

Bericht Datenbankprojekt

Datenbanksysteme, FS 23

Filmeempfehlungssystem in MySQL von Solo2

Previtali Nico, nico.previtali@stud.hslu.ch

18/06/2023

Einführung & Ausgangslage

Während meinem Data Science Studium bin ich schon einige Male mit Empfehlungssystemen konfrontiert worden, ohne mich mit den zugrunde liegenden Datenverarbeitungsprozessen auseinanderzusetzen. Dieses Projekt ist somit ideal, um ein tieferes praktisches Verständnis zu erlangen, sowohl für den Aufbau von Informationssystemen, für die Empfehlungsmethodik als auch für die Datenbanktechnik.

Wenn ich einen neuen interessanten Film suche, benötige ich viel Zeit, um eine gute Empfehlung im Internet zu finden und basierend auf nur einer allgemeinen Filmempfehlung, welche mehrere ähnliche Filme enthält, bleibt die Suche meistens erfolglos. Überlasse ich die Suche einem bestehenden Empfehlungssystem, werde ich auf ein spezifischen Eco-System eingeschränkt (Netlix, Apple, Google etc.) und ich überlasse diesen Anbietern meine Benutzerdaten. Ich würde lieber ein Filmeempfehlungssystem verwenden, das wenige persönliche Daten von mir benötigt und trotzdem bessere personalifizierte Vorschläge generiert.

Die einfachste Lösung wäre einen Ausgangsfilm zu wählen, und basierend auf mehreren unterschiedlichen Bewertungsdaten einen Vorschlag zu erhalten. Dazu können zunächst die Bewertungen aus zwei grossen Filmquellen verwendet werden: MovieLens und die IMDb, welche mehrere Millionen Filmbewertungen gesammelt haben.

Zukünftig können die Datenquellen nach dem Benutzerinteresse erweitert werden (ArtHouse, Film Noire, 80ies, etc.). Auf diese Weise wird die 'Sicht' des Benutzers weniger stark eingeschränkt (Bubble) und somit stärker personalifiziert ohne dabei sein Benutzerverhalten einer Softwarefirma zu verkaufen.

1. Projektidee & Use Case

- Entscheidungsunterstützung für Wertgenerierung
- Eingesetzte Analysemethode für die Entscheidungshilfe
- Ermittelte Datenquellen für die Analyse
- Datenbanktechnologie mit Begründung

Ziel dieses Projekts ist es, Entscheidungen über die Filmauswahl durch die Bereitstellung relevanter Informationen und Empfehlungen zu unterstützen, die auf einer Datenanalyse basieren. Dies entspricht dem Data Value Cycle, bei dem aus Daten Werte geschaffen werden, indem sie analysiert und in nützliche Informationen umgewandelt werden. In unserem Projekt werden Filminformation und Benutzervorlieben verwendet um personalifizierte, unabhängige und interessante Filmempfehlungen für den Benutzer zu generieren.

1. Mittels einem Anwendungsfalldiagramm ermitteln wir die von aussen sichtbaren Interaktionen des Benutzers mit dem zu entwickelnden System. Dabei erstellen wir eine erste 'Systemvision' und ermitteln die zentralen Systemanforderungen.
2. Wir ermitteln die Kennzahlen, welche wir für unseren zentralen Anwendungsfall 'Zeige Filmempfehlungen' benötigen. Das sind beispielsweise die Anzahl der Bewertungen, die durchschnittlichen Bewertungen, die Häufigkeit der Tags und die Relevanz der Tags für jeden Film. Diese Kennzahlen helfen bei der Entscheidungsfindung, indem sie Filme identifizieren, die den Präferenzen des Benutzers entsprechen, und so die Wahrscheinlichkeit erhöhen, dass die empfohlenen Filme dem Geschmack des Benutzers entsprechen.
3. Ausgehend von den Kennzahlen finden und bestimmen wir die relevanten Datenquellen. Dazu werden die Datenquellen analysiert, neu organisiert und auf unsere Kennzahlen hin untersucht.

