hochschule mannheim

BDEA SS23

Aufgabe 3: NoSQL Szenario (Social Network)

Gruppe B

Maximilian Broszio | Marvin Karhan | Rafael Kosiel

Dateien und Inhalte

- Der Ordner [`Queries`] enthält die Textdateien mit den SQL-Abfragen.
- Der Ordner [`Source`] enthält den SourceCode. Dieser ist auch öffentlich unter https://github.com/derfel-c/bdea-a3-social-network einsehbar (inclusive denen hier genannten dokumenten).
- Das Dokument [`lessons-learned.pdf`] enthält die Lessons Learned.
- Dieses Dokument [`BDEA_GruppeB_A3.pdf`] enthält die zugehörigen Dokumentationen der Aufgabe.

Quick Start Guide

1. `Terminal`
   Initialisiert & Startet das ArangoDB Cluster
   1. `docker compose up -d`
   2. `pip install -r requirements.txt`
   3. `cd backend`
      Initialisiert die Daten
   4. `flask ––app . run`
2. `Terminal`
   1. `cd frontend`
      Started das Frontend zum testen der querys
   2. `ng serve -o`

Datenbank, Datenmodell und Queries

Datenbank

Bei der Suche nach einer geeigneten Datenbank für die Aufgabenstellung ist unsere Wahl auf ArangoDB als Datenbank gefallen.
Wie aus der Vorlesung bekannt ist, eignen sich Graphdatenbanken besonders gut zur Darstellung von Verbindungen und Abhängigkeiten von Daten, wie es in der Aufgabe mit den “Likes” der Fall ist.
Eine Recherche nach eben jenem Typ von Datenbank (via db-engines.com) lieferte uns einen ersten Einstiegspunkt in die vorhandene Auswahl an Graphdatenbanken, vgl. Abbildung 1.

Rang			DBMS	Datenbankmodell	Punkte		
Jun 2023	Mai 2023	Jun 2022			Jun 2023	Mai 2023	Jun 2022
1.	1.	1.	Neo4j	Graph	52,77	+1,66	-6,76
2.	2.	2.	Microsoft Azure Cosmos DB	Multi-Model	36,57	+0,58	-4,41
3.	3.	3.	Virtuoso	Multi-Model	5,24	-0,33	-0,93
4.	4.	4.	ArangoDB	Multi-Model	4,89	+0,01	-0,61
5.	5.	5.	OrientDB	Multi-Model	4,53	+0,03	-0,33
6.	6.	6.	Amazon Neptune	Multi-Model	3,03	+0,13	+0,21
7.	7.	↑ 8.	JanusGraph	Graph	2,83	+0,15	+0,43
8.	8.	↑ 19.	Memgraph	Graph	2,82	+0,18	+2,38
9.	9.	↑ 15.	NebulaGraph	Graph	2,72	+0,11	+1,61
10.	10.	↓ 7.	GraphDB	Multi-Model	2,55	+0,07	-0,07
11.	11.	↓ 9.	TigerGraph	Graph	2,20	+0,17	+0,18

Abbildung 1: Top 10 Graphdatenbanken gemäß www.db-engines.com, Juni 2023

Bei der Erstplatzierung Neo4j handelt es sich um eine reine Graphdatenbank. Da wir vor Antritt der Aufgabe nicht recht wussten, was uns alles erwarten wird, haben wir uns dazu entschieden lieber auf eine der in Abbildung 1 gelisteten populären Multi-Model Datenbanken zu setzen. Microsoft Azure Cosmos DB schied aus, da hierfür eine kommerzielle Lizenz benötigt wird. Die aktuelle Version von Virtuoso aus dem Jahr 2022 ist schon etwas älter, wohingegen die nächste Platzierung ArangoDB eine aktuelle Version vom März 2023 bietet.
Da ArangoDB als Multi-Modell Datenbank neben Graphen auch eine dokumentenbasierte Datenhaltung bietet, fiel unsere Wahl kurzerhand auf ArangoDB.
ArangoDB wird mit 2 Datenbank Knoten, einem Agent und einem Coordinator verwendet. Das Setup findet via docker-compose statt. Dabei ist der Coordinator für das Verteilen der Daten und den Datenimport via CLI zuständig.

Datenmodell

Abbildung 2 skizziert den Aufbau unserer Daten. Hierbei sei angemerkt, dass ArangoDB für jede Collection einen _key und einen _id Eintrag erzeugt (vgl. https://www.arangodb.com/docs/stable/indexing-index-basics.html). Diese kann man auch manuell überschreiben. Die _id hat als Präfix immer den Collection-Name z. B. user/id, während der _key nur aus der id besteht.

