Vergleich von SQL und NoSQL

Datenbankdesign und Implementierung

Fachhochschule Nordwestschweiz

Joël Grosjean

18. Januar 2022

Inhaltsverzeichnis

Abbildungsverzeichnis

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 ERM (Entity-Relationship-Modell) mithilfe von Visio zu erstellen. Leider fehlte mir dafür die nötige Lizenz. Deshalb habe ich dann das ERM 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.

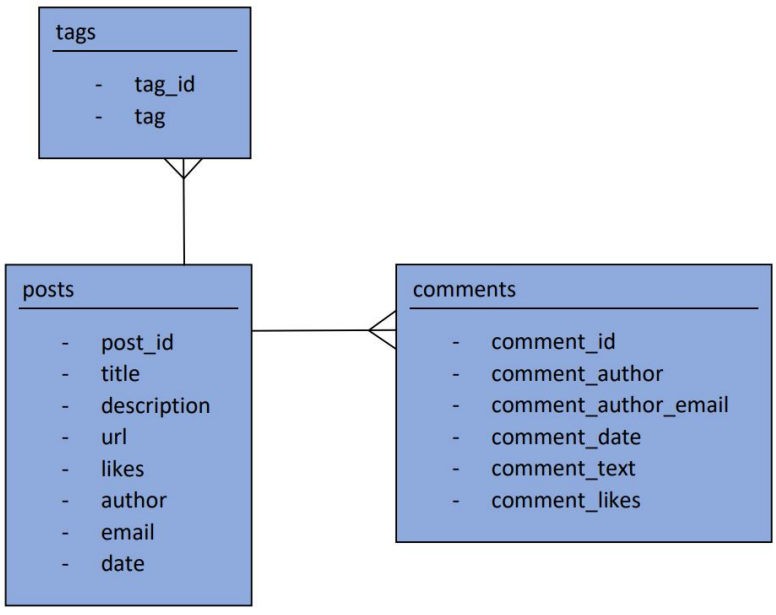

Figure 1: Entity-Relationship-Modell

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 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 ERM 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.

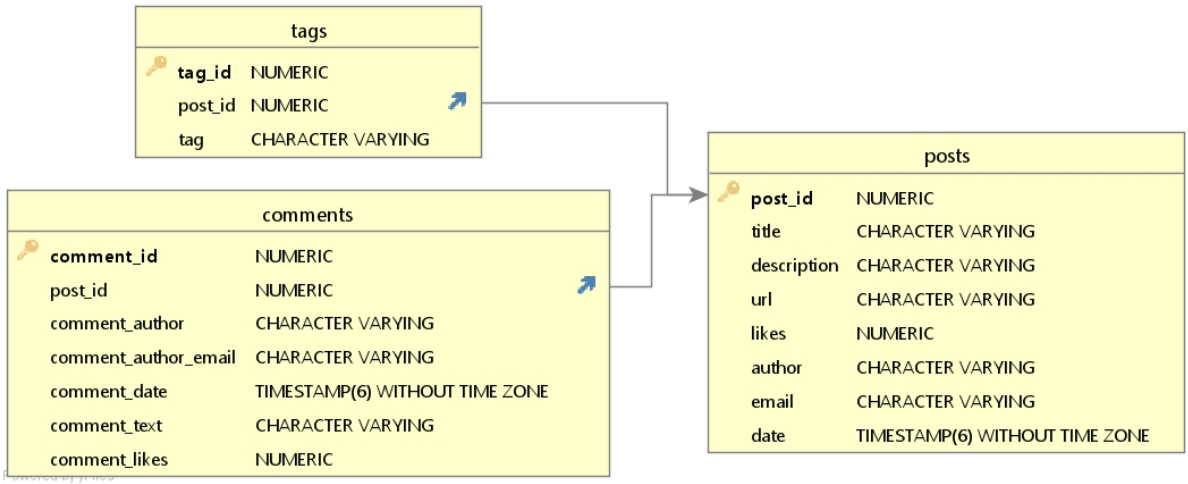

Figure 2: UML-Diagramm

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. Ein weiterer Vorteil von MongoDB ist, dass es gemäss

dem CAP-Theorem die Verfügbarkeit der Konsistenz vorzieht. Dies ist genau, was wir bei dieser Anwendung wollen.

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 die in einem allfälligen Frontend darzustellenden Daten zur Verfügung zu stellen? Hier will ich die beiden Datenbanken anhand mehrerer 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

Im File 'test_database.py' habe ich Queries erstellt, um die beiden Datenbanken zu vergleichen. Dabei habe ich die Zeit gemessen und diese in einem CSV-File gespeichert. Danach habe ich diese in einem Jupyter Notebook visualisiert.

