Vergleich von SQL und NoSQL

Datenbankdesign und Implementierung

Fachhochschule Nordwestschweiz

Joël Grosjean

17. Januar 2022

Contents

List of Figures

Meine Problemstellung

Ich habe mich entschieden MongoDB mit SQL zu vergleichen. Ich habe mich für MongoDB entschieden, weil MongoDB die vermutlich bekannteste NoSQL-Datenbank ist. Da MongoDB ein sehr breites Anwendungsgebiet hat und ich MongoDB auch in der Arbeitswelt vermutlich mal begegnen werde, ergab es Sinn MongoDB etwas genauer anzuschauen.

Um SQL mit MongoDB zu vergleichen, will ich eine Datenbank erstellen, welche die Metadaten von Blogartikeln speichert. Dies soll auch die Kommentare und Tags eines Blogartikels umfassen. Somit hat die SQL Datenbank also 3 Tabellen. Ich sehe die Vorteile von MongoDB vor allem bei der Abfragezeit, weil zum Darstellen auf eines Blog-Posts genau ein gesamtes Dokument abgerufen werden muss. Dies ist bei einer Blog-Webseite wohl die häufigste Abfrage. Zudem sind Abfragen so auch sehr simpel. MongoDB ist allgemein für die Schnelligkeit bekannt und verspricht somit besonders gute Ergebnisse. Weitere Vorteile sind vermutlich die Flexibilität und Einfachheit von MongoDB und ganz allgemein die Implementationszeit.

Meine Vorkenntnisse

Ich habe keine berufliche Erfahrung mit Datenbanken. Jedoch habe ich in dem Data Science Studium schon ein paar SQL-Datenbanken genutzt. Ich habe ebenfalls in der Form des Modul GDB schon mit Datenbanken Erfahrungen sammeln können. Jedoch habe ich vor dieser Abgabe noch keine eigenen Datenbanken aufgesetzt und eigene Daten eingelesen.

Konzeptionelles Datenmodell

Ich habe versucht ein ERD (Entity Relationship Diagram) mithilfe von Visio zu erstellen. Leider fehlte mir dafür die nötige Lizenz. Deshalb habe ich dann das ERD mithilfe von Microsoft Word erstellt. Dies ist suboptimal, doch funktionierte hierfür einwandfrei. Hierbei geht es grundsätzlich darum zu sehen, welche Daten in unserer Datenbank gespeichert werden und in welchem Zusammenhang sie mit den restlichen Daten stehen.

