Abstract

In dem Vortrag zu Graph Datenbanken war das Ziel, über den Umgang mit Graph Datenbanken zu informieren.

1. Abgrenzung gegenüber relationalen Datenbanken und RDF Triple Stores

Der erste Punkt dreht sich um den Unterschied zwischen relationale Datenbanken und Graph Datenbanken. Relationale Datenbanken speichern die Daten nur in Tabellen, welche miteinander über Primär und Fremdschlüssel verknüpft werden. Bei den Graph Datenbanken werden die Verhältnisse zwischen Daten über Subjekt-Prädikat-Objekte dargestellt. Dadurch kann man die Daten viel einfacher und mit weniger Speicher verknüpfen. Mit den relationalen Datenbanken wäre dies auch möglich, allerdings bräuchte man dafür viele Tabellen, die miteinander verbunden werden müssen, was auch eine hohe Komplexität hervorruft.

2. Typische Graph-Algorithmen

Ein weiterer Punkt sind die typischen Graph-Algorithmen, mit denen Verbindungen zwischen Knoten bestimmt werden können. In dem Vortrag wurden Depth- and Breadth-First Search, Path-Finding with Dijkstra's Algorithm, The A* Algorithm, Graph Theory and Predictive Modeling und Local Bridges vorgestellt.

2.1 Depth- and Breadth-First Search

Bei diesem Algorithmus werden die Vorteile von der Tiefen- und Breitensuche vereint. Es wird wie in der Breitensuche zu erst die gesamte Ebene erfasst und im Anschluss in die Tiefe gegangen. Dies bedeutet man erfasst alle Knoten, welche mit dem Ausgangspunkt verbunden sind (erste Ebene). Danach werden von dem ersten Knoten (meist der linke Knoten) alle weiteren verbundenen Knoten erfasst. Nun werden wieder alle Knoten des Knotens ganz links erfasst. Dies geht so weiter, bis der Knoten ganz links keine weiteren Knoten mehr besitzt. Ebenso wird dann der 2. Knoten von links in der letzten Ebene auf Knoten überprüft, im Anschluss der 3. Knoten von links und wenn es keine Knoten mehr in dieser Ebene gibt, geht es eine Ebene darüber weiter bis zum 2. Knoten des Ausgangknotens. Hier werden nun wieder die Ebenen bis zur letzten Ebene des Knotens am weitesten links betrachten. Dies läuft so weiter, bis man das Ziel gefunden hat.

2.2 Path-Finding with Dijkstra's Algorithm

Der Dijkstra Algorithmus ist schon ein sehr alter Algorithmus, welcher 1956 entwickelt und über die Jahre weiter optimiert wurde. Er baut auf der Breitensuche auf, um den kürzesten Weg zum Ziel zu finden. Dies passiert durch 5 einfache Schritte:

1. Start- und Endknoten auswählen und Startknoten zu den gelösten Knoten hinzufügen
2. Vom Startknoten aus Pfadlänge zu jedem Nachbar aufzeichnen
3. Kürzesten Weg zu einem Nachbarn nehmen und als gelöst markieren
4. Aus der Menge der gelösten Knoten die nächsten Nachbarn aufzeichnen

   (keine Knoten aufzeichnen, die bereits gelöst sind!)
5. Schritt 3 und 4 wiederholen, bis Zielknoten als gelöst markiert wurde

2.3 The A* Algorithm

Der A* Algorithmus ist eine Weiterentwicklung des Dijkstra Algorithmus. Dabei werden Heuristische Kosten mit einbezogen, um den möglicherweisen besseren Pfad zu wählen. Die Heuristischen Kosten sind dabei geschätzte Kosten zum Ziel. Dies ist ebenfalls in 5 einfachen Schritten erklärt:

1. Knoten dessen Gesamtkosten am niedrigsten sind wird überprüft
2. Alle Nachbarknoten betrachten und die Kosten ermitteln
3. Mehrfach gefundene Knoten vergleichen und die meisten Kosten terminieren
4. Schon bearbeitete Knoten müssen nochmal bearbeitet werden, wenn ein besserer Pfad gefunden wird
5. Schritt 1 – 3 wiederholen, bis Zielknoten bearbeitet wird

