, FILTER, UNION

SPARQL bietet mit Konstrukten wie `OPTIONAL`, `FILTER` und `UNION` mächtige Ausdrucksmittel, die jedoch die Optimierung erheblich erschweren. OPTIO-

NAL erzeugt implizit einen left outer join, wodurch bestimmte Join-Reihenfolgen ausgeschlossen werden. FILTER-Ausdrücke, insbesondere wenn sie auf BIND-Ergebnisse oder externe Funktionen zugreifen, lassen sich häufig nicht vorziehen oder aufteilen. UNIONs hingegen führen zu mehrfachen Teilplänen mit potenziell überlappenden Ergebnissen.

Ein einfaches Beispiel ist eine Anfrage nach Geburtsdaten oder Sterbedaten prominenter Persönlichkeiten:

Listing 3: Beispiel: OPTIONAL-Abfrage in SPARQL

```
SELECT ?person ?birth ?death WHERE {
  ?person dbo:birthDate ?birth .
  OPTIONAL { ?person dbo:deathDate ?death . }
}
```

Ein naiver Optimierer könnte das OPTIONAL-Muster ungünstig an den Anfang des Ausführungsplans setzen, was zu deutlich schlechterer Performance führt.

4.6 Strukturelle Probleme: Blank Nodes und Named Graphs

Blank Nodes – also Ressourcen ohne URI – erschweren das Identifizieren und Joins über Entitäten. Da sie nicht global referenzierbar sind, müssen sie intern durch systemgenerierte Identifikatoren ersetzt werden. Dies erschwert insbesondere Inferenzmechanismen und die Nachverfolgbarkeit von Aussagen.

Named Graphs wiederum erlauben die Gruppierung von Tripeln in kontextuelle Teilgraphen. Zwar bietet dies konzeptionelle Vorteile (z. B. zur Provenienz oder Versionierung), doch führt dies in der Optimierung zu zusätzlichem Planungsaufwand, da Anfragen graphübergreifend analysiert werden müssen.

5 Optimierungsansätze im RDF-Kontext

Angesichts der strukturellen und semantischen Herausforderungen von RDF-Datenbanken wurden in den letzten Jahren zahlreiche Optimierungsstrategien entwickelt, die speziell auf die Besonderheiten graphbasierter Datenmodelle zugeschnitten sind. Einige davon orientieren sich an Prinzipien aus der relationalen Welt, andere greifen auf neue Konzepte zurück. Dieses Kapitel

gibt einen systematischen Überblick über zentrale Optimierungsmethoden und ordnet sie in Bezug auf ihre Einsatzbedingungen und Wirksamkeit ein.

5.1 Heuristikbasierte Optimierung

Viele RDF-Engines, insbesondere Open-Source-Systeme wie Apache Jena oder Eclipse RDF4J (ehemals Sesame), implementieren heuristische Optimierungen, um typische Anfragen effizienter zu verarbeiten. Dabei kommen einfache Regeln zum Einsatz, die unabhängig von detaillierten Statistiken funktionieren.

Ein zentrales Prinzip ist die Reorganisation der Tripelpattern: Selektive Muster – also solche mit häufigen Konstanten oder spezifischen Prädikaten – werden möglichst weit vorne im Anfragebaum verarbeitet. Dadurch kann die Bindungsmenge früh reduziert werden, was sich auf nachfolgende Joins positiv auswirkt. Ebenfalls etabliert ist der sogenannte *Filter Pushdown*, bei dem FILTER-Ausdrücke so nah wie möglich an die Datensätze verschoben werden, auf die sie sich beziehen [15].

Ein typisches Beispiel ist die Umstrukturierung folgender Anfrage:

Listing 4: Unoptimierte vs. heuristisch optimierte Reihenfolge von Tripeln

```
# Unoptimiert:
?x dbo:birthPlace ?place .
?x rdf:type dbo:Person .
?x foaf:name ?name .

# Optimiert:
?x rdf:type dbo:Person .
?x foaf:name ?name .
?x dbo:birthPlace ?place .
```

Hierbei wird angenommen, dass ‘rdf:type‘ eine stark selektive Einschränkung darstellt und daher zuerst evaluiert werden sollte.