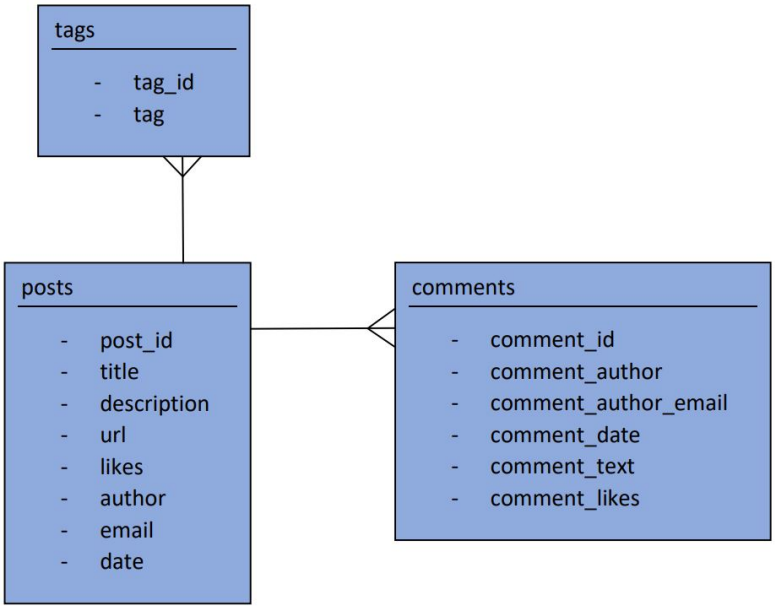

Figure 1: Entity Relationship Diagram

Die Datenbank lässt sich demnach mit drei simplen Tabellen implementieren. Die erste Tabelle speichert die Metadaten der Posts selbst. Diese beinhalten übliche Metadaten für Blogposts wie zum Beispiel URL, Titel, Anzahl Likes und Datum der Veröffentlichung. Weiterhin gibt es eine Tabelle für die Tags. Darin wird nur der Hashtag gespeichert. Dieser muss bei SQL einfach korrekt mit einem Post verbunden werden, aber mehr dazu kommen wir noch bei der Umsetzung. Die Datenbank hat ebenfalls eine Tabelle für Kommentare. Dies sind die Kommentare, welche unterhalb eines Posts geschrieben werden können. Somit kann es von null bis hin zu vielen Kommentaren pro Post geben. Auch hier speichern wir die nötigsten Informationen wie zum Beispiel Autor, Datum der Veröffentlichung und natürlich den Text des Kommentars.

Meine Umsetzung

Ich habe die beiden Datenbanken mithilfe von drei Python-Scripts aufgesetzt. Das Script ‘create_database.py‘ erstellt die Datenbanken. Bei SQL habe ich mich für eine PostgreSQL Datenbank entschieden. Im ersten Schritt muss hier eine Verbindung mit pgAdmin aufgebaut werden. Dies habe ich mithilfe des Packages ‘psycopg2‘ gemacht. Danach wird eine Datenbank aufgesetzt und die nötigen Tabellen erstellt. Dabei müssen die Struktur der Tabellen und die Verbindungen zwischen den Tabellen mitgegeben werden. Dies ist zeitaufwendig und braucht etwas Planung, um einwandfrei zu funktionieren. Das zuvor erstellte ERD ist hier sehr praktisch. Um die Datenbank zu visualisieren, habe ich bei der PostgreSQL Datenbank mithilfe der Applikation DbVisualizer vollautomatisch ein UML-Diagramm erstellt. Dieses UML-Diagramm ist das relationale Modell. Darauf sind die drei Tabellen mit den Datentypen der einzelnen Spalten zu sehen. Es ist ebenfalls ersichtlich, wie die Foreign-Keys mit den Primary-Keys verbunden sind.

