Property-Graphen: eine kurze Einführung

Zu den Tätigkeiten der Stabstelle Forschung und Entwicklung der VZG gehört auch das Ausprobieren und Evaluieren neuer Verfahren und Techniken. So kommt es dass ich mich seit Anfang 2024, angeregt durch einen Anwendungsfall im Projekt NFDI4Objects, verstärkt mit so genannten Property-Graphen zur Strukturierung und Verarbeitung von (Meta)daten beschäftige.

Property-Graphen (oft auch *Labeled Property Graphen*) fallen als Datenbankmodell unter die so genannten NoSQL-Datenbanken und noch spezieller unter die Graphdatenbanken. Hierbei werden Daten nicht wie bei SQL in Form von Tabellen sondern in Form von Graphen aus *Knoten* gespeichert, die durch *Kanten* miteinander verbunden sind. Eine Besonderheit von Property-Graphen ist, dass sowohl Knoten als auch Kanten jeweils ein oder mehrere *Labels* haben und mit *Eigenschaften* versehen werden können.

## Beispiel

Zur Veranschaulichung soll folgende Sammlung einiger Charaktere, Beziehungen und Eigenschaften aus dem Star-Wars-Universum dienen: Padmé, Anakin und Luke sind Personen unterschiedlichen Geschlechts und R2D2 ist ein Roboter. In Episode 1 gehört R2D2 zu Padmé, die ihn in Episode 2 Anakin zu ihrer gemeinsamen Hochzeit schenkt, und in Episode 4 gelangt der Roboter zu Luke, der in Episode 3 als Kind von Padmé und Anakin geboren wurde.

Diese Informationen lassen sich in einem Property-Graphen mit Charakteren als Knoten und ihren Beziehungen als Kanten modellieren. [Abbildung 1](#fig-image) zeigt eine mögliche Visualisierung dieses Graphen. Darin sind Knoten-Identifier fett hervorgehoben, Labels kursiv und Eigenschaften in Festbreitenschriftart. Bis auf die Beziehung zwischen Padmé und Anakin sind alle Kanten gerichtet.

|  |
| --- |
| Abbildung 1: Visualisierung des Beispiel-Graphen |

Quelle: [Artikel-Notizbuch](https://jakobib.github.io/pgraphen2024/index.qmd.html)

Zur Kodierung von Property-Graphen gibt es verschiedene Datenformate und Datenbanksysteme, die sich in Details wie den erlaubten Zeichen in Identifiern, der Wiederholbarkeit von Labels und Eigenschaften und in möglichen Datentypen von Eigenschaftswerten (im Beispiel Zeichenkette für die Eigenschaft gender und Zahl für die Eigenschaft episode) unterscheiden. Im Folgenden werden zwei mögliche Kodierungen vorgestellt und Gemeinsamkeiten und Unterschiede zum RDF-Format erklärt.

## Property-Graph-Datenbanken

Etabliert wurden Property-Graphen insbesondere durch das Datenbankmanagementsystem (DBMS) [Neo4J](https://neo4j.com/). Die Open-Source-Software war lange Marktführer in diesem Bereich und setze dort Standards wie die Abfragesprache *Cypher* (siehe [Listing 1](#lst-cypher) und [Listing 2](#lst-match)), die inzwischen auch von anderen Systemen unterstützt wird. Dazu zählen derzeit die Open-Source-DBMS [Kùzu](https://kuzudb.com/), [Memgraph](https://memgraph.com/) und [FalkorDB](https://www.falkordb.com/). Im letzten Update des SQL-Standard (SQL:2023) wurde zumdem unter dem Namen SQL/PGQ eine Erweiterung für Property-Graphen in SQL-Datenbanken definiert. Es ist also davon auszugehen, dass in Zukunft weitere DBMS Property-Graphen unterstützen.

Der Beispielgraph kann in Neo4J oder in einer damit kompatiblen Datenbank mit den Cypher-Statements in [Listing 1](#lst-cypher) angelegt werden. Da Knoten-Identifier in der Datenbank rein intern sind, sind die Namen zusätzlich als Eigenschaft name angegeben. Außerdem unterstützt Cypher nur gerichtete Kanten, daher ist die Beziehung zwischen Padmé und Anakin weggelassen.

|  |
| --- |
| Listing 1: Beispiel-Graph mit Cypher-Statements  CREATE (Anakin:person {gender:"male", name:"Anakin"}) CREATE (Luke:person {gender:"male", name:"Luke"}) CREATE (Padmé:person {gender:"female", name:"Padmé"}) CREATE (R2D2:robot {name:"R2D2"}) CREATE (Padmé)-[:owns {episode:1}]->(R2D2) CREATE (Anakin)-[:owns {episode:2}]->(R2D2) CREATE (Anakin)-[:child {episode:3}]->(Luke) CREATE (Padmé)-[:child {episode:3}]->(Luke) CREATE (Luke)-[:owns {episode:4}]->(R2D2) |