Als Erstes habe ich die wohl häufigste Abfrage auf einer Blogging-Website erstellt: Alle Metadaten, die zu einer 'post_id' gehören, abzurufen. Für SQL bedeutet dies, dass die Daten entweder mit zwei Joins abgerufen werden oder dass 3 Abfragen zu je einer Tabelle geschickt werden.

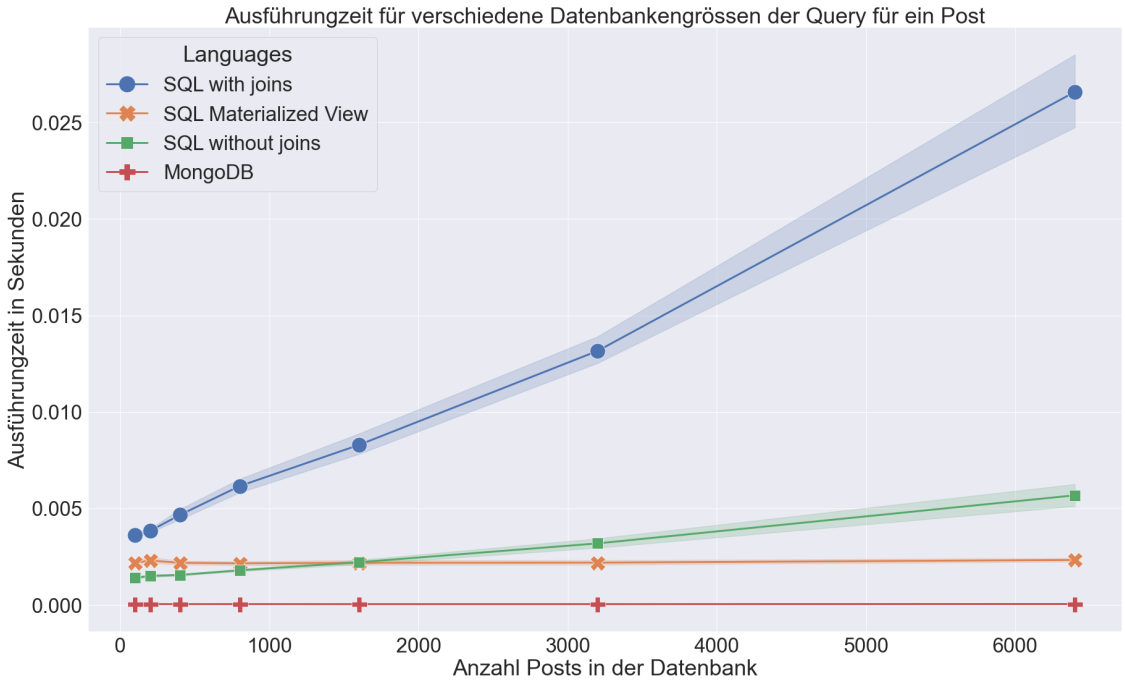

Figure 3: Query für einen Post

Als Erstes hatte ich nur die Abfrage mit Joins mit MongoDB verglichen. Dabei sieht man kaum, dass MongoDB über der Nulllinie ist. Deshalb hatte ich als Nächstes die SQL Version ohne Joins erstellt. Dies war schon um einiges besser, jedoch immer noch viel schlechter als die MongoDB Version. Danach hatte ich die Idee, mit einem Indexierten Materialized View zu arbeiten. Ich hatte zuvor noch nie einen Materialized View erstellt und brauchte deshalb relativ viel Zeit, um dies zu implementieren. Auch das Indexieren kannte ich bisher nur konzeptionell. Als die Abfrage schliesslich lief, fiel sofort auf, dass sie deutlich besser war als die beiden anderen SQL-Abfragen. Besser war sie vor allem deshalb, weil die Laufzeit viel langsamer stieg. Laut der Theorie hat eine indexierte Abfrage, welche auf dem Konzept des Binary Search Tree basiert, die asymptotische Komplexität O(log n) und dies scheint ungefähr mit den Messungen übereinzustimmen. Auch die Abfrage von MongoDB scheint in diese Komplexitätsklasse zu passen, da auch die Dokumente indexiert sind. MongoDB ist extrem schnell und immer noch viel schneller als unsere optimierte SQL Abfrage. Dies war aber auch die einfachste Abfrage für MongoDB und die schwierigste für SQL. Der Vergleich war also etwas einseitig. Deshalb schauen wir uns jetzt noch einige andere Abfragen an.

