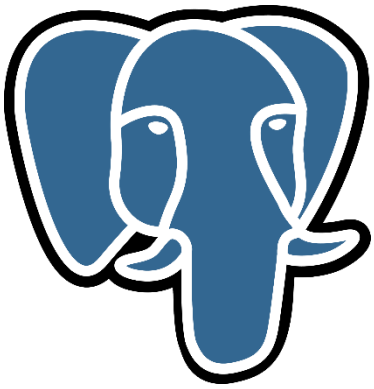
FH-Joanneum

Mobile Software Development (MSD21)

Datenbank für das MultimediaCentre Projekt



Semester:	WS23
Abgabe Datum:	30.06.2024, 23:55
Betreuer Professor:	Ulm Michael
Erstellt von:	Jeremia Ott

Inhaltsverzeichnis

Problemstellung

Datenbanken gibt es wie Sand am Meer, wo viele sich sehr ähnlich sind, weshalb man sie in die gleiche Kategorie packen kann. Auch wenn nur Kategorien anschaut werden, kann man leicht überfordert sein, wenn es darum geht die richtige Datenbank für einen bestimmten Fall zu finden.

Mit diesem Problem soll sich die Projektarbeit, anhand eines Fallbeispiels „MultiMediaCentre“ (siehe Motivation) beschäftigen, dabei sollen zwei Hauptprobleme gelöst werden:

1. Es soll möglich sein ein Verbinndung zu anderen Einträgen zu erstellen.
   Serien im westlichen Raum folgen einer Staffelstruktur. Bei Anime ist dies oft nicht der Fall. Hier ein paar Beispiele:
   - normale Struktur (Staffelstruktur): Staffel 1 => Staffel 2 => Staffel 3
   - Madoka Magica: Staffel 1 => Film => Film
   - Demon Slayer: Staffel 1 => Film (welcher später in ein Staffel 2 umgemünzt wurde) => Staffel 3 => Film (Recapfilm + Fortsetzung, dieser Film wurde später als eine Folge in Staffel 4 umgesetzt) => Staffel 4
   - Tokyo Ghoul:
     Staffel 1 => Staffel 2
     Manga => Staffel 3 => Staffel 4
     Dieser Anime basiert eigentlich auf einen Manga, schlägt aber am Ende der Staffel 1 einen anderen Weg ein als der Manga ein.
2. Die Datenbank soll so umgesetzt werden, dass es möglichst einfach ist sie um neue Medien zu erweitern.

Motivation

In der heutigen Zeit gibt es immer mehr Media das konsumiert werden kann. Deshalb ist es nicht verwunderlich, dass es immer schwieriger wird sich zu merken was bereits konsumiert wurde oder was noch in Zukunft konsumiert werden soll. Dieses Problem wurde bereits im Jahr 2006 erkannt. MyAnimeList, JustWatch oder auch IMDb was sogar noch älter ist, zeigen das ganz deutlich. Doch leider haben auch diese Seiten ihre Limits. MyAnimeList zum Beispiel, ist wie der Name schon vermuten lässt, nur auf Anime ausgerichtet. Wogegen JustWatch und IMDb ein wichtiges Feature, das besonders bei Anime wichtig ist, das Anzeigen von Verbindungen zu anderen Einträgen, nicht besitzt. Ein weiters Problem dieser Seiten, dass sie online sind und ohne Internet nicht funktionieren. Das Letzte Problem, ist das keine dieser Seiten unterstützt das nieder schreiben/speichern von Musik, Manhwa, Manga, Comics, Bücher und viele andere Medien.

Die zwei Hauptproblem sind das Anzeigen von zusammenhängenden Einträgen und die limitierten Typen von Einträgen (keine Bücher, Musik, usw.). Das Projekt, das diese Probleme lösen soll, ist „MultimediaCentre“, ein Projekt, welches es ermöglicht alle Medien, die man sehen, lesen oder hören möchte niederzuschreiben und in Zukunft einfach zu durchsuchen.

Da dieses Projekt den Rahmen einer Projektarbeit sprengen würde, wurde sich dazu einschienen in diesem Projekt nur den ersten Schritt zu tuen. Welcher das Erstellen der Datenbank darstellt.