Nun können die Daten mit Cypher-Abfragen ausgewertet werden ([Listing 2](#lst-match)):

|  |
| --- |
| Listing 2: Abfragen in Cypher-Syntax  # Wer sind die Eltern der Person mit dem Namen Luke? MATCH (p)-[:child]->(:person {name:"Luke"}) RETURN p  # Wie heißen die Besitzer von R2D2 ab Episode 2? MATCH (p)-[e:owns]->({name:"R2D2"}) WHERE e.episode >= 2 RETURN p.name |

Wie bei allen Datenbanken können die Abfrageergebnisse natürlich nur so gut sein wie die Datenbasis: so würde eine Frage nach den Kindern von Anakin nur Luke ergeben, weil seine Zwillingsschwester Leia im Beispiel-Graph fehlt. Grundsätzlich stellt die Modellierung von Property-Graphen aber ein leistungsfähiges und flexibels Werkzeug vor allem für semi-strukturierte und verknüpfte Daten da.

## Property Graph Exchange Format

Während die Standardisierung der Datenbanksprache Cypher relativ weit fortgeschritten ist, gibt es noch kein etabliertes Dateiformat zum Austausch von Property-Graphen. Zusammen mit den Wissenschaftlern Hirokazu Chiba, Ryota Yamanaka und Shota Matsumoto entwickele ich deshalb das *Property Graph Exchange Format* (Chiba 2024). [Listing 3](#lst-pg) zeigt die Kodierung des vollständigen Beispiel-Graphen im PG-Format. Die Standardisierung beinhaltet äquivalente Kodierungen in JSON (PG-JSON und PG-JSONL).

|  |
| --- |
| Listing 3: Beispiel-Graph im PG Format  # Knoten mit Knoten-Typ (Label) und Eigenschaften Padmé :person gender:female Anakin :person gender:male  Luke :person gender:male  R2D2 :robot    # Kanten mit Kanten-Typ (Label) und Eigenschaften Padmé -> R2D2 :owns episode:1 Padmé -- Anakin :marriage episode:2 Anakin -> R2D2 :owns episode:2  Anakin -> Luke :child episode:3 Padmé -> Luke :child episode:3  Luke -> R2D2 :owns episode:4 |

und wird durch die Implementierung der Programmbibliothek [pgraphs](https://www.npmjs.com/package/pgraphs) zur Konvertierung zwischen verschiedenen Graph-Formaten und Datenbanksystemen begleitet. So wurde der Beispielgraph in Neo4J in [Listing 1](#lst-cypher) automatisch mit dem Aufruf pgraph -t cypher -i name star-wars.pg aus [Listing 3](#lst-pg) erzeugt.

## Vergleich mit RDF

RDF ist ein alternatives Graph-basiertes Datenmodell, das zusammen mit Konzepte wie Linked Data und Semantic seit Jahrzehnten propagiert wird. Rein formal bestehen Daten auch im RDF-Modell aus Knoten und Kanten, wobei Knoten als “Ressourcen” und Kanten als “Triples” bezeichnet werden. Kanten-Label heißen in RDF “Properties”, Entsprechungen zu Knoten-Label und Eigenschaften gibt dagegen nicht.

Abgesehen von formalen Unterschieden, die sich durch Datenkonvertierung weitgehend überbrücken lassen, ist hilfreich die unterschiedliche Motivationen zu verstehen, aus denen RDF und Property Graphen jeweils entstanden sind: RDF ist grundsätzlich ein Austauschformat zum Publizieren und Zusammenführen von Daten. Grundstein bilden die übergreifend nutzbaren URIs zur weltweit eindeutigen Identifizierung von Konzepten. Bei Property-Graphen geht es dagegen primär um die effiziente Speicherung und Auswertung von vernetzen Daten in einer abgeschlossenen Datenbank.