4. Nachdem die logischen Systemanforderungen und die Datenquellen bestimmt sind soll die technische Umsetzung bestimmt werden. Verwenden wir SQL, No-SQL oder AI?

2.1 Anwendungsfalldiagramm



Die Datenanalyse in unserem Projekt unterstützt die Entscheidungsfindung, indem sie Benutzern dabei hilft, Filme zu identifizieren, die ihren Interessen und Präferenzen entsprechen. Analyseergebnisse wie durchschnittliche Bewertungen, Anzahl der Bewertungen, Tag-Häufigkeiten und Tag-Relevanz werden bereitgestellt, um den Benutzern bei der Entscheidungsfindung zu helfen. Diese Ergebnisse werden in einer benutzerfreundlichen Oberfläche präsentiert, die es den Benutzern ermöglicht, ihre Auswahl basierend auf verschiedenen Kriterien zu verfeinern.

Empfehlungssysteme leiden unter dem Kaltstartproblem: Da über neue Benutzer nicht genug Daten implizit verfügbar sind, können keine wirklich wertegenerierende Vorschläge empfohlen werden. Um dieser Schwachstelle entgegenzuwirken, soll der Benutzer explizit einen Startfilm auswählen und über präzises Präferieren von Filmen beschreibenden Tagwörtern (Stichwörter, e.g., realistisch, gedankenanregend, humorvoll) seine Vorlieben ohne grossen Aufwand übermitteln können.

2.2 Zentrale Kennzahlen für die Filmempfehlung

Für ein aussagekräftiges Empfehlungssystem sind konsistente und umfangreiche Daten zentral. Das Empfehlungssystem soll bei der Bewältigung dieser Informationsflut unterstützen, indem es dem Benutzer aus einer unübersichtlichen Menge an Filmen eine Teilmenge empfiehlt. Dazu bestimmt unser System zu dem aktuell ausgewählten Ausgangsfilm (UC01 Bestimme Ausgangsfilm) weitere, ähnliche Filme und verwenden die beiden klassischen Typen von Empfehlungsdiensten: 1. Kollaborative Filterung (UC02: Wähle Empfehlung über ähnliche Benutzer) und 2. Inhaltsbasierte Empfehlung (UC03: Wähle Empfehlung über Filmtag).

Bei unserem speicherbasierten Empfehlungssystem (Datenbank) nutzen wir alle gespeicherten Bewertungsdaten, um mit Hilfe von Ähnlichkeiten die möglichen Benutzer-Objekt-Kombinationen zu ermitteln. Wir betrachten den Benutzer als zentrale Entität für den die personalisierte Empfehlung bestimmt werden soll und mit Hilfe der Bewertungsdaten ermitteln wir die Ähnlichkeiten im Benutzerverhalten. Unsere Empfehlungssystem benötigt somit vier zentrale Informationstypen: A) Benutzer Informationen, B) Film Informationen C) Filmbewertungen und D) Filmtags. Diese Datentypen mit Ihren Kennzahlen sind denn auch die Hauptdarsteller bei unseren zentralen Anwendungsfällen für den Informations-Input oder Output.

1.1.1. Die zentralen Anwendungsfälle

UC01 Bestimme Ausgangsfilm	
Beschreibung	Der Benutzer sucht mittels Filtermöglichkeiten einen Ausgangsfilm aus, welcher Ihm eine Auswahl von möglichen Filmen anzeigt.
Input	Der Film kann mittels dem Filmenamen oder Teile eines Filmenamens gesucht werden.
Output	Der Benutzer erhält eine Liste der gefilterten Filme und kann danach den Ausgangsfilm bestimmen. Neben dem Filmenamen wird die eindeutige ID des Ausgangsfilmes bestimmt. Die selektierte ID des Ausgangsfilm verwenden wir als Input für das Kollaborative oder Inhaltsbasierte Filtern (Use Cases UC02 und UC03).