Lösungsansätze

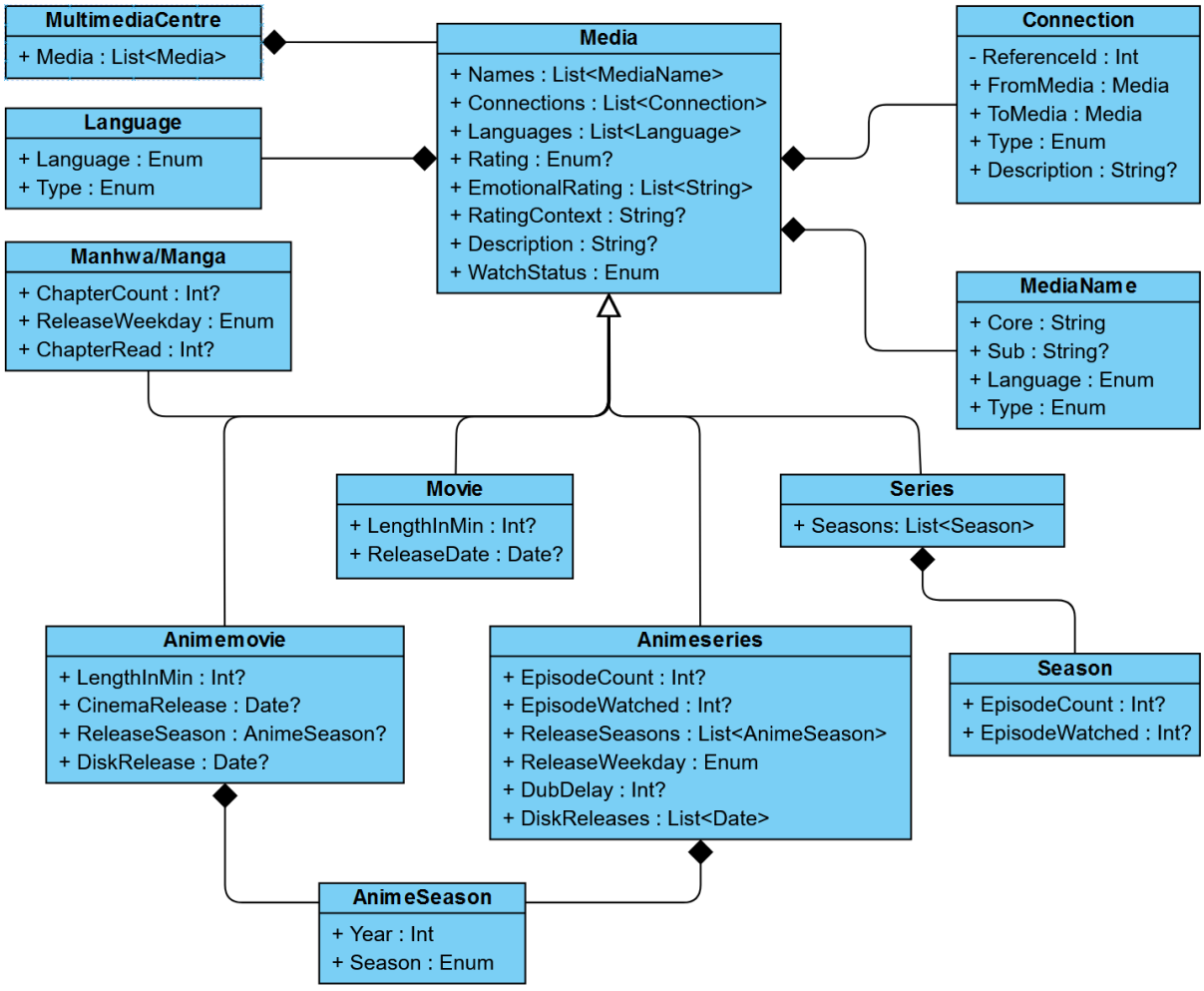
Für diese Projektarbeit werden drei mögliche Lösungsansätze umgesetzt.

1. Relational Datenbank (SQL Datenbank) welches die am meisten verbreitete Möglichkeit ist. Entschieden wurde sich dabei für eine PostgreSQL Datenbank. Als ORM wird Dapper verwendet, da es am leichtgewichtigsten ist.
2. Dokument Datenbank (NoSQL Datenbank) welches hierarchische Daten potenziell besser darstellen kann. Die Entscheidung ist dabei auf die bekannteste Dokument Datenbank gefallen, MongoDB.
3. Hybriddatenbank, welches eine Kombination aus Relational Datenbank und Dokument Datenbank ist. Die Synchronisation wird dabei von der Anwendung übernommen.