Die Entitäten “users” und “tweets” sollten gemäß Aufgabenstellung selbsterklärend sein. Bei beiden handelt es sich um dokumentenbasierten Collections. Neben den notwendigen Angaben “_key” und “_id” haben User in unserem Fall lediglich einen zufallsgenerierten Namen. Die Entität “tweets” entspricht dem Aufbau der tweets.csv Datei, mit dem Unterschied, dass wir die Angabe “number_of_likes” vernachlässigen und diese stattdessen über die Entität “likes”, in Form einer Edge-Collection, realisieren. Letztere enthält im Feld “_from” die “_id” eines “users” und im Feld “_to” analog dazu die “_id” eines “tweets”.
Die Entität “wrote” repräsentiert die “wer hat den Tweet verfasst” Relation. Die Entität “cache” stellt den Fan-Out-Cache dar und cached die Tweets, die einem User auf der

Startseite angezeigt werden sollen. Diese beiden Entitäten werden ebenfalls als Edge-Collections realisiert.
Die Entität “follows” stellt die Follower-Beziehung zwischen zwei Usern dar; z. B. User A (“_from”) folgt User B (“_to”) und ist auch eine Edge-Collection. Die “_to” und “_from” Felder der Edge-Collection sind standardmäßig indexiert, was bessere Lese-Performance ermöglicht.

