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 bilden ein Datenbankmodell, das unter die so genannten NoSQL-Datenbanken und noch spezieller unter die Graphdatenbanken fällt. 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 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 I gehört R2D2 zu Padmé, die ihn in Episode II Anakin zu ihrer gemeinsamen Hochzeit schenkt, und in Episode IV gelangt der Roboter zu Luke, der in Episode III 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. [Listing 1](#lst-pg) zeigt die Kodierung des Graphen im **PG Format**, das ich derzeit zusammen mit den Wissenschaftler Hirokazu Chiba, Ryota Yamanaka, und Shota Matsumoto als Austauschformat für Property-Graphen entwickle. In dem Beispiel entsprechen die Namen der Charaktere den Node-Identifiern und ihr Typ den Node-Labels . Das Geschlecht und die jeweilige Episode sind als Properties den Knoten und Kanten zugeordnet. Kanten haben ebenfalls ein Label mit der Beziehungsart und sie können gerichtet (->) oder ungerichtet (--) sein.

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

Der Graph kann auf verschiedene Weise gespeichert und visualisiert werden. [Abbildung 1](#fig-image) zeigt eine mögliche Darstellung ohne Properties. Mit Knoten, Kanten mit Richtungen, Labels und Properties sind schon alle Elemente von Property-Graphen aufgezählt. Je nach konkretem Datenformat und Datenbanksystem können Labels und Properties auch mehrere Werte annehmen und verschiedene Datentypen annehmen.

|  |
| --- |
| Abbildung 1 |

## 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 2](#lst-cypher) und [Listing 3](#lst-match)), die inzwischen auch von anderen Anbietern unterstützt wird. Dazu zählen derzeit die Open-Source-Systeme [Kùzu](https://kuzudb.com/), [Memgraph](https://memgraph.com/) und [FalkorDB](https://www.falkordb.com/), so dass wie bei Relationalen DBMS (RDBMS) die Gefahr für [Vendor Lock-In](https://it-in-bibliotheken.de/management.html#vendor-lock-in) gering ist. Im letzten Update des SQL-Standard (SQL:2023) wurde zumdem unter dem Namen SQL/PGQ eine Teilmenge von Cypher als Abfragesprache für Property-Graphen in SQL-Datenbanken definiert, es ist also davon auszugehen, dass in Zukunft einige RDBMS Property-Graphen auch direkt unterstützen.

Der Beispielgraph kann in Neo4J oder in einer damit kompatiblen Datenbank mit folgenden Cypher-Statements angelegt werden ([Listing 2](#lst-cypher)). Da Knoten-Identifier in der Datenbank rein intern sind, sind die Namen zusätzlich als Property name angegeben. Außerdem unterstützt Cypher nur gerichtete Kanten, daher ist die Beziehung zwischen Padmé und Anakin weggelassen.

|  |
| --- |
| Listing 2: Beispiel-Graph als 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)-[:parent {episode:3}]->(Luke) CREATE (Padmé)-[:parent {episode:3}]->(Luke) CREATE (Luke)-[:owns {episode:4}]->(R2D2) |

Nun können die Daten mit Cypher-Abfragen ausgewertet werden ([Listing 3](#lst-match)):

|  |
| --- |
| Listing 3: Abfragen in Cypher-Syntax  # Wer sind die Eltern der Person mit dem Namen Luke? MATCH (p)-[:parent]->(: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. Im Projekt NFDI4Objects haben wir uns deshalb dazu entschieden die Zusammenführung heterogener Daten aus verschiedenene Quellen in einem Property-Graphen durchzuführen.

## Vergleich mit RDF

Zu den Graphdatenbanken gehören neben Property Graphen auch die RDF-Datenbanken, in denen Daten nach dem RDF-Datenmodell gespeichert werden können.

* Triples
* Können integriert werden
* Benötigen Ontologien

Vor- und Nachteile von Property Graphen im Vergleich zu RDF:

* Flexibler: keine Ontologien notwendig
* Abfragesprache Cypher etwas einfacher als SPARQL
* Nicht so einfach mit externen Daten integriertbar

Fazit: Beides hat seinen Einsatzzweck.

## Property Graphen an der VZG

* NFDI4Objects
* PG format
* pgraphs: Werkzeug zur Konvertierun

## Zusammenfassung

* Flexibler als RDF
* Standardisierung in SQL. Bisher Oracle
* Abfragesprache Cypher