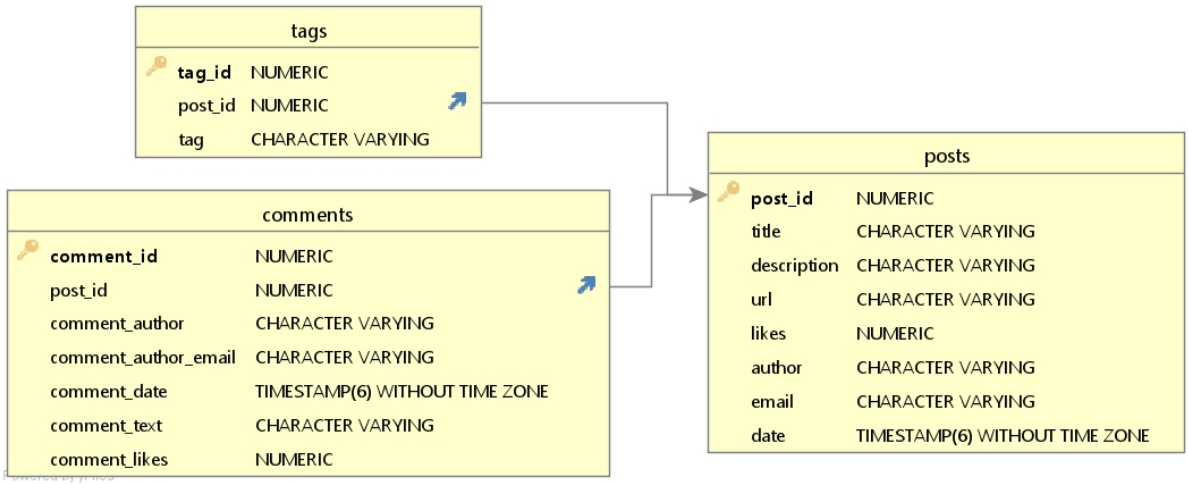

Figure 2: UML

Für MongoDB muss ebenfalls eine Verbindung mit dem Cluster aufgebaut werden und danach muss eine Collection erstellt werden. Dafür nutzte ich das Package ‘pymongo‘. Die Verbindung wird mit nur 3 Zeilen Code aufgebaut. Danach braucht es nur eine weitere Zeile Code, um alle Blogposts hinzuzufügen. Dabei kann die Struktur von verschiedenen Dokumenten unterschiedlich sein und muss auch nicht angegeben werden. Um die MongoDB Datenbank zu visualisieren, habe ich ein Beispieldokument, welches alle möglichen Variablen enthält, herauskopiert. Dieses zeigt auf, wie ein solches Dokument aufgebaut ist.

```
{
  "_id": { "$oid": "61d4a8b47e51883717060e1a" },
  "post_id": { "$numberInt": "4" },
  "title": "Here buy TV at anyone.",
  "description": "East despite work perform first.",
  "url": "http://www.clark.com/",
  "likes": { "$numberInt": "40495" },
  "author": "williamsjennifer",
  "email": "rodgersheather@yahoo.com",
  "date": { "$date": { "$numberLong": "1614290590000" } },
  "tags": ["#Here", "#buy", "#TV", "#at", "#anyone"],
  "comments": [
    {
      "comment_id": { "$numberInt": "37" },
      "comment_author": "timothymayer",
      "comment_author_email": "isaiah02@hotmail.com",
      "comment_date": { "$date": { "$numberLong": "1635125256000" } },
      "comment_text": "Into.",
      "comment_likes": { "$numberInt": "11" }
    },
    {
      "comment_id": { "$numberInt": "38" },
      "comment_author": "fowlertodd",
      "comment_author_email": "gholt@hotmail.com",
      "comment_date": { "$date": { "$numberLong": "1605428573000" } },
      "comment_text": "Be.",
      "comment_likes": { "$numberInt": "772" }
    }
  ]
}
```


Die Daten wirken teilweise, wie beispielsweise bei der URL sinnfrei. Dies liegt daran, dass die Daten mithilfe des ‘faker‘ Packages erstellt wurden. Wichtig war, dass die synthetischen Daten das richtige Format hatten, der genaue Inhalt war mir dabei egal. Für das Erstellen der synthetischen Daten nutzte ich das File ‘create_data.py‘. Dieses erstellt dieselben Daten für MongoDB und PostgreSQL im korrekten Format zur Befüllung der Datenbanken.

Vorteile von MongoDB

Ein wichtiger Vorteil gegenüber SQL ist die dynamische Entwicklung und die hohe Skalierbarkeit der Datenbank von einzelnen Servern bis hin zu komplexen Architekturen über mehrere Rechenzentren. Da die Dokumente sehr unterschiedlich sein können, ist man in der Weiterentwicklung der Datenbank extrem flexibel. Bei einer Website mit Blogposts ist es möglich, dass man später die Website erweitern will und somit neuere Blogposts andere Metadaten als ältere Blogposts haben. Dies ist für MongoDB kein Problem, während es bei SQL eher zeitaufwendig ist.

Auch die Struktur ist Frontend tauglich, weil sie JSON (respektive BSON) ist. Dies erleichtert die Implementation erheblich. Generell habe ich erwartet, dass der Entwicklungsaufwand kleiner ist als bei SQL und dass MongoDB weniger Zeilen Code benötigt. Ich sehe bei diesem Projekt die Vorteile von MongoDB vor allem bei der Abfragezeit, weil zum Darstellen auf eines Blog-Posts genau ein gesamtes Dokument abgerufen werden muss. Zudem sind Abfragen so auch sehr simpel. Bei den Abfragen sind ebenfalls keine Joins nötig, was die Abfragen theoretisch weniger komplex macht.

Messkriterien

Nun will ich herausfinden, wie gut die MongoDB Datenbank tatsächlich ist und will den Unterschied zu SQL messen. Ich habe mich entschieden, die beiden Datenbanken anhand der folgenden Messkriterien zu bewerten:

- Die Abfragezeit: Wie lange brauchen die beiden Datenbanken, um mir die in einem allfälligen Frontend darzustellenden Daten zur Verfügung zu stellen? Hier will ich mehrere die beiden Datenbanken für mehrere Abfragen vergleichen.
- Die Anzahl Code-Zeilen: Wie viele Zeilen Code brauchte ich für die beiden Implementationen?
- Die Implementationszeit: Sie soll vergleichen, wie lange ich brauchte, um die beiden Datenbanken aufzusetzen.
- Der Speicherbedarf: Welche Datenbank braucht mehr Speicher?