2.4 Graph Theory and Predictive Modeling

Dieser Algorithmus unterteilt sich in 2 Abschnitte, die Triadic Closures und die Structural Balance. Beide Abschnitte beziehen sich auf soziale Graphen. Der Begriff Triadic Closures wird verwendet, um das Verhältnis zwischen 2 Knoten zu schätzen, wenn diese über einen dritten Knoten miteinander Verbunden sind. Die Aussage von Granovetter lautete: „Wenn ein Knoten A eine starke Beziehung zu B und C hat, dann haben B und C mindestens eine schwache und möglicherweise eine starke Verbindung zueinander!“. Dabei wurde festgestellt, dass zwischen den beiden Knoten B und C keine starke Verbindung bestehen kann, sondern nur eine schwache Verbindung, da dies sonst zu Konflikten und Unstimmigkeiten kommen kann. Dies nennt man daher die Structural Balance.

2.5 Local Bridges

Die Local Bridges beziehen sich auf den Punkt 2.4. Wenn es nun 2 dieser Konstrukte gibt, wie kann man diese nun Verbinden. Also A hat eine starke Beziehung zu B und C. B und C haben eine schwache Beziehung zueinander. Dann gibt es dann noch den Knoten D, welcher eine starke Beziehung zu E und F hat. Diese beiden Knoten haben wiederum eine schwache Beziehung zueinander. Jetzt entsteht die Local Bridge zwischen A und D. Die Frage hierbei war, ob die Bridge eine starke oder schwache Beziehung ist. Die Frage ist leicht beantwortet, da eine starke Beziehung wieder Konflikte

zwischen zum Beispiel D und B hervorrufen würde. Somit ist eine Local Bridge immer eine schwache Beziehung zwischen 2 Konstrukten mit mehreren Anzahlen von Knoten.

3. Graph Dataflow Systems

Beinhaltet die Darstellung von Graphen und das Ändern und Transformieren dieser Graphen.

3.1 Gelly

Gelly ist die Graph API für Apache Flink. Mit Gelly wird die Entwicklung von Grafikanalyse-Anwendungen in Flink zu vereinfacht. Somit ist das transformieren und modifizieren von Graphen möglich. Ebenso bietet Gelly eine Bibliothek für Graph Algorithmen

3.2 Gradoop

Gradoop ist ein Ausdrucksstarkes Graphdatenmodell in Verbindung mit analytischen Operatoren und Algorithmen, welches eine Hohe Benutzerfreundlichkeit und Skalierbarkeit bietet. Dabei werden verschiedene Softwareprojekte von Apache mit einbezogen, wie Apache Flink, Apache Hbase und Apache HDFS. Die Verarbeitung von Gradoop erfolgt Batch-orientiert, das bedeutet die Daten werden aus Datenquelle gelesen, durch ein analytisches Programm verarbeitet und das Ergebnis wird in eine Datensenke geschrieben. Ebenso bietet Gradoop eine Java-Programmierschnittstelle, um analytischer Programme zu formulieren. Mit Gradoop ist es auch mögliche beliebige Datenformate automatisch in das eigene Datenmodell zu transformieren. Die Kombination mit bereits vorhandenen Flink-Bibliotheken ist auch möglich. Dem Benutzer sind somit keine Grenzen gesetzt, um seine Graph Datenbank so zu gestalten, wie er es möchte.

4. Graph-Datenbanksysteme

Mit diesen Systemen können die Graph Datenbanken umgesetzt und verwaltet werden.

4.1 Neo4j

Auf der Website von Neo4j wird mit 3 Punkten für dieses Programm geworben. Der erste Punkt ist Schnelligkeit, da man mit Hilfe von Cypher einfache Abfragen erstellen kann. Der nächste Punkt ist Skalierbarkeit. Dabei wird die Causal Clustering-Technologie erwähnt, welche eine hohe Verfügbarkeit und globale Skalierbarkeit bietet. Es soll sogar möglich sein, Diagramme mit 100 Millionen Knoten mit nahezu der gleichen Geschwindigkeit wie Diagramme mit Tausend Knoten abzufragen. Der letzte Punkt ist die Zuverlässigkeit. Dies ermöglicht Neo4j mit einer NoSQL Datenbank. Diesen Anspruch hat nicht nur das Unternehmen, sondern auch die Kunden. Ein weiterer

Vorteil von Neo4j ist die Sandbox Use Case, mit der man viele verschiedene Bausteine und Plugins für sein Projekt mit einbeziehen oder schreiben kann.