UC02 Wähle Empfehlung über ähnliche Benutzer – Kollaborative Filterung	
Beschreibung	Kollaborative Filterung: Die Hauptidee besteht darin, Filme zu finden, die von Benutzern geschätzt werden, welche den Ausgangsfilm geschätzt haben: 1. Bestimmen aller Benutzer, welche den Ausgangsfilm (> 3.5) hoch bewertet haben 2. Bestimmen aller Filme, welche von diesen Benutzern ebenfalls hoch (> 3.5) bewertet wurden
Kennzahlen	Wir benötigen für den UC02 die folgenden Kennzahlen: 1. Benutzerbewertungen: Gewichtete Durchschnitt pro Film und Datenquelle (siehe 2.1.2). 2. Benutzerbewertungen: Anzahl Bewertungen pro Film und Datenquelle 3. Benutzerbewertungen: Der Rang der Bewertung pro Film innerhalb jeder Datenquelle. 4. Der kumulierte Rang des Filmes über alle Datenquellen hinweg.

Output	<p>Die Kennzahlen der Benutzerbewertungen werden pro Datenquelle (z.B. IMDB) ausgewertet, danach wird pro Datenquelle eine Rangliste erstellt. Mit der Hilfe dieser Rangbewertung können Datenquellen mit unterschiedlichen Bewertungssystemen besser verglichen/kumuliert werden. Mittels dem Rang und der Anzahl der Bewertungen werden die Gewinner ermittelt und dem Benutzer angezeigt. Die Filme aus der 'Blacklist' des Benutzers werden nicht angezeigt.</p> <p>Anzeige: Filmtitel, Erscheinungsjahr und Genre</p>
--------	---

UC03 Wähle Empfehlungen über Filmtag	
Beschreibung	<p>Inhaltliche Filterung: Die Hauptidee besteht darin, Filme zu finden welche eine hohe Ähnlichkeit zum Ausgangsfile aufzeigen. Dazu verwenden wir mehrere Stichwörter (Filmtags) welche den Film genauer beschreiben: z.B. realistisch, gedankenanregend, humorvoll.</p> <ol style="list-style-type: none"> 1. Bestimmen aller Filmtags im Ausgangsfilm. 2. Der Benutzer kann die Gewichtung aller vorgeschlagenen Filmtags via GUI justieren, z.B. weniger realistisch aber mehr humorvoll. 3. Bestimmen aller Filme, welche Filmtags mit entsprechender Gewichtung enthalten.
Kennzahlen	<p>Wir benötigen für den UC03 die folgenden Kennzahlen:</p> <ol style="list-style-type: none"> 1. Film ID: Dies ist die eindeutige Identifikationsnummer eines Films und erlaubt uns die Informationen für einen bestimmten Film zu beziehen. 2. Filmtag ID und Filmtag Namen: Diese Kennzahlen sind wichtig, um die Tags, die zu den Filmen zugeordnet sind, zu identifizieren und zu erkennen. Sie bieten dem Benutzer eine Beschreibung der Merkmale des Films (z.B. realistisch, humorvoll, Horror). 3. Relevanzwert: Dies ist der Relevanzwert eines Tags für einen bestimmten Film, der aus den Benutzerbewertungen berechnet wird. Er dient als Gewichtungsfaktor, um die Bedeutung jedes Tags für jeden Film zu beurteilen. 4. Relevanz Differenz: Dies ist eine Metrik, die den Unterschied in der Relevanz zwischen den Tags eines Eingabefilms und den Tags aller anderen Filme misst. Sie wird verwendet, um ähnliche Filme zu finden: Je kleiner die gesamte Differenz, desto ähnlicher sind die Filme in Bezug auf ihre Tag-Verteilung.
Output	<p>Die Entscheidungsfindung des Benutzers wird durch diese Kennzahlen unterstützt, indem Filme mit ähnlicher Tag-Verteilung und damit ähnlichen Eigenschaften vorgeschlagen werden, basierend auf dem Ausgangsfilm. Die Tag-Verteilung und ihre Relevanz bieten eine detaillierte und genaue Beschreibung der Merkmale des Films, die dazu beitragen können, die Präferenzen des Benutzers zu erfüllen. Dem Benutzer werden die ersten 10 Filmvorschläge angezeigt. Die Filme aus der 'Blacklist' des Benutzers werden nicht angezeigt.</p> <p>Anzeige: Filmtitel, Erscheinungsjahr und Genre</p>