Nun wollen wir die Ausführungszeit für eine Abfrage, welche nach einem häufigen Hashtag sucht, untersuchen. Hier sollen also die Metadaten aller Posts, welche einen gewissen Hashtag in ihrer Tag-Liste haben, abgerufen werden. Die Kommentare können hier ausgelassen werden, da hier nur eine Übersicht der Posts erwünscht ist.

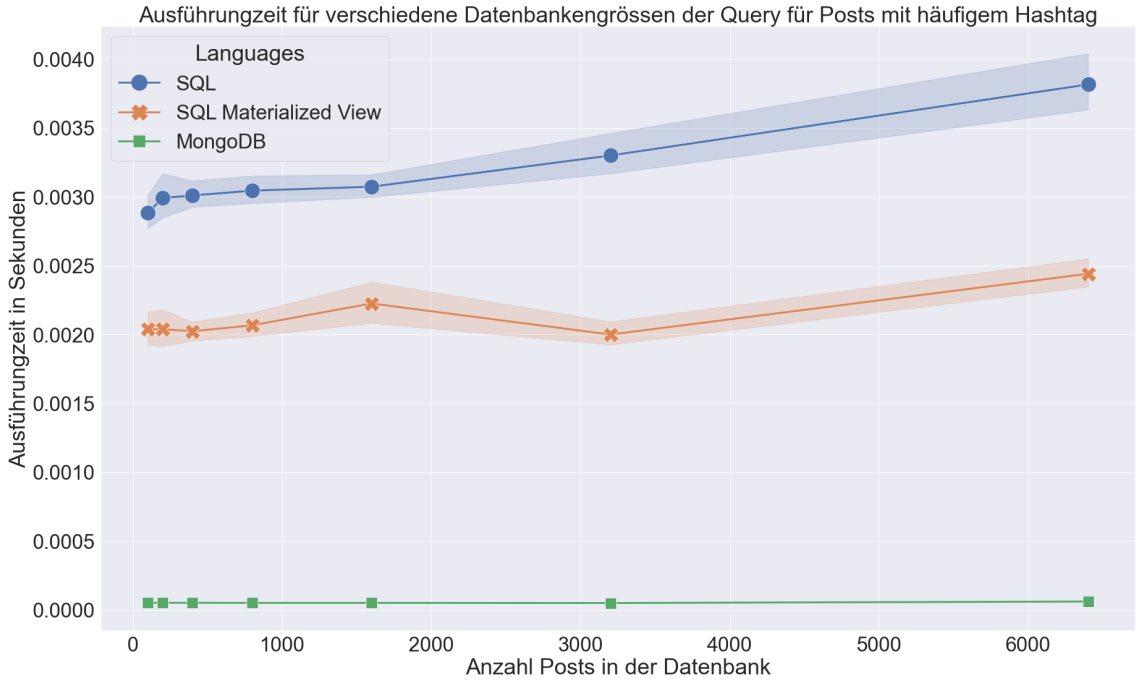

Figure 4: Query für Posts mit häufigstem Hashtag

Auch hier ist der Unterschied zwischen MongoDB und SQL sehr gross. Der Materialized View von SQL ist dabei etwa ein Drittel schneller als die einfache Version, während MongoDB mindestens 30-mal schneller ist.

Nun will ich noch die Ausführungszeit einer weiteren Query anschauen. Bei dieser Query wird nach allen Kommentaren, die ein User gemacht hat, gesucht. Dabei werden nur die Kommentare zurückgegeben. Für MongoDB bedeutet dies, dass bei jedem Dokument in den Kommentaren der Kommentar-Autor angeschaut werden muss.

