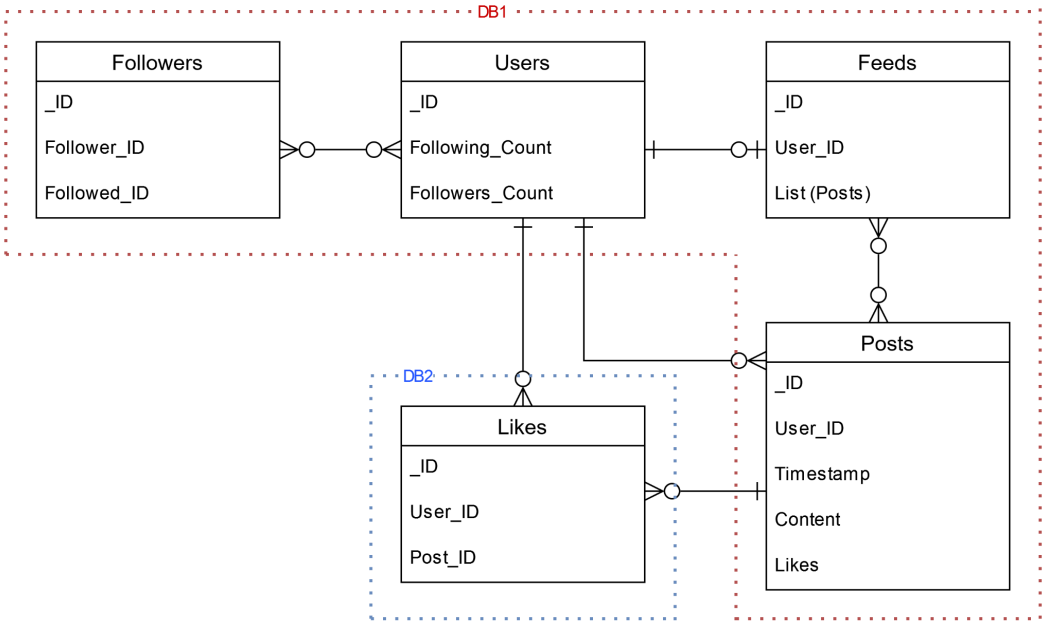
3. Aufgabe: NoSQL Szenario-Gruppe D

Jonas Blum, Alejandro Restrepo Klinge, Floris Wittner
SoSe 24 - Informatik HS Mannheim - Marcus Kessel
https://github.com/mrrestre/BDEA24_GruppeD_Aufgabe3

Datenbankmodell
Im Datenbankmodell sind die unterschiedlichen Collection und deren Verbindung untereinander zu sehen. Außerdem ist zu erkennen, in welchen Datenbanken die jeweiligen Collections gespeichert sind. Die Collections sind dabei auf zwei Datenbankinstanzen aufgeteilt.



A. Warum MongoDB

Die Datenbank muss in mehrere Docker aufgeteilt werden können. Zudem müssen Benutzer, Feeds, Likes und Posts gespeichert werden. Dabei gilt es zu beachten, dass Benutzer anderen Benutzern folgen und von anderen Benutzern gefolgt werden können. Ein Benutzer kann Posts liken und selbst welche verfassen. Sollte er anderen Benutzern folgen, sieht er deren Posts in seinem Feed.

Mithilfe einer grafischen Datenbank können die Verbindungen zwischen Benutzern, Likes und Posts dargestellt werden. Eine dokumentenbasierte Datenbank wie MongoDB würde kein starres Schema benötigen und der Feed könnte gut als Liste von Posts dargestellt werden.

Aufgrund der vielen Beispiele mit MongoDB und bereits bestehender Kenntnis mit dieser Datenbank, wurde nach kurzer Einarbeitung in die Aufgabenstellung und in Graphen- sowie Dokumenten Datenbanken MongoDB ausgewählt.