UC04 Bestimme Filmkriterien – Einfache Filterung	
Beschreibung	Ergänzende einfache Filterung im UC02 und UC03: Der Benutzer filtert die Filme mittels verschiedenen Filterkriterien.
Kennzahlen	Einfache Filterkriterien: Erscheinungsjahr, vorgegebene Tags und Genre.
Output	Anzeige der gefilterten Filme.

1.1.2. Die Bewertungsarten

Bei unserem Anwendungsfall verwenden wir nicht nur die durchschnittliche Bewertung, sondern verwenden die folgenden Kenngrößen:

A) Durchschnittliche Bewertung (Extraktion): Die durchschnittliche Bewertung eines Films wird ermittelt, indem alle Bewertungen für diesen Film addiert und durch die Anzahl der Bewertungen dividiert werden. Dies liefert eine einfache, aber dennoch aussagekräftige Kennzahl, um Filme basierend auf den Meinungen der Benutzer zu vergleichen.

Mathematische Formel: $\bar{B} = \frac{\sum_{i=1}^n b_i}{n}$

Dabei ist:

- \bar{B} : Durchschnittliche Bewertung
- b_i : Bewertung i
- n : Anzahl der Bewertungen

B) Gewichteter Durchschnitt: Der gewichtete Durchschnitt berücksichtigt neben der durchschnittlichen Bewertung auch die Anzahl der Bewertungen. Filme mit vielen Bewertungen werden stärker gewichtet als Filme mit wenigen Bewertungen. Dies verhindert, dass Filme mit nur wenigen, aber sehr guten Bewertungen zu stark bevorzugt werden. Die Berechnung des gewichteten Durchschnitts erfolgt durch die Multiplikation der durchschnittlichen Bewertung mit der Anzahl der Bewertungen und anschließend durch die Division durch die Summe aller Bewertungen.

Mathematische Formel: $W_D = \frac{\sum_{i=1}^n (b_i \cdot w_i)}{\sum_{i=1}^n w_i}$

Dabei ist:

- W_D : Gewichteter Durchschnitt
- b_i : Bewertung i
- w_i : Gewicht (Anzahl der Bewertungen) für Bewertung i
- n : Anzahl der Bewertungen

C) Quellenrang durch Quadratwurzel der Bewertungsanzahl: Der Quellenrang wird als $R_i = w_i / \sqrt{n_i}$ berechnet. Hierbei ist R_i der Rang der i-ten Quelle. Dies ist eine Form der Regularisierung, die oft in Empfehlungssystemen verwendet wird, um den Einfluss von Filmen mit einer geringen Anzahl von Bewertungen zu reduzieren.

Die Idee ist, dass wenn ein Film nur wenige Bewertungen hat, selbst wenn diese alle hoch sind, wir nicht sicher in der Qualität des Films sein können, wie wir es wären, wenn der Film viele hohe Bewertungen hätte. Daher werden durch die Teilung durch die Quadratwurzel der Anzahl der Bewertungen Filme mit

mehr Bewertungen im Vergleich zu Filmen mit weniger Bewertungen einen höheren Quellenrang erhalten, auch wenn ihre durchschnittliche Bewertung gleich ist.