Zum Vergleich zeigt [Listing 4](#lst-ttl) den Beispiel-Graphen in RDF-Turtle-Syntax und [Listing 5](#lst-sparql) die Entsprechung der ersten Cypher-Abfrage aus [Listing 2](#lst-match) in der RDF-eigenen Abfragesprache SPARQL. Die Rolle der Knoten-Label und Knoten-Eigenschaften übernehmen in RDF zusätzliche Kanten. Kanten-Eigenschaften und ungerichteten Kanten sind in diesem Beispiel dagegen weggelassen, weil ihre Entsprechung in RDF etwas komplexer zu modellieren ist. Obwohl der RDF-Graph ([Abbildung 2](#fig-rdf-image)) weniger Informationen enthält, ist er im Vergleich zu [Abbildung 1](#fig-image) etwas unübersichtlicher.

|  |
| --- |
| Listing 4: Beispiel-Graph in RDF/Turtle (ohne Kanten-Eigenschaften)  <Padmé> a <person> ; <gender> "female" . <Anakin> a <person> ; <gender> "male" . <Luke> a <person> ; <gender> "male" . <R2D2> a <robot> .  <Padmé> <owns> <R2D2> . <Anakin> <owns> <R2D2> . <Anakin> <child> <Luke> . <Padmé> <child> <Luke> . <Luke> <owns> <R2D2> . |

|  |
| --- |
| Listing 5: SPARQL-Abfrage  # Wer sind die Eltern der Person Luke? SELECT ?p { ?p <child> <Luke> . <Luke> a <person> } |

|  |
| --- |
| Abbildung 2: Visualisierung des Beispiel-Graphen in RDF |

Quelle: [Artikel-Notizbuch](https://jakobib.github.io/pgraphen2024/index.qmd.html)

Mit der kommenden Version RDF 1.2 (auch *RDF-star*) können RDF-Triple mit weiteren Tripeln angereichtert werden, so dass ine direkte Entsprechung zu Kanten-Eigenschaften möglich wäre (siehe [Listing 6](#lst-ttl2) und [Listing 7](#lst-sparql2)). Abgesehen von der begrenzten Software-Unterstützung werden Daten mit dieser Erweiterung allerdings nicht unbedingt übersichtlicher.

|  |
| --- |
| Listing 6: Kanten des Beispiel-Graph in RDF/Turtle 1.2  <Padmé> <owns> <R2D2> {| <episode> 1 |} . <Anakin> <owns> <R2D2> {| <episode> 2 |} .  <Anakin> <parent> <Luke> {| <episode> 3 |} . <Padmé> <parent> <Luke> {| <episode> 3 |} . <Luke> <owns> <R2D2> {| <episode> 4 |} . |

|  |
| --- |
| Listing 7: SPARQL 1.2 Abfrage  # Wer sind die Besitzer von R2D2 ab Episode 2? SELECT ?p {   BIND( << ?p <owns> <R2D2> >> AS ?e )  FILTER ( e.episode >= 2 ) } |

Die wesentlichen Unterschiede von RDF und Property-Graphen sind in [Tabelle 1](#tbl-rdf-pg) zusammengefasst. Grundsätzlich ist RDF vor allem für die Zusammenführung von Daten aus unterschiedlichen Quellen sinnvoll. RDF-Daten werden daher tendenziell eher projektübergreifend und nachnutzbar angelegt. Dieser Vorteil birgt in der Praxis allerdings auch die Gefahr von langwierigen Prozessen und theoretischen, komplexeren Lösungen.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Tabelle 1: RDF und Property-Graphen im Vergleich   | RDF | Property Graphen | | --- | --- | | Graph aus Knoten und Kanten | Graph aus Knoten, Kanten und Eigenschaften | | Etabliert durch das W3C | Standardisierung noch nicht abgeschlossen | | Identifier sind globale URIs | Interne oder lokale Knoten-Identifier | | Globale Ontologien und Regeln | Lokale Datenbank-Schemas | | Abfragesprache SPARQL | Abfragesprache Cypher | |

## Property Graphen an der VZG

An der VZG werden Property-Graphen vor allem im Projekt [NFDI4Objects](https://www.nfdi4objects.net/) eingesetzt. Darüber hinaus sollen Sacherschließungsdaten des K10plus für Analysen in einer Graphdatenbank indexiert werden. In beiden Fällen ist das Ergebnis ein so genannter *Knowledge Graph*. Für die Integration mit anderen Datenquellen sollen beide Knowledge Graphen auch in RDF bereitgestellt und RDF-Daten durch Konvertierung in das PG-Format mit Property-Graphen zusammengeführt werden.

## Zusammenfassung

Property-Graphen bilden ein flexibles und nicht zu kompliziertes Werkzeug zur Strukturierung, Speicherung und Auswertung vernetzter Daten. Im Gegensatz zu RDF sind Property Graphen allerdings nicht als allgemeines Austauschformat gedacht. Die VZG setzt Property-Graphen ein und ist an der laufenden Standardisierung beteiligt.

Chiba, Voß, Hirokazu. 2024. „Property Graph Exchange Format“. <https://github.com/pg-format/specification>.