Abfragezeit

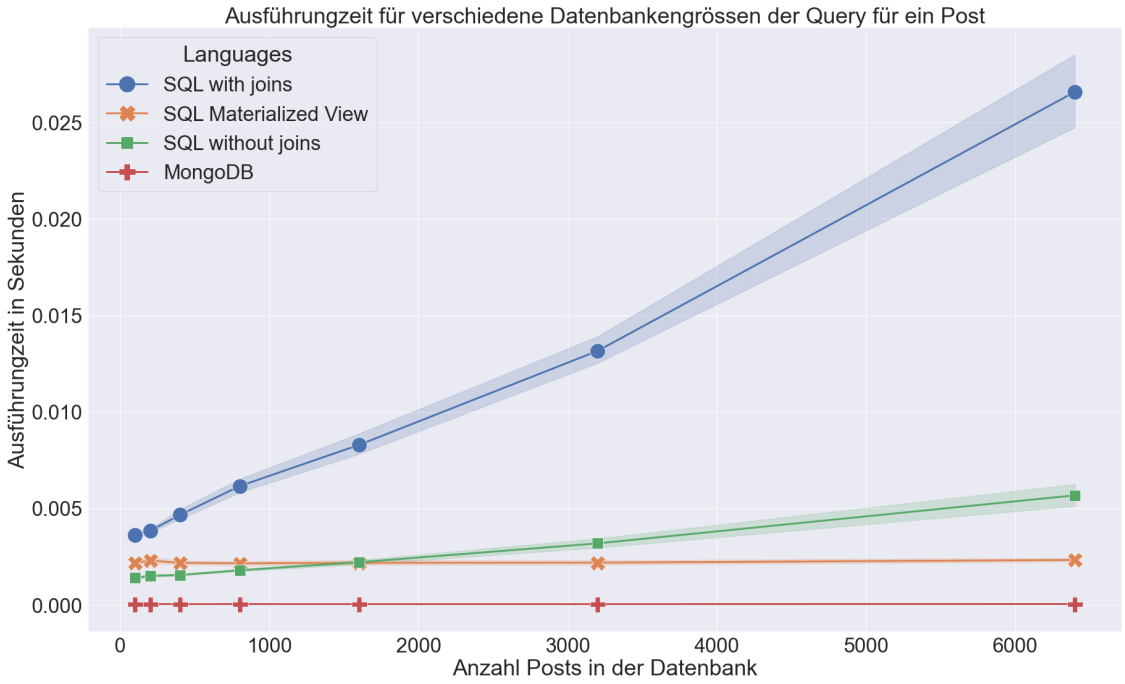

Figure 3: Query für einen Post

Anzahl Code-Zeilen

Hier war der Unterschied wirklich enorm. Zum Erstellen der Datenbank brauchte ich mit SQL 88 Zeilen Code und mit MongoDB nur 10 Zeilen. Bei SQL kamen dann noch zusätzlich 57 Zeilen für Materialized-Views und Indexes dazu. SQL brauchte vor allem so viel Zeilen, weil ich die Struktur der Datenbank angeben musste, während ich bei MongoDB nur eine leere Collection erstellen musste. Gesamthaft brauchte SQL also mit 145 Zeilen Code gegenüber MongoDB mit 10 Zeilen Code fast 15-mal mehr Code.

Auch die Abfragen von MongoDB waren hier deutlich kürzer und simpler. Tatsächlich waren unsere MongoDB Abfragen nur eine Zeile lang, während die SQL-Abfragen mit den Joins schon mal ein Paar Zeilen