Datum: 03.07.2024

B. Wie werden die Daten eingefügt:

1. User und deren Verbindungen werden Spaltenweise aus der Inputdatei (twitter_combined.txt) eingelesen
    1.1 User Eintrag erstellt, falls dieser noch nicht in *Users Collection* vorhanden ist
    1.2 Verbindung der User untereinander werden in *Followers Collection* gespeichert
    1.3 Counter Following_Count und Followers_Counts werden in *Users Collection* aktualisieren

2. Posts in DB einfügen
    2.1 Alle Posts aus der Inputdatei tweets.csv werden zufällig den 100 meistgefolgten Users zugeordnet. Diese Users werden anhand der Followers_Count der *Collecion Users* ermittelt.
    2.2 Die Posts werden in der *Posts Collection* gespeichert.

3. Likes in DB einfügen
    3.1 In der *Collection Likes* werden zufällig pro Eintrag in der Posts Collection 0-100 Likes gespeichert.
    3.2 Der User, welcher dem Like zugeordnet wird, wird zufällig hinterlegt.
    3.3 Die Anzahl von Likes für einen Post wird in der Post in der *Post Collection* aktualisiert

4. Feed pro User erzeugen
    4.1 Für jeden Eintrag in der *Posts Collection* werden anhand der User_ID die Followers aus der *Followers Collection* ermittelt.
    4.2 Für jeden dieser Followers wird der in der *Posts Collection* gespeicherter Post in die Liste der jeweiligen *Feeds Collection* gespeichert.

C. Wie werden die Daten abgefragt

1. Finden der 100 Accounts mit den meisten Followern
    - In *User Collection* sortieren nach follower_count absteigend und top 100 Dokumente wählen

2. Finden der 100 Accounts, die den meisten der Accounts folgen, die in 1. gefunden wurden
    - Basierend auf der vorherigen Abfrage, wird in User Collection pro User gezählt, wie viele der in der ersten Abfrage gefundenen 100 Accounts die Person folgt. Die Top 100 werden ausgegeben

3. Anzahl von Follower
    - In user collection following_count auslesen

4. 25 neuesten Posts in Feed
    - User Feed finden (anhand von user_ID) Sortierung nach posts timestamp top 25

5. 25 meist geliketen Posts in Feed

- User Feed finden (anhand von user_ID) Sortierung nach posts likes top 25

6. Auflisten der 25 beliebtesten Posts, die ein geg. Wort enthalten (falls möglich auch mit UND-Verknüpfung mehrerer Worte)
   - Suche in der Post Collection (content) mit Regex pattern

Lessons Learned:

1. Tabellendefinition und Datenbankstruktur

Wir haben bemerkt, dass wir die Tabellenstruktur basierend auf den Abfragen am Ende definieren müssen. Bevor wir die Tabellenstruktur entsprechend angepasst hatten, gab es eine schlechte Performance und viele Aggregationen waren für eine Abfrage notwendig. Bei unserem Endergebnis sind die Collections so aufgebaut, dass sie als Priorität 1 eine performante Abfrage ermöglichen und als Priorität 2, dass das Einfügen von Daten performant funktioniert.

2. Performance beim Einfügen von (großen) Daten

Beim Einfügen von großen Datenmengen war die Codeoptimierung eine besondere Herausforderung, um eine akzeptable Ladezeit zu erreichen. Hierbei half bei uns unterschiedliche Herangehensweisen (z.B. Bulk Insertion, Multithreading, Pipelines, Zwischentabellen). Außerdem hilft auch das Vorsortieren beziehungsweise das Preprocessing von Daten vom Einfügen in die DB an vielen Stellen.

3. Arbeiten mit Indices

Obwohl die allgemeine Regel gilt, dass Indices in MongoDB das Einfügen von Daten verlangsamen, haben wir teilweise gemerkt, dass die richtige Wahl von Indices manche Schreiboperationen schneller gemacht haben. Das half insbesondere bei Operation, bei welchen Daten aus unterschiedlichen Collections gleichzeitig gefragt werden müssen. Beispielsweise werden bei der Feed-Erzeugung die Collections Posts, Followers und Feeds sehr oft abgefragt.

4. Limitationen in MongoDB

Wir haben bemerkt, dass die maximale Dokumentengröße in MongoDB 16 MB beträgt. Deshalb ist es bei Operationen mit Pipelines in Schritte, wo die Verarbeitungsdaten größer werden können (lookup, project) notwendig, die Daten in geeignete Zwischenspeicher zu bringen. Nachdem die Pipelines gelaufen sind, können diese wieder in den richtigen Collections zusammen aggregiert werden.