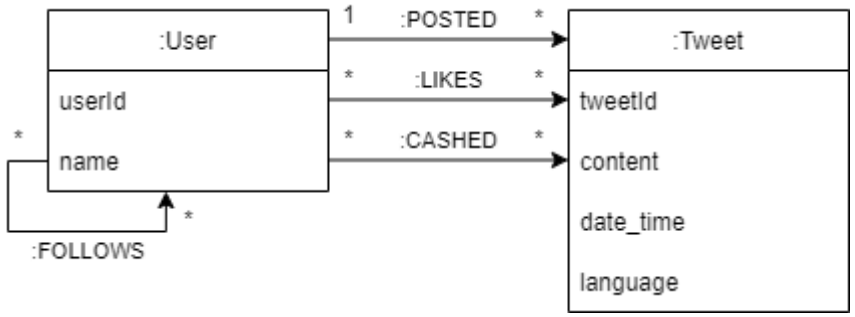
Datenmodell



Da wir uns für Neo4J, eine Graphdatenbank, entschieden haben, konnten wir die User und Tweets jeweils als Knoten darstellen mit den Labels :User und :Tweet. Um zu kennzeichnen, welcher User den Tweet gepostet hat, gibt es eine POSTED-Relationship. Auch die Likes und das Konzept des Fan-Outs der Tweets in den Cache der User haben wir als Relationship-Beziehungen umgesetzt.

Bei letzterem hielten wir es nicht für sinnvoll, die Tweets, wie in der Vorlesung vorgeschlagen, zu duplizieren, um sie für den User zu cachen, weil dies den grundsätzlichen Vorteilen einer Graphdatenbank widersprechen würde, nämlich, dass alle Beziehungen zwischen Knoten als einfache Beziehungen (Relationships) realisiert werden. Durch die CACHED-Beziehung sollte trotzdem eine Performance-Verbesserung beim Laden der Startseite eines Users eintreten, weil auf direktem Wege auf die gecachten anzuzeigenden Tweets zugegriffen werden kann und nicht der Umweg über alle verfolgten Accounts des Users getätigt werden muss.

Erläuterung zum Import

Beim Import der Daten haben wir uns dazu entschieden, zu jedem Knoten-Label und Beziehungs-Label zunächst eine CSV-Datei zu erstellen und diese anschließend über den von Neo4J bereitgestellten CSV Import in die Datenbank zu importieren. Dies war deutlich schneller als die Daten über einzelne Queries oder eine sehr große Query zu importieren.

Bei den Tweets haben wir viele der Daten des ursprünglichen CSVs nicht verwendet, weil diese nicht auf das Szenario anwendbar waren oder keinerlei Relevanz hatten. Wir haben lediglich den Content, die Zeit des Postens und die Sprache übernommen. Die Anzahl an Likes haben wir nicht als Anhaltspunkt für die Like-Generierung genommen, weil uns die Zahlen unrealistisch hoch für die Anzahl an Usern und Followern des neu geschaffenen Netzwerkes erschienen. Wir haben stattdessen die Likes komplett neu generiert nach dem Schema, dass von den Followern des Posters etwa jeder 10. Follower den Post liked.

Erläuterung zu den Queries

Die Queries waren aufgrund des simplen Datenmodells und den aussagekräftigen Beziehungen zwischen den Usern und Tweets einfach umzusetzen. Während in einer relationalen Datenbank aufwändige Joins über große Datenmengen gemacht werden müssten, konnten in dieser Graphdatenbank über einfache MATCHES bezüglich der Beziehungen und IDs auch komplexere Queries effizient ausgeführt werden.

Selbst für das Suchen nach einzelnen Wörtern innerhalb der Tweets war keine Anpassung des Datenmodells nötig, noch nicht einmal das Anlegen eines Full-text search index, wie wir zunächst annahmen. Es könnte allerdings sein, dass das Anlegen eines Full-text search index, welcher von Neo4J bereitgestellt wird, nötig werden würde, wenn die gespeicherten Texte länger werden.

Der Fan-Out in den Cache der Follower beim Erstellen eines neuen Posts musste in mehreren Queries realisiert werden, da nicht nur der Tweet an sich erstellt werden muss, sondern auch die POSTED-Beziehung und pro Follower eine CACHED-Beziehung. Das würde bei einer größeren Anzahl an Usern als in unserem Beispiel vermutlich zu Performance-Problemen führen. Da die Abfrage-Queries im Gegensatz zu den CREATE-Queries in unserem Beispiel schneller sind, wäre es also in einem realen Use-Case wahrscheinlich sinnvoll, auf einen Fan-Out dieser Art zu verzichten.

Aufteilung auf mehrere Knoten

Wir haben auf Basis der Neo4J-Dokumentation einen Cluster bestehend aus drei Knoten aufgebaut. Datenbankverbindungen über das Jupyter-Notebook werden immer zu einem der Knoten aufgebaut. Da immer nur einer der Neo4J-Knoten als LEADER-Knoten (mit WRITE-Zugriff) deklariert ist, können CREATE-Queries nur an diesen Knoten gesendet werden. Die neu angelegten Daten werden dann aber über das gesamte Cluster verteilt. READ-Zugriffe können an jeden Knoten des Clusters gesendet werden. Welcher Knoten der LEADER-Knoten ist, kann sich ändern, beispielsweise wenn der aktuelle LEADER-Knoten ausfällt.

Insgesamt existiert also ein Cluster, welcher die Daten zwar nicht nicht auf verschiedene Knoten verteilt, aber durch die Duplikation der Daten eine Ausfallsicherheit der Datenbank sicherstellt.