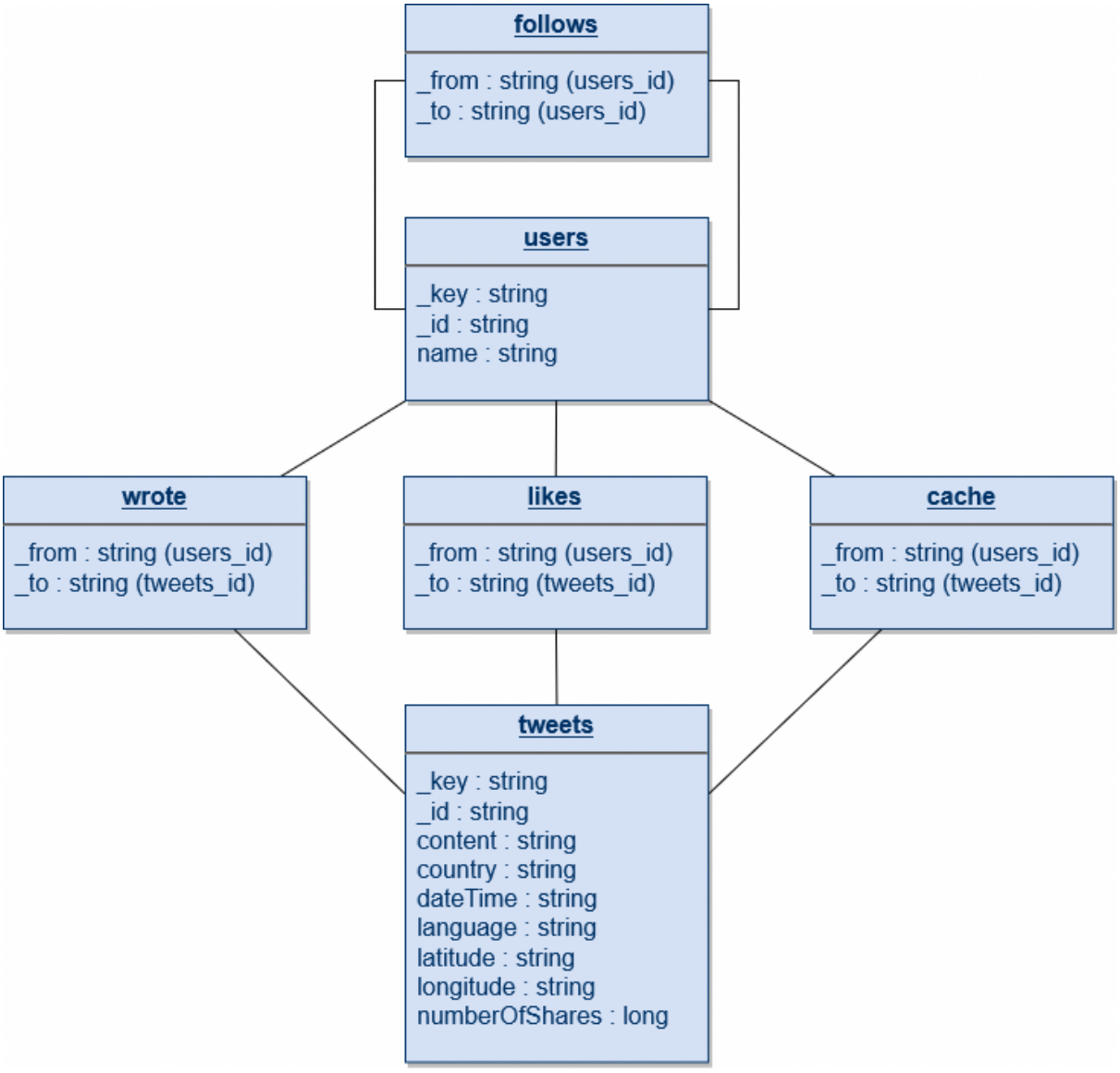

Abbildung 2: Datenmodell

Zudem haben wir anhand des in Abbildung 2 beschriebenen Datenmodells einen Graphen definiert, um zu testen, ob das positive Auswirkungen auf die Performance hat oder die Queries einfacher zu schreiben sind. Die Definition eines Graphen in AQL, basierend auf unserem Datenmodell sieht wie folgt aus:

```
LET likesEdgeDefinition = {
  "collection": "likes",
  "from": ["users"],
```

```
  "to": ["tweets"]
}

LET wroteEdgeDefinition = {
  "collection": "wrote",
  "from": ["users"],
  "to": ["tweets"]
}

LET fanOutEdgeDefinition = {
  "collection": "cache",
  "from": ["users"],
  "to": ["tweets"]
}

LET followsEdgeDefinition = {
  "collection": "follows",
  "from": ["users"],
  "to": ["users"]
}

INSERT {
  "_key": "TwitterGraph",
  "edgeDefinitions": [likesEdgeDefinition, wroteEdgeDefinition,
fanOutEdgeDefinition, followsEdgeDefinition]
} INTO _graphs
```

Da wir bereits alle Querys geschrieben haben, bevor wir einen Graph definiert haben, haben wir nur die Querys 1. und 2. mit der Graphtraversal-Syntax (vgl. https://www.arangodb.com/docs/stable/aql/graphs-traversals.html) getestet.

Daten-Initialisierung

Zu Beginn wird die Datenbank “twitter2” erstellt, falls diese noch nicht existiert. Anschließend werden die User erstellt. Dafür wird die twitter_combined.txt Datei zeilenweise gelesen. Für jede einzigartige Id wird ein User mit zufallsgenerierten Namen erstellt.

Für das Bilden der “follows” Relationen wird die gleiche Datei eingelesen. Die Relationen werden gebildet und in Batches in die Datenbank geschrieben.

Um die Tweets zu erstellen, wird die Datei tweets.csv eingelesen und mit Pandas formatiert, sodass Zahlen und das Datum richtig repräsentiert werden. Anschließend werden diese in der Datenbank gespeichert.

Für den Aufbau der “wrote” Relationen werden zuerst über die query_users_with_most_followers query die user sortiert anhand der Followeranzahl besorgt. Die Tweets werden dann anhand des originalen Autors und des Index auf unsere erstellten User verteilt. Durch die vorherige Sortierung sind die User mit den meisten Followern auch die Verfasser der meisten Tweets.

Da bei der Like-Generierung zu viele Daten erzeugt wurden und dies zu einem Überlaufen des RAMs geführt hat, wurde dies auf das arangoimport tool anstatt der Arango REST-API ausgelagert. Dafür wird ein 5,5 GB großes JSON-File erzeugt, das dann über das arangoimport tool importiert wird.

Für die Erstellung des Fanout Caches werden die User, die Usern folgen, die Tweets verfasst haben, über die query_user_ids_where_followed_users_have_tweets abgefragt. Von diesen Usern werden dann die Tweets abgefragt und in den Cache gespeichert. Die Größe des Fanout Caches ist wie bei der Like-Generierung sehr groß und muss ebenfalls über das arangoimport tool geladen werden. Dafür wird zuerst ein 10 GB großes JSON-File erzeugt, das dann über das arangoimport tool importiert wird.

Queries

Die Queries befinden sich in der jeweiligen Textdatei im Queries Ordner, bzw. in der Python Datei ../backend/queries.py im Quellcode.

1. **Auflisten der Posts, die von einem Account gemacht wurden, bzw. ihm zugeordnet wurden**
    - → *query_posts_of_user*
    - Die Abfrage durchsucht alle Kanten, die von der spezifizierten Benutzer-ID in der 'users' Sammlung ausgehen, im Kontext des 'TwitterGraph' Graphen. Sie filtert diese Kanten auf diejenigen, die zur 'wrote' Sammlung gehören, was bedeutet, dass sie nur die Kanten berücksichtigt, die repräsentieren, dass der spezifizierte Benutzer einen Tweet geschrieben hat. Für jede dieser Kanten wird dann ein Objekt zurückgegeben, das den Benutzer (v) und den von ihm geschriebenen Tweet (DOCUMENT(e._to)) enthält. Die Graph Query ist ca. 50% schneller als die dokumentenbasierte Query.