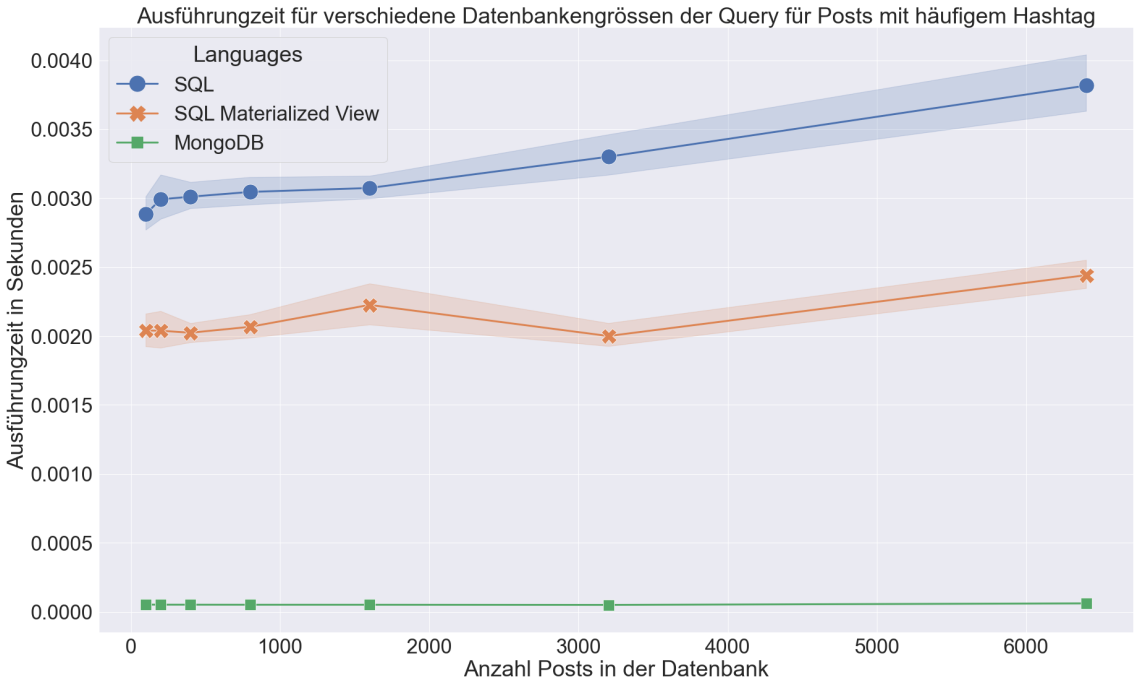

Figure 4: Query für Posts mit häufigstem Hashtag

Code brauchen konnten. Deutlich mehr Zeilen SQL-Code zu schreiben, braucht natürlich auch mehr Zeit, weshalb wir jetzt die Implementationszeit anschauen.

Implementationszeit

Auch bei der Implementationszeit war MongoDB klar besser als SQL. Obwohl ich vorher noch nie eine MongoDB Datenbank erstellt hatte, brauchte ich etwa 5-mal weniger Zeit zum Erstellen dieser Datenbank als bei SQL. Sobald die Datenbank verbunden war, konnte ich sie direkt befüllen. Dies war unglaublich praktisch und die Einfachheit kam etwas unerwartet. SQL steht in einem noch schlechteren Licht, wenn ich die Zeit zum Implementieren der Materialized Views und der Indexes dazuzähle. Insgesamt brauchte ich zum Implementieren der SQL Datenbank etwa 7-mal mehr Zeit als zum Implementieren der MongoDB Datenbank.

Speicherbedarf

MongoDB mit 6400 Dokumenten ist 13.77MB

MongoDB mit 6400 Dokumenten ist 69 MB mit den MVs MongoDB mit 6400 Dokumenten ist 11.5 MB

Fazit

Was hat meinen Lernerfolg beeinflusst und was ist die wichtigste Erkenntnis?