Fazit

Die Verwendung sowohl **des gewichteten Durchschnitts als auch des Quellenrangs** bei den Berechnungen berücksichtigt sowohl die durchschnittliche Gesamtbewertung als auch das Vertrauen in diese Bewertung auf Basis der Anzahl der Bewertungen. Dies kann eine differenziertere Sicht auf die Daten bieten und kann dazu beitragen, qualitativ hochwertige Filme zu identifizieren, die von einer großen Anzahl von Benutzern bewertet wurden.

Diese Methodik sorgt dafür, dass unser Empfehlungssystem sowohl die Qualität (durch den gewichteten Durchschnitt) als auch die Beliebtheit (durch die Regularisierung) der Filme berücksichtigt. Dies führt zu robusteren und zuverlässigeren Empfehlungen.

2.3 Ermittelte Datenquellen

Die Daten für dieses MySQL-Projekt stammen aus zwei umfangreichen und renommierten Filmedatenbanken, nämlich MovieLens und IMDb. Beide Datenbanken haben über 20 Jahre hinweg Filme und Filmbewertungen gesammelt und sind als verlässliche Informationsquellen in der Filmbranche etabliert. Zudem sind die Datensätze dieser Datenbanken opensource und daher leicht zugänglich, was sie zu einer idealen Wahl für unser Projekt macht.

Der Hauptgrund für die Verwendung dieser Datenquellen liegt in der Notwendigkeit, präzise Vorhersagen treffen zu können und aussagekräftige Durchschnittswerte zu erhalten. Um ein fundiertes und zuverlässiges Datenmodell zu erstellen, ist es unerlässlich, dass die zugrunde liegenden Daten aus vertrauenswürdigen und umfangreichen Quellen stammen. Durch die Kombination der Informationen aus MovieLens und IMDb können wir ein umfassendes Bild der Filmbranche gewinnen und so eine qualitativ hochwertige Analyse und Prognose sicherstellen.

2.3.1 MovieLens

MovieLens hat bereits eine umfangreiche Arbeit zur Datenextraktion von 283228 auf Ihrer Plattform registrierten Benutzer geleistet, um einen strukturierten Datensatz für Forschungszwecke bereitzustellen. Die Benutzerdaten entspringen der Zeitspanne vom 09. Januar 1995 bis 26. September 2018 und wurden am letzten Tag als das Datenset ml-latest generiert. Folgende Informationen sind verfügbar:

1. **ratings.csv**: Gesammelte MovieLens-Benutzerbewertungen zu Filmen auf einer Skala von 0,5 bis 5 Sternen. Die Datei enthält anonymisierte Benutzer-IDs, Film-IDs, Bewertungen und Zeitstempel
2. **tags.csv**: Gesammelte benutzergenerierte Filmtags. Die Datei enthält anonymisierte Benutzer-IDs, Film-IDs, Tags und Zeitstempel.
3. **movies.csv**: Grundlegende Informationen zu den Filmen, wie Titel und Genres
4. **links.csv**: Externe Links zu IMDb- und TMDb-Seiten für jeden Film
5. **genome-scores.csv** und **genome-tags.csv**: MovieLens hat mit Hilfe von Machine Learning Algorithmen benutzergenerierte Inhalte wie Tags, Bewertungen, Textrezensionen und Kritiken verarbeitet, um den sogenannten *Tag Genome* zu erstellen. Der *Tag Genome* lässt sich als Dichtematrix beschreiben: Jeder Film hat also für jeden Tag einen Relevanzwert.
<< Jesse Vig, Shilad Sen, and John Riedl. 2012. The Tag Genome: Encoding Community Knowledge to Support Novel Interaction. ACM Trans. Interact. Intell. Syst. 2, 3: 13:1–13:44.
<https://doi.org/10.1145/2362394.2362395> >>