2. **Finden der 100 Accounts mit den meisten Followern**
    - → *query_users_with_most_followers*
    - Zählung der IDs in “_to” innerhalb der Kantenrelation “follows”, sowie Sortierung und Begrenzung der Ergebnisliste. Leider ist die Performance der Graph Query hier deutlich schlechter, da beim Durchlaufen des Graphen immer wieder von dem User aus alle Kanten zu den Kantenrelation “follows” gezählt werden muss. Wohingeben bei der dokumentenbasierten Query direkt die “follows” edge collection mit dem “COLLECT” keyword verarbeitet werden kann.

3. **Finden der 100 Accounts, die den meisten der Accounts folgen, die in 2) gefunden wurden**

- → *query_top_followers_of_top_users*
- Filterung der Kantenrelation “follows” in “_to” auf die oben in [2] gefundenen User. Danach Vorgehen analog zu oben, mit dem Unterschied, dass hier “_from” statt “_to” einbezogen wird.

4. Auflisten der Informationen für die persönliche Startseite eines beliebigen Accounts; die Startseite soll Folgendes beinhalten:

4.1. die Anzahl der Follower

- → *query_follower_count_of_user*
- Filterung der Kantenrelation “follows” in “_to” auf die gegebene UserID und Zählung der Einträge als Ergebnis.

4.2. die Anzahl der verfolgten Accounts

- → *query_count_user_is_following*
- Vorgehen analog zu oben [4.1], mit dem Unterschied, dass hier “_from” statt “_to” einbezogen wird.

4.3. wahlweise die 25 neuesten (zeitbasiert) oder die 25 beliebtesten Posts (like-basiert) der verfolgten Accounts (per DB-Abfrage)

- → *query_top25_tweets_of_followed_user*
- Zunächst Bestimmung der User, denen gefolgt wird, in der Kantenrelation “follows”. Danach Bestimmung der Tweet IDs der von den gefolgten Usern verfassten Tweets in der Kantenrelation “wrote” via “_from”. Abschließend Filterung der Tweets basierend auf den im vorangegangenen Zwischenschritt erhaltenen TweetIDs und Sortierung des Ergebnisses nach Datum oder Like-Anzahl.

4.4. Caching der Posts für die Startseite (vgl. 4), erfordert einen sog. Fan-Out in den Cache jedes Followers beim Schreiben eines neuen Posts

- → *query_tweets_for_user_from_cache, insert_tweet_for_user_in_cache*
- Basierend auf der “_id” des users wird in der Cache Edge-Collection mit dem FILTER keyword nach diesem gesucht. Das Tweet-Ergebnis wird dann basierend auf dem Datum absteigend sortiert, sodass die neuesten Tweets auf der Startseite oben sind.

- Beim Erstellen eines neuen Tweets wird dieser zuerst in die Tweet Collection eingefügt. Danach werden alle user gesucht, die dem Verfasser des neuen Tweets folgen. Hierfür wird in der “follows” Edge-Collection das “_to” Feld verglichen. Für jeden gefundenen Eintrag werden neue Einträge im Cache, für jeden User der dem Autor folgt, mittels UPSERT angelegt. Außerdem wird die verknüpfung des Verfassers und dem Tweet in der “wrote” Edge-Collection gespeichert.

4.5. Auflisten der 25 beliebtesten Posts, die ein gegebenes Wort enthalten (falls möglich auch mit UND-Verknüpfung mehrerer Worte)

- → *query_top25_posts_containing_words*

- Filterung der Collection “tweets” in “content” nach den gesuchten Wörtern. Die Filterung erfolgt mittels des CONTAINS Befehls. Für die Überprüfung mittels Contains werden der Content und das gesuchte Wort in Lower-Case-Schreibweise verglichen. Das Ergebnis wird dann in absteigender Reihenfolge sortiert und ist auf die geforderten 25 Einträge beschränkt.

Extra-Queries

- query_user_ids_where_followed_users_have_tweets
    - Liefert die User, die Usern folgen, die Tweets verfasst haben.
- query_random_user_id_with_tweets
    - Liefert einen zufälligen User der Tweets verfasst hat
- query_random_user_id
    - Liefert eine zufällige User id
- query_user_by_tweet_id
    - Liefert den User der den Tweet geschrieben hat

Query caching

Manche Querys lohnt es sich zu cachen, dabei haben wir zusätzlich zum In-Memory Caching über die Cache Collection vereinzelt Application-Level Caching eingesetzt. Beispielsweise nutzen wir zum Demonstrieren der Querys eine Query, die uns einen zufälligen User gibt, welcher Usern folgt, die Tweets verfasst haben. Dafür muss eine aufwändige Query ausgeführt werden, die über alle User und deren Kantenrelation “follows” iteriert und die passenden User Collected. Diese Liste bleibt immer gleich und muss somit nicht neu generiert werden.