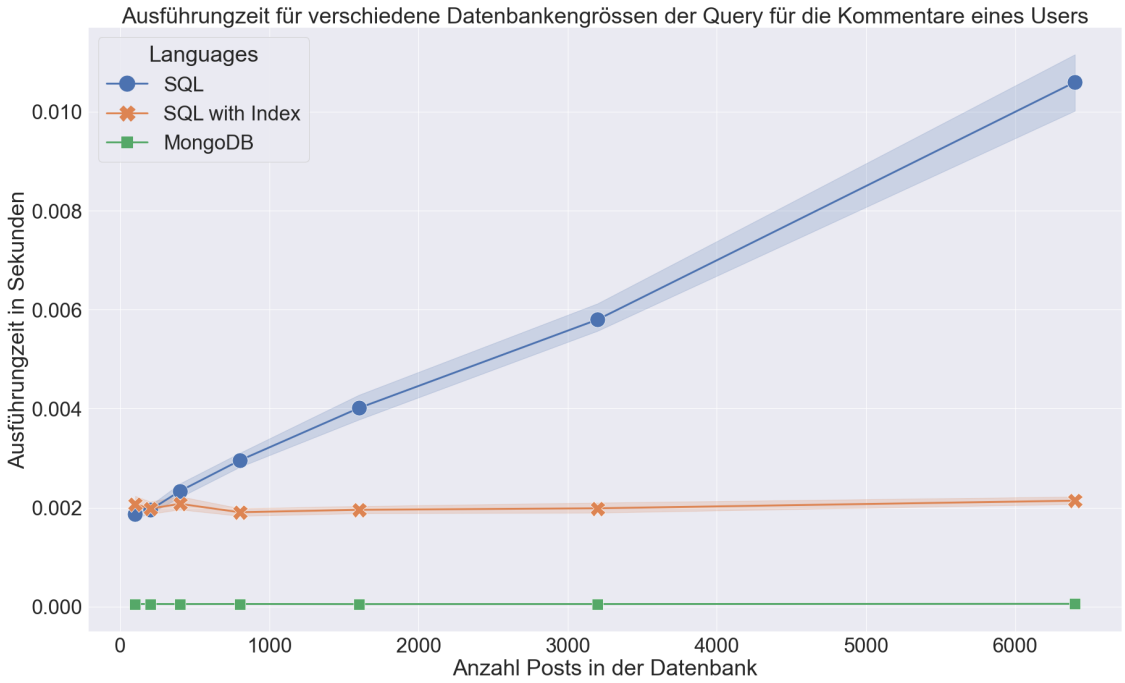

Figure 5: Query für die Kommentare eines bestimmten Users

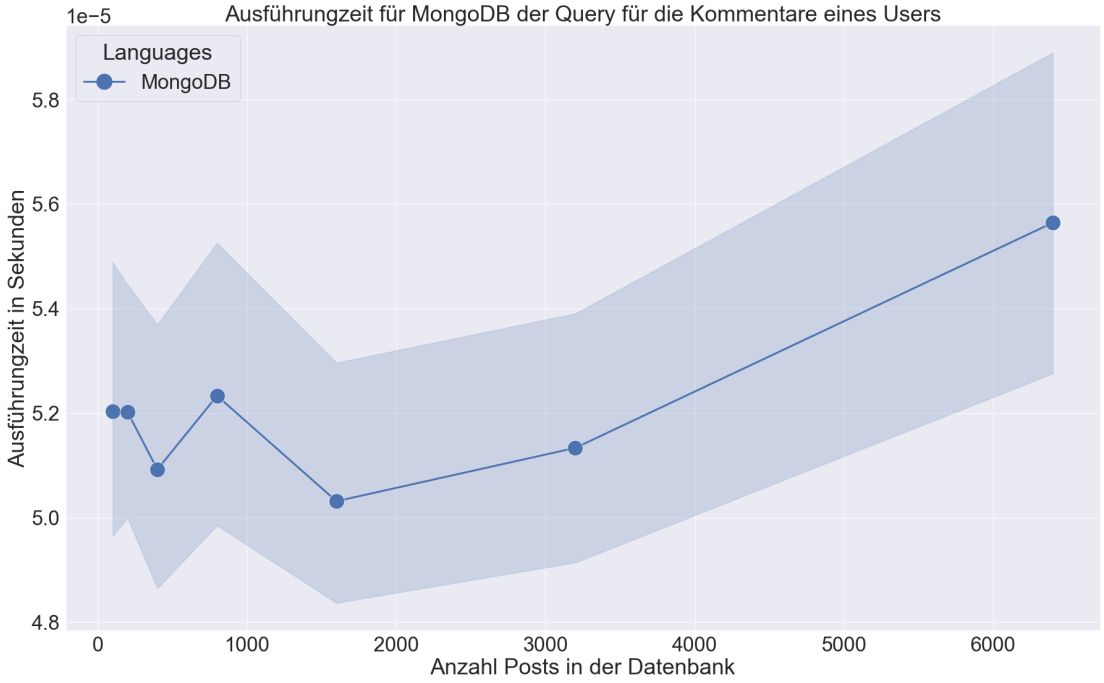

Figure 6: MongoDB Query für die Kommentare eines bestimmten Users