Diagramme

UML Klassen Diagramm

Zuerst habe ich ein Model mit C# Klassen erstellt. Jede Datenbank, die erstellt wurde, ist anhand dieses Modells erstellt worden.



Allgemeiner Kontext

Media:

- **Rating:** Für das Rating wird das Tierratingsystem verwendet.
  S => Es muss mindestens ein S-Tier Moment besitzen und durchschnittlich mindestens C-Tier, das heißt S-Tier kann durchschnittlich gesehen schlechter sein als A-Tier
  A => Muss durchschnittlich gesehen extreme Gut (A-Tier) sein
  B => Muss durchschnittlich gesehen Gut (B-Tier) sein, Oft sehr angenehm zu schauen.
  C=> Habe ich nicht bereut anzuschauen, würde ich aber nicht weiter empfehlen
  D=> Habe ich abgebrochen oder bereut anzuschauen.
- **EmotionalRating:** Hier werden die Media ein oder mehrere Emotionen zugeordnet die beim Schauen/Anhören/Betrachten empfunden wurden.
  z.B. Angst, Aufregung, Trauer, usw.
- **Rating und EmotionalRating** kann in Zukunft für ein Recommendationsystem verwendet werden.
- **RatingContext:** Hier kann noch weiterer Kontext zum Tier-Rating oder emotionalen Rating gegeben werden.
- **WatchStatus:** Gibt an ob Media im Status watched, ongoing oder none befindet.

Connection: Stellt Verbindungen zwischen Medien dar (SpinOff, OST, Fortsetzung, usw). Wird vor allem in Anime Bereich für Fortsetzungen verwendet.

- **ReferenceId:** Wird benützt für das schnelle Abfragen (Query) von allen Connections zu ermöglichen, die mit einer Media direkt oder auch indirekt zusammenhängen. Diese Id wird von der Anwendung verwaltet.
- **Type:** Gibt an was für eine Verbindung FromMedia und ToMedia haben. z.B. Fortsetzung, SpinOff.
- **Description:** z.B. Staffel 2, Staffel 3 oder sonstige Beschreibungen.

Language:

- **Type:** z.B. OmU (Sub), OV, Dub

MediaName: Name der Media.

- **Core:** Core Titel. z.B. Madoka Magica (Core)– Rebellion (Sub)
- **Sub:** Sub Titel. z.B. Madoka Magica (Core)– Rebellion (Sub)
- **Type:** Gibt an, ob ein Synonym, Original Title oder auch ob grade keine spezielle Art (None) verwendet wird.
- **Language:** Gibt die Sprache an. Kann aber auch keine Sprache sein (None) z.B. für Musik oder fast alles, was keine Filme oder Serien sind.

Series:

- **SeasonCount:** Im Westen ist es üblich eine Staffelstruktur zu haben (Staffel 1 => Staffel 2 => Staffel 3). Diese Struktur kann runtergebrochen werden auf die Anzahl an Staffeln (bei diesem Beispiel 3).
  Animeserien besitzen diese Staffelstruktur nicht immer und haben somit auch kein SeasonCount.

AnimeSeries: Der große Unterschied zu normalen Serien ist das hier oft keine Staffelstruktur vorliegt, was unteranderem einer der indirekten Hauptmotivationen für dieses Projekt (siehe Motivation).