4.2 Oracle 12c Spatial and Graph/NDM Features

Dieses Programm wurde für die Verwaltung von räumlichen und grafischen Datenbanken entwickelt. Dabei werden erweiterte Funktionen für die Verwaltung und Analyse von Geodaten, sozialen Eigenschaftsdiagrammen und RDF – verknüpften Datenanwendungen verwendet. Für die Analyse ist es von großer Wichtigkeit, die Zusammenhänge zu verstehen. Allerdings stellen Neue Technologien wie Cloud-Dienste, mobiles Tracking, soziale Medien und Echtzeitsysteme neue Anforderungen an die Verwaltung des Datenvolumens. Deswegen wird für die räumliche Analyse eine breite Palette von Funktionen und Diensten von Oracle 12c Spatial and Graph bereitgestellt. Aktuelle Branchenangebote konzentrieren sich in der Regel auf die Bereitstellung von Analysen oder einer Diagrammdatenbank. Oracle Spatial 12c and Graph sind insofern einzigartig, denn sie bieten leistungsstarken In-Memory-Analysten mit wesentlich mehr integrierten Analysen an.

5. Graph-DB-Anfragesprache Cypher für Neo4j

Die Anfragesprache Cypher hat einen ähnlichen Syntaxaufbau wie SQL, allerdings können die Daten visuell angezeigt werden. Ein weiteres Feature ist das traversieren des Graphen, anstatt die Verknüpfung durch Joins zu realisieren. Hierbei wird die Geschwindigkeit der Abfragen durch die Indexierung der Knoten beeinflusst. Im Vergleich zur normalen SQL Abfrage gibt es hier allerdings ein paar Besonderheiten. Es wird MATCH für die Beschreibung der Struktur des gesuchten Musters verwendet, anstatt des gewohnten SELECT. WHERE hat hierbei die Funktion zusätzliche Bedingungen für die Muster zu definieren.

6. Interessante Anwendungen für Graph Datenbanken

Die meisten Anwendungen basieren auf sozialen Netzwerken und Semantik Web, da diese mit Graph Datenbanken viel schneller und einfacher umgesetzt werden können. Es sind auch einige eigene Ideen zu Anwendungen hinzugekommen. Ein Beispiel ist die Ahnenforschung, welche mit Graph Datenbanken viel besser angezeigt und verwaltet werden kann. Eine weitere Überlegung war für die Zweigstellen von Firmen. Im Konkretem wurde da an eine Supermarktkette gedacht. Dabei werden zu jeder Fiale verschiedene Eigenschaften für den Standort gespeichert, wie zum Beispiel die Größe der Fiale, ob die Fiale an der Hauptstraße gelegen ist oder wie die Öffnungszeiten der Fiale sind.

7. Vorstellung und Umsetzung eigener Graph Datenbank

Hierbei ging es um ein Projekt aus dem 1. Semester des Master Studiengangs. Die Aufgabe war es den Abstand zwischen 2 Schauspielern mittels Breitensuche zu ermitteln. Also 2 Schauspieler, A und B, welche im selben Film mitgespielt haben, kennen sich direkt und haben somit den Abstand 0 zueinander. Schauspieler B hat aber noch in einem 2. Film mitgespielt und kennt daher Schauspieler

C. Somit haben Schauspieler A und C den Abstand 1 zueinander, da sie sich über den Schauspieler B kennen können. Im laufenden Projekt hat sich herausgestellt, dass die Datenbank nicht nur Schauspieler, sondern alle Mitarbeiter beim Film enthält, daher hat sich die Aufgabenstellung etwas geändert. Welchen Abstand hat ein Mitarbeiter zu einem anderen Mitarbeiter? Dabei galt es die Laufzeiten des Suchvorganges zu erfassen und zu Dokumentieren. Aufgrund zu langer Laufzeiten sollte eine Optimierung vorgenommen werden. Eine Graph Datenbank, wobei jeder direkte Nachbar eines Knotens (Mitarbeiter) in einer neuen Datenbank gespeichert wurde, war da die Lösung. Somit war es möglich die Laufzeit auf ein Zehntel zu reduzieren.