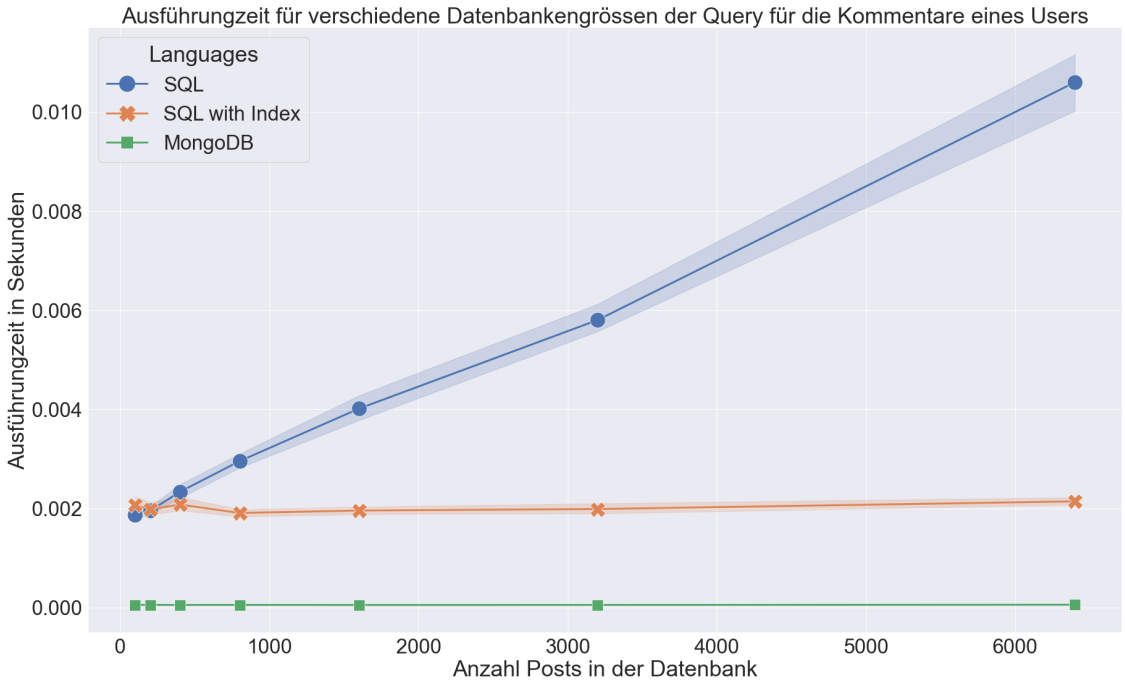

Figure 5: Query für die Kommentare eines bestimmten Users

Hier fällt direkt auf, wie riesig der Nutzen eines Indexes bei SQL sein kann. Die Laufzeit bei dieser Query steigt mit Index logarithmisch und somit wirklich kaum, während er vorher linear stieg. Unerwartet ist jedoch, dass auch hier MongoDB so viel besser ist, als zuvor erwartet.

Dies wirkt nun doch etwas unwahrscheinlich. Ich hatte erwartet, dass die Queries von MongoDB schneller sind, als jene von SQL, doch der Unterschied wirkt einfach zu gross. Deshalb habe ich mich entschieden, auch noch die Zeit zu untersuchen, welche die beiden Datenbanken für das Ausführen der Queries und zusätzlich das Speichern der Daten in einer Liste brauchen.