2.3.2 IMDb

Die Internet Movie Database (IMDb, englisch für Internet-Filmdatenbank) ist eine US-amerikanische Datenbank zu Filmen, Fernsehserien, Videoproduktionen und Computerspielen. Im Juni 2022 gab es zu über elf Millionen Titeln Einträge. Diese verteilen sich unter anderem auf Einträge zu 613.000 verschiedenen Spielfilmproduktionen, über 260.000 Videofilmen, über 136.000 TV-Filmen, 227.000 TV-Serien, fast 6,8 Millionen Serienepisoden und mehr als 1,9 Millionen Einträge zu Podcast-Episoden. Zudem sind über 11,7 Millionen Film- und Fernsehschaffende aufgeführt. Betrieben wird die Datenbank seit 1998 von Amazon. Folgende Informationen sind verfügbar:

1. **title.basics.tsv.gz**: Für jeden Filmtitel wurde eine ID (tconst) definiert, der Titeltyp, der Haupttitel, der Originaltitel, das Veröffentlichungsjahr, das Endjahr für Serien, die Filmdauer in Minuten, die Genres und Informationen zur Kinderfreundlichkeit gesammelt.
2. **title.ratings.tsv.gz**: Für die Filmbewertung wurde ein gewichteter Durchschnitt aller Benutzerbewertungen auf einer Skala von 1-10 mit der dazugehörigen Anzahl Bewertungen gesammelt.

2.4 Technische Umsetzung

2.4.1 Kriterien

Sprache: SQL gibt es schon seit über 40 Jahren und ist daher vielen bekannt, gut dokumentiert und weit verbreitet. Sicher und vielseitig, eignet es sich besonders gut für komplexe Abfragen. Allerdings schränkt SQL den Benutzer auf die Arbeit innerhalb eines vordefinierten Tabellenschemas ein. Deshalb ist es aufwendiger, die Daten zu organisieren und zu verstehen, bevor sie sich verwenden lassen. Die dynamischen Schemata von NoSQL Datenbanken erlaubt eine höhere Flexibilität beim Hinzufügen neuer Attribute und Felder, jedoch fehlt den NoSQL Sprachen die Standardschnittstelle, sodass komplexere Abfragen schwierig auszuführen sein können.

Fazit: Für unsere Filmdatenbank sind beide Ansätze denkbar. Dabei sind die vordefinierten SQL-Tabellenschemas kein Nachteil bei der Umsetzung.

Skalierbarkeit/Performance: SQL-Datenbanken lassen sich vertikal skalieren, indem die Rechenleistung der vorhandenen Hardware erhöht wird. NoSQL ermöglicht eine bessere horizontale Skalierbarkeit mit zusätzlichen Servern und Netzwerkknoten.

Fazit: Für unseren Anwendungsfall ist die horizontale Skalierbarkeit relevant (SQL), weil alle Informationen der Datenquellen auf einer DB in strukturierter Form gesammelt werden und somit keine unstrukturierten Daten hinzugefügt werden

Konsistenz: Die SQL-Datenbank befolgt bei jedem Schritt unveränderliche Größen: Regeln, die validieren und Verfälschungen verhindern. Dies wird mittels Konsistenzprüfungen auf einer SQL-Datenbank unterstützt (Datenbankschema, Stored Procedures). **Datenintegrität:** Gewährleistet, dass Daten bei Aktualisierungen über Tabellen einheitlich bleiben.

Fazit: Es ist wichtig, dass die Daten immer gültig, akkurat und für die Abfragen verarbeitungsbereit sind, sodass möglichst präzise Empfehlungen generiert werden können. Für gute Empfehlungen ist es wichtig, dass die Daten zeitgerecht sind und aktualisiert werden.

So würden widersprüchliche Relevanzwerte für die Tags desselben Filmes zu den falschen Empfehlungen führen.