5.2 Kostenbasierte Optimierung

Komplexere RDF-Datenbanken wie Virtuoso oder Blazegraph nutzen kostenbasierte Optimierer, die verschiedene Anfragepläne bewerten und auf Basis geschätzter Ausführungskosten den effizientesten Plan wählen. Ähnlich

wie in relationalen Systemen stützt sich dieser Prozess auf Statistiken über Prädikatsverteilungen, Kardinalitäten und Wertfrequenzen [10].

Besonderheiten ergeben sich jedoch aus der Struktur der Daten: RDF-Prädikate verhalten sich nicht wie feste Spalten, sondern können sehr unterschiedlich interpretiert und verteilt sein. Moderne Systeme greifen daher zu Hybridstrategien, bei denen heuristische Vorauswahl mit statistischer Bewertung kombiniert wird.

Ein Beispiel ist die Abfrage nach allen Akteuren eines Films, bei der JOINs über ‘dbo:starring‘, ‘dbo:director‘, ‘dbo:releaseDate‘ und weitere Prädikate erforderlich sind. Ein Optimierer wählt dabei nicht nur die Join-Reihenfolge, sondern berücksichtigt auch, welche Teilpläne potenziell hohe Kosten verursachen (z. B. bei großen Bindungsräumen oder UNION-Konstrukten).

5.3 Strukturbasierte Optimierung nach Graphmustern

Neuere Forschungsarbeiten klassifizieren SPARQL-Anfragen nach ihrer strukturellen Form – z. B. als Sterne (ein zentrales Subjekt mit mehreren Prädikaten), Pfade (verkettete Prädikate), Bäume oder Zyklen. Diese Muster lassen sich nutzen, um Anfragen gezielt zu zerschneiden oder zusammenzufassen.

Ein Sternmuster wie:

Listing 5: Sternförmiges SPARQL-Muster

```
?person foaf:name ?name ;
        dbo:birthDate ?birth ;
        dbo:deathDate ?death ;
        dbo:occupation ?job .
```

ermöglicht die Bündelung der Prädikate in sogenannten *Property Tables* oder erlaubt parallele Verarbeitung durch semantisches Partitionieren [8]. Für Pfade wiederum kann ein sogenannter *Path Index* erzeugt werden, der häufige Prädikatssequenzen vorab berechnet und indiziert.

5.4 Speicherstruktur und physische Repräsentation

Die physische Speicherung hat erheblichen Einfluss auf die Effizienz der Anfrageverarbeitung. Während einfache RDF-Systeme eine klassische Tripel-Tabelle (Subjekt, Prädikat, Objekt) nutzen, greifen optimierte Systeme auf spezialisierte Layouts zurück:

- **Vertical Partitioning:** Eine separate Tabelle je Prädikat. Vorteilhaft für selektive Prädikatsanfragen, jedoch Join-intensiv.
- **Property Tables:** Gruppierung häufig gemeinsam auftretender Prädikate in eine Zeile – ähnlich einer relationalen Projektion.
- **Hexastore:** Speicherung aller sechs Permutationen von SPO (z. B. POS, OSP etc.), um beliebige Zugriffsmuster effizient zu unterstützen [13].

Die Wahl der Speicherstruktur beeinflusst unmittelbar die Indexierung, Join-Kosten und Parallelisierbarkeit.

5.5 Kompilierung in relationale Ausführungspläne

Einige Systeme nutzen sogenannte R2RML-Mappings, um RDF-Daten auf relationale Strukturen abzubilden. SPARQL-Anfragen werden dabei in SQL überführt, sodass bestehende relationale Optimierer (z. B. aus PostgreSQL) genutzt werden können. Voraussetzung ist jedoch eine wohldefinierte Ontologie oder Mapping-Tabelle. Beispiele für derartige Systeme sind Ontop oder D2RQ.

Vorteilhaft ist dabei die Wiederverwendbarkeit bewährter SQL-Techniken; allerdings leidet die Flexibilität bei der Abbildung komplexer RDF-Graphenstrukturen.

5.6 Materialisierung und Vorberechnung

Zur Verbesserung der Performance bei häufigen Abfragen kommen in RDF-Systemen zunehmend materialisierte Views zum Einsatz. Dabei handelt es sich um vorab berechnete Antwortmengen oder Teilergebnisse, die zur Laufzeit direkt verwendet werden können. Insbesondere bei bekannten Mustern – etwa rekursiven Beziehungen oder häufigen Joinketten – lohnt sich die Vorberechnung [3].