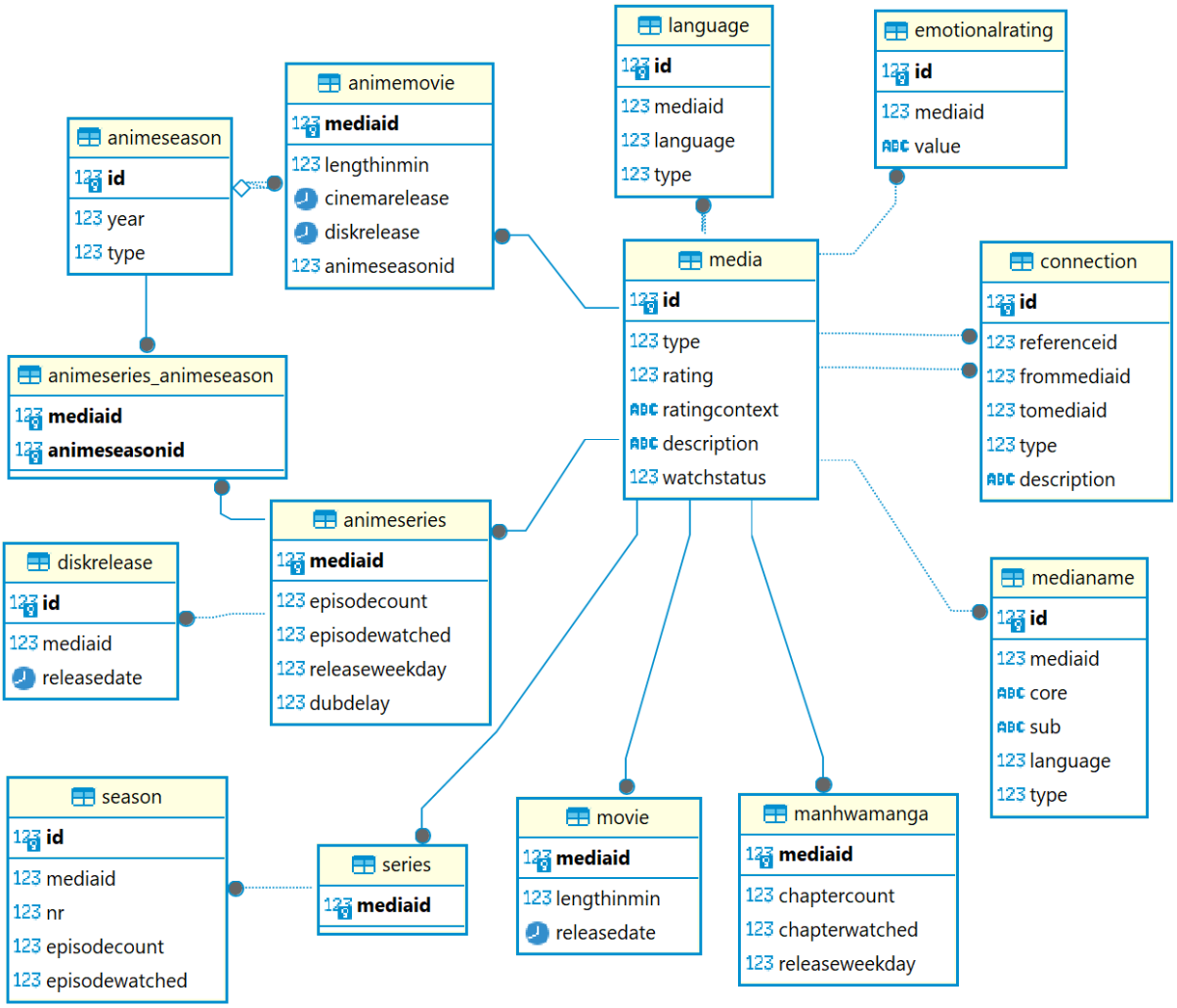
- Anstelle eines SeasonCount wird bei Anime auf Connections gesetzt, um Fortsetzung zu kennzeichnen. Zur Erklärung siehe Problemstellung.
- **ReleaseSeasons:** (siehe AnimeSeason)
- **ReleaseWeekday:** Die erste Folge kommt immer zu Beginn einer Anime-Season. Alle weitere im Wochentakt. Somit reicht Season und Wochentag, um den Release ein Animeserie zu verstehen.
- **DubDelay:** Machen Animeserien kommt mit Synchronisation heraus, allerdings kommen diese oft um 0-3 Wochen verschoben.
- **DiskReleases:** Animeserien werden immer über mehrere Volumes veröffentlicht, die an verschieden Tagen herauskommen.

AnimeSeason: Jede Anime Staffel kann immer einer oder mehreren Seasons zugeordnet werden. Anime-Seasons haben ähnlichen aber nicht den gleichen Rhythmus wie Wetter-Seasons.

Das Ganze soll so gestaltet sein, dass es sehr leicht durch mehr Typen erweitert werden kann, wie beispielsweise Musik, Youtube Video, Buch, usw.

ER Diagramm (SQL)

Wie das Model mit einer SQL Datenbank aussieht.



Ausnahmen (Normalformen)

Es wurden die dritte Normalform angewendet, allerdings gibt es ein paar Ausnahmen, die hier erläutert werden.

Die Series Relation ist eigentlich nicht notwendig, doch um einfache Erweiterbarkeit zu gewährleisten wurde sie trotzdem erstellt.

Die Type in der Relation Media ist eigentlich durch Relationen wie Movie, Series, usw. abgebildet, doch um eine vereinfachte Abfrage zu ermöglichen wurde er hinzugefügt.

Potenzelle Verbesserungen

Die nachfolgenden Bedingungen sind bereits in der Anwendung gewährleistet, könnten aber noch in der Datenbank, mit Hilfe von Constraints und Triggern, direkt umgesetzt werden.

Jede Serie muss mindestes eine Season besitzen und jede Media muss mindestens einen MediaName besitzen. Bei diesen Beziehungen handelt es sich um 1 zu N Beziehungen, welche ein N von mindestens 1 besitzen.

Man könnte dafür sorgen, dass eine Media entweder ein Film oder eine Serie aber nicht beides gleichzeitig sein kann.

Dafür sorgen das EpisodeWatched mit EpisodeCount gleichgesetzt wird, wenn WatchStatus sich zu „Finshed“ ändert. Diese Änderung ist nicht nur positiv, da der vorherige Zustand von EpisodeWatched verloren geht.

ER Diagramm (NoSQL)

Wie das Model mit einer NoSQL Datenbank aussieht.

Fehlt

ER Diagramm (Hybrid)

Wie das Model mit einer Hybriddatenbank aussieht.

Fehlt

Beschreibung der konkreten Umsetzung / eigene Implementierung

Fehlt

Known Problems

SQL vs NoSQL

Vorteile einer Relationalen Datenbank:

- Das benützten von SQL als Query Sprache. Da SQL weit verbreitet, muss oft keine neue Sprache gelernt werden.
- SQL ist sehr gut in zusammenrechnen von Datensätzen.
- Es gibt gute Dokumentationen und es relativ einfach zu lernen

Nachteile:

- Debugging kann aufgrund von schlechten Fehlermeldungen sehr schwer sein.

Vorteile einer NoSQL Datenbank:

- Flexibles Schema
- Geringe Infrastruktur kosten
- Hohen Datendurchlauf und Erreichbarkeit

Nachteile:

- Weniger ausgereifte Technologie und Hart zu verwalten
- Limitierte Abfrage Möglichkeiten
- Daten Inkonsistenz und schlechte Performance in manchen komplexen Szenarien

Was sind ausschlaggebende Gründe, um NoSQL in Erwägung zu ziehen?

- Bei massiven Mengen von Datensätzen (big data)
- Bei benötigen von Echtzeit Zugriff von Daten
- Bei häufig ändernden Datenmodellen
- Bei einer Priorität von horizontaler Skalierbarkeit

Server

Die Anwendung soll lokal auf dem Windows Rechner ausgeführt werden, weshalb es fraglich ist, dass man einen Datenbank-Server (PostgreSql) installieren muss, um die Anwendung ausführen zu können. Eine mögliche Lösung wäre SQLite zu verwenden.

Normalisierung

Unter dem Begriff Normalisierung verstehet man das Aufteilen von Attributen auf mehrere Relationen (Tabellen) um Redundanzen zu vermeiden.

Unter Redundanzen versteht man das mehrfache Speichern einer Information.
Zum Beispiel, nur weil eine Media mehrere Namen haben kann, muss das Rating nicht auch mehrfach abgespeichert werden.

Erste Normalform (1NF)

Alle Informationen in einer Tabelle sollen atomar vorliegen.

Eine Information ist dann atomar, wenn eine Information nicht mehr geteilt werden kann.
Zum Beispiel ein Spalte Name die eine Information „Jeremia Ott“ enthält ist nicht atomar, da diese Information in einen Vor- und Nachnamen geteilt werden kann.

Zweite Normalform (2NF)

Für die zweite Normalform muss sich eine Relation in der ersten Normalform befinden und Nichtschlüsselattribute nicht nur von einem Teil des Schlüsselattribut abhängt, sondern von einem ganzen Schlüssel.

Beispiel:

Media					
MediaId	MediaType	Rating	Name	Language	NameType
1	Anime	A	Attack on Titan	Englisch	Original
1	Anime	A	Shingeki no Kyojin	Japanisch	Original

Nach Anwendung der zweiten Normalform:

Media		
Id	Type	Rating
1	Anime	A

Name			
Id	Name	Language	Type
1	Attack on Titan	Englisch	Original
1	Shingeki no Kyojin	Japanisch	Original

Unterstrichen bedeute es ist Teil des Schlüssels der Relation. Eine Relation kann auch einen zweiteiligen Schlüssel haben, solange alle nicht Schlüsselattribute von beiden Teilen abhängen.