2.4.2 Entscheid technische Umsetzung

Die vielleicht bekannteste SQL-DB ist MySQL, ein quelloffenes und kostenloses RDBM, das allerdings auch in proprietären Lizenzen erhältlich ist. Nutzer setzen es vermehrt für Webanwendungen ein. Es ist bekannt für Kompatibilität, Support und gute Leistung. MySQL hat mit Funktionen wie einem JSON-Datentyp, dem „Document Store“ und der Unterstützung für Sharding (horizontale Skalierung) auch Zugeständnisse an NoSQL-Anwender gemacht.

Fazit: Für das Filmeempfehlungssystem verwenden wir ein MySQL Datenbank-System weil die SQL-Datenbank besser geeignet ist für a) kleine Daten, b) tabellarisch modellierte Daten und c) Systeme, bei denen Konsistenz entscheidend ist.

2. Datenmodell & Datenbankschema

- Datenmodell nach der ER-Technik v. Chen (1976)
- Datenbankschema in der Datenbank-Syntax (z.B. SQL)
- Zusammenhang Modell und Schema
- Verbale Beschreibung der wichtigsten Datenklassen und Attributen

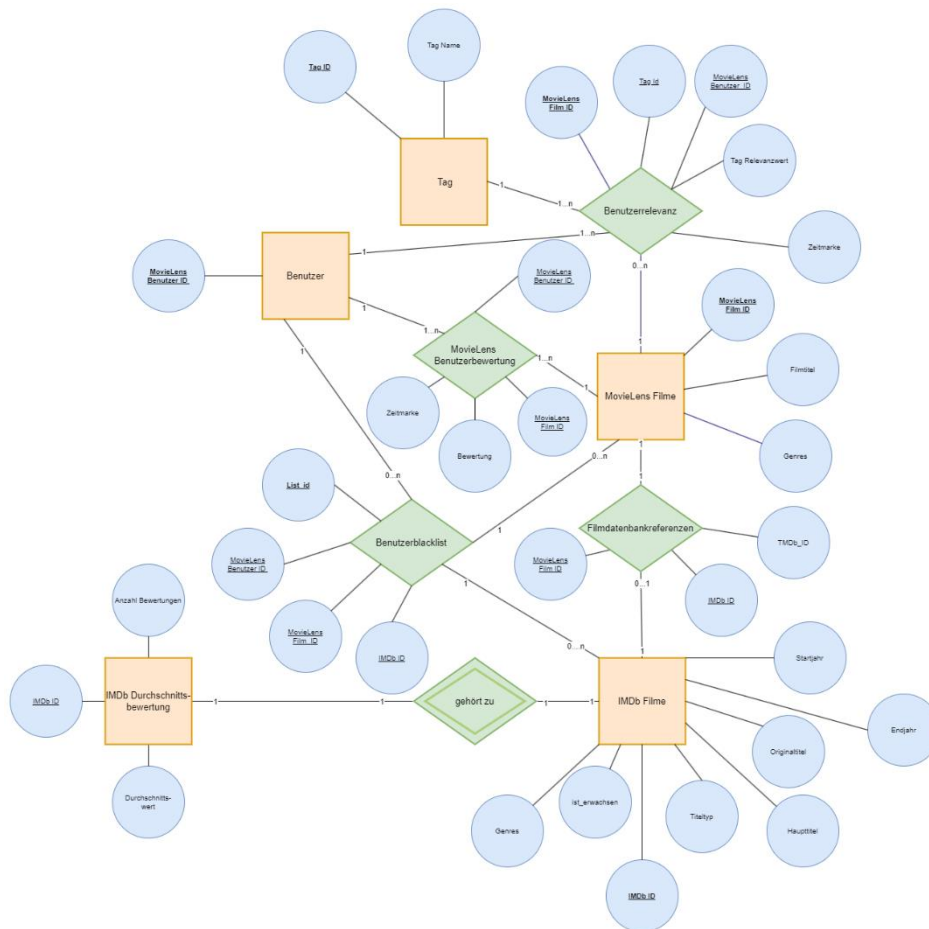


Abbildung 1 CHENN - ERD