Alternativ können semantische Indizes über Subklassen, Subproperties oder Instanzen erzeugt werden, um Inferenzkosten zur Laufzeit zu vermeiden.

5.7 Ontologiebasierte Anfrageumschreibung

Schließlich ermöglichen Ontologien eine semantische Erweiterung der Anfrageverarbeitung. In sogenannten *ontology-aware systems* werden SPARQL-

Anfragen automatisch um Regeln ergänzt, die sich aus OWL-Axiomen ableiten. Beispielsweise kann bei einer Anfrage nach ‘dbo:University‘ auch nach Subklassen wie ‘dbo:TechnicalUniversity‘ gesucht werden, ohne dass dies explizit formuliert wurde [12].

Problematisch ist jedoch, dass komplexe Ontologien zu einer exponentiellen Erweiterung des Anfragebaums führen können, was die Optimierung massiv erschwert. In der Praxis ist daher häufig eine Mischung aus statischer Inferenz (Materialisierung) und dynamischer Umschreibung (Rewriting) anzutreffen.

6 Fazit

Hier schließt du deine Arbeit ab.

Literatur

[1] Daniele Antonellini, Agnese Poggi und Riccardo Rosati. „OWL and SPARQL: From Theory to Practice“. In: *Semantic Web Journal* 6.4 (2015), S. 361–375.

[2] Tim Berners-Lee, James Hendler und Ora Lassila. „The Semantic Web“. In: *Scientific American* 284.5 (2001), S. 34–43.

[3] Najib Elzein und Thomas Eiter. „Materialized View Maintenance for RDF Datasets“. In: *Semantic Web* 10.6 (2019), S. 1051–1074.

[4] Andrey Gubichev und Thomas Neumann. „Explaining RDF Query Results“. In: *International Conference on Management of Data (SIGMOD)*. 2013, S. 245–256.

[5] Andreas Harth und Stefan Decker. „Optimized Index Structures for Querying RDF from the Web“. In: *Web Semantics: Science, Services and Agents on the World Wide Web* 7.2 (2010), S. 113–130.

[6] Aidan Hogan u. a. „Knowledge graphs“. In: *ACM Computing Surveys (CSUR)* 54.4 (2021), S. 1–37.

[7] Alfons Kemper und André Eickler. *Datenbanksysteme: Eine Einführung*. 11. Aufl. De Gruyter Oldenbourg, 2022.

[8] Muhammad Saleem u. a. „A Structural Approach for Optimizing SPARQL Queries“. In: *The Semantic Web – ISWC 2019*. 2019, S. 41–59.

[9] Patricia G. Selinger u. a. „Access path selection in a relational database management system“. In: *ACM SIGMOD Record* 9.2 (1979), S. 23–34.

[10] Juan F. Sequeda, Olaf Hartig und Daniel P. Miranker. „Cost-Based Query Rewriting for SPARQL-to-SQL Translation“. In: *Journal of Web Semantics* 26 (2014), S. 2–19.

[11] Christoph Stahl und Gero Decker. „Cost-based Query Optimization for SPARQL“. In: *Datenbank-Spektrum* 12.1 (2012), S. 5–14.

[12] George Tsatsanifos, Theodore Dalamagas und Timos Sellis. „On the Computation of the Transitive Closure of Ontologies“. In: *ESWC*. 2012, S. 154–168.

[13] Christoph Weiss, Petko Georgiev und Abraham Bernstein. „Hexastore: Sextuple Indexing for Semantic Web Data Management“. In: *Proceedings of the VLDB Endowment* 1.1 (2008), S. 1008–1019.

[14] Marcin Wylot u. a. „RDF Data Storage and Query Processing Schemes: A Survey“. In: *ACM Computing Surveys* 51.4 (2018), S. 1–36.

[15] Bo Yuan, Zhifeng Bao und J. X. Yu. „Effective Heuristics-Based SPARQL Query Optimization“. In: *International Conference on Web Information Systems Engineering (WISE)*. 2015, S. 227–241.

[16] S. Yuksel, Y. Cankaya und A. Yuksel. „An analysis of RDF storage models and query optimization techniques“. In: *Procedia Computer Science* 59 (2015), S. 516–525.