Dritte Normalform (3NF)

Auch hier gilt eine Relation ist nur dann in der dritten Normalform, wenn auch die zweite erreicht wurde. Außerdem muss ein nicht Schlüsselattribut immer direkt von dem Kandidatenschlüssel abhängen.

Kandidatenschlüssel ist die kleinste Kombination von Attributen, die eine Spalte eindeutig identifiziert.
Beispiel:

Kunde						
Id	Nachname	Vorname	Straße	HausNr	PLZ	Ort
1	Ott	Jeremia	Baldringer Weg	31	8490	Bad Radkersburg

Nach Anwendung der dritten Normalform:

Kunde					
Id	Nachname	Vorname	Straße	HausNr	PLZ
1	Ott	Jeremia	Baldringer Weg	31	8490

Postleitzahl	
PLZ	Ort
8490	Bad Radkersburg

Ausblick und Fragestellung der Bachelorarbeit

Wie groß ist der Unterschied zwischen Relationaler, Dokument und Hybriddatenbank?

Um herauszufinden, wie gut jede Lösung abschlägt, sollen jede Lösung nachfolgenden Metriken verglichen. Zugriffsgeschwindigkeit, Menge an mehrfach gespeicherten Daten (Redundanz), Speicherplatz (Datenkompression), Komplexität der Implementierung und Erweiterbarkeit.

Zugriffsgeschwindigkeit: Hier für kann ein Datenbanktest-Framework verwendet werden.

Daten Redundanz: Bei der Redundanz sollte vermutlich in Prozent gesehen werden. Wie viel Prozent der Daten ist redundant? Außerdem macht es vielleicht Sinn die Redundanten Daten in einem Graphen darzustellen, um die Verteilung sehen zu können.

Speicherplatz: Damit wird sich angeschaut wie viel Speicherplatz die Datenbank benötigt. Somit wird also indirekt auch die Datenkompression angeschaut.

Komplexität der Implementierung: Dies wird vermutlich eher eine Subjektive Beurteilung der Query-Sprache.

Erweiterbarkeit: Hier bei wir beurteilt, wie schwierig es ist bestimmte Erweiterungen umzusetzen. Zum Beispiel mehr Medien Typen wie Bücher und Musik.

Zugriffsgeschwindigkeit, Daten Redundanz, Speicherplatz sollen mit dem gegebenen Fallbeispiel aus Sicht des Best, Worst und Average Case betrachtet werden.

Best Case: 500
Average Case: 3000
Worst Case: 30000

Die Verteilung auf die einzelnen Medien sieht dabei wie folgt aus:

Serien	Filme	Animeserien	Anime Filme	Manhwa/Manga
1/9	1/9	1/3	1/9	1/3

Diese Verteilung beschreibt, die vermutet Verteilung der Medien.

Wenn nun ein Crawler geschrieben wird der Name von Medien sammelt und speichert, kann man mit der Verteilung und den Namen einen Generator schreiben der semi-realistische Daten für die Datenbank erzeugt, um alle Cases nachzustellen.

Es ist sehr wahrscheinlich das, wenn die Namen nicht möglichst divers sind es das Ergebnis der Speicherplatz verfälscht, da gleiche Namen vermutlich besser komprimiert werden können.

Quellen

- https://who.is/whois/myanimelist.net
- https://who.is/whois/imdb.com
- https://www.youtube.com/watch?v=hj2yFugmpz8&t=470s
- https://www.datenbanken-verstehen.de/datenmodellierung/normalisierung/
- https://www.datenbanken-verstehen.de/datenmodellierung/normalisierung/erste-normalform/
- https://www.datenbanken-verstehen.de/datenmodellierung/normalisierung/zweite-normalform/
- https://www.datenbanken-verstehen.de/datenmodellierung/normalisierung/dritte-normalform/
- https://www.datenbanken-verstehen.de/datenmodellierung/normalisierung/boyce-codd-normalform/
- https://www.geeksforgeeks.org/open-source-nosql-databases/
- https://www.coursera.org/articles/nosql-vs-sql