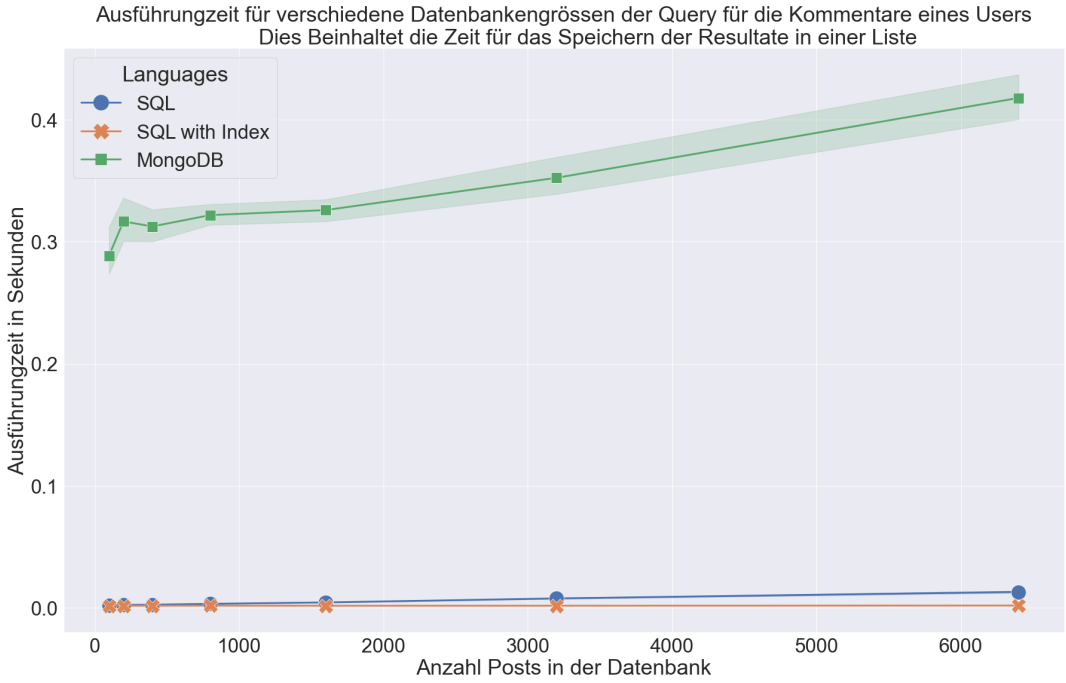

Figure 6: Query für die Kommentare eines bestimmten Users mit Speicherzeit

Nun sieht es genau umgekehrt aus: MongoDB ist deutlich langsamer als SQL. Schon für eine einfache Query auf nur 100 Einträgen braucht MongoDB hier 0.3 Sekunden. Dies wirkt extrem langsam und deshalb ebenfalls sehr unrealistisch. Es gilt natürlich zu bedenken, dass MongoDB nicht lokal läuft und dass bei den nun gemessenen Zeiten die Daten ebenfalls in eine Python Liste umgewandelt werden. Evtl. liegt diese Zeitdifferenz also an diesen Dingen. Diese Situation wirft die Frage auf, ob der Cursor von MongoDB die Daten schon von MongoDB herunterlädt oder ob dieser nur die Query beschreibt, die ausgeführt werden soll. Die Zeitmessungen scheinen auf letzteres hinzudeuten. Das heisst, dass die MongoDB Zeit wahrscheinlich nur gemessen werden kann, indem die Daten danach in eine Liste gespeichert werden.

Es ist tatsächlich recht schwierig, die Zeit zum Ausführen von Queries zwischen den beiden Datenbanken zu vergleichen. Da SQL auf meinem lokalen Computer läuft und MongoDB in der Cloud läuft, ist der Vergleich nicht wirklich vielsagend. Für mich stellt sich ebenfalls die Frage, welche Laufzeit genau verglichen werden kann. Am Ende ist ja wichtig, dass die Website läuft, ohne lange Ladezeiten im Frontend zu verursachen. Dies bedeutet, dass der beste Vergleich die Ladezeit im Frontend wäre. Hierbei müssten beide Datenbanken in der Cloud sein.

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 die MongoDB

Abfragen nur eine Zeile lang, während die SQL-Abfragen mit den Joins schon mal ein paar Zeilen 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. Dies macht für den Entwickler natürlich einen enormen Unterschied.

Speicherbedarf

Beim Speicherbedarf hatte ich erwartet, dass MongoDB im Nachteil ist, dies hat sich jedoch kaum bestätigt. Die Grösse der MongoDB Datenbank mit 6400 Dokumenten ist 13.77 MB. Die SQL Datenbank, welche mit denselben Daten befüllt ist, ist 11.50 MB gross. Der Unterschied hier ist also minim. Wenn man jedoch die Materialized Views und Indexes zu der SQL Datenbank hinzufügt, dann wächst sie zu einer Grösse von 69 MB an. Nun nimmt sie also etwa 5-mal so viel Speicherplatz in Anspruch wie die MongoDB Datenbank. Auch hier ist der Gewinner klar MongoDB.

Fazit

MongoDB hat mich für diesen Use Case allgemein überzeugt. Der Speicherbedarf ist gleich oder kleiner wie bei SQL. Man braucht deutlich weniger Zeit, um die Datenbank zu implementieren, und der Code dafür ist kurz und kompakt. Einzig bei der Abfragezeit bin ich mir nicht sicher, wie gut sie ist, weil meine Tests dafür versagten. Ich habe viele Zeit in diese Tests gesteckt, doch fiel mir erst gegen Ende auf, dass ich evtl. gar nicht dasselbe für beide Datenbanken gemessen habe. Wenn man jedoch den Nutzerberichten auf dem Internet glaubt, ist die Abfragezeit für beide Datenbanken sehr vergleichbar, wobei MongoDB besonders für das schnelle Schreiben auf die Datenbank bekannt ist.

Was mich besonders erstaunt hat, war die Einfachheit von MongoDB. Das Aufsetzten der Datenbank war sehr einfach und das Erstellen von den Queries war auch ziemlich simpel. Meine wichtigste Erkenntnis war also, dass MongoDB durchaus Vorteile gegenüber SQL haben kann. Im Besonderen überraschte mich also die Einfachheit von MongoDB. Für diesen Use Case würde ich also definitiv MongoDB gegenüber SQL vorziehen. Eine wichtige Erkenntnis war ebenfalls, dass es schwer ist, die Abfragezeiten von zwei so unterschiedlichen Datenbanken zu vergleichen. Was mich auch erstaunte, war wie gross der Nutzen des Indexierens und der Materialized Views bei SQL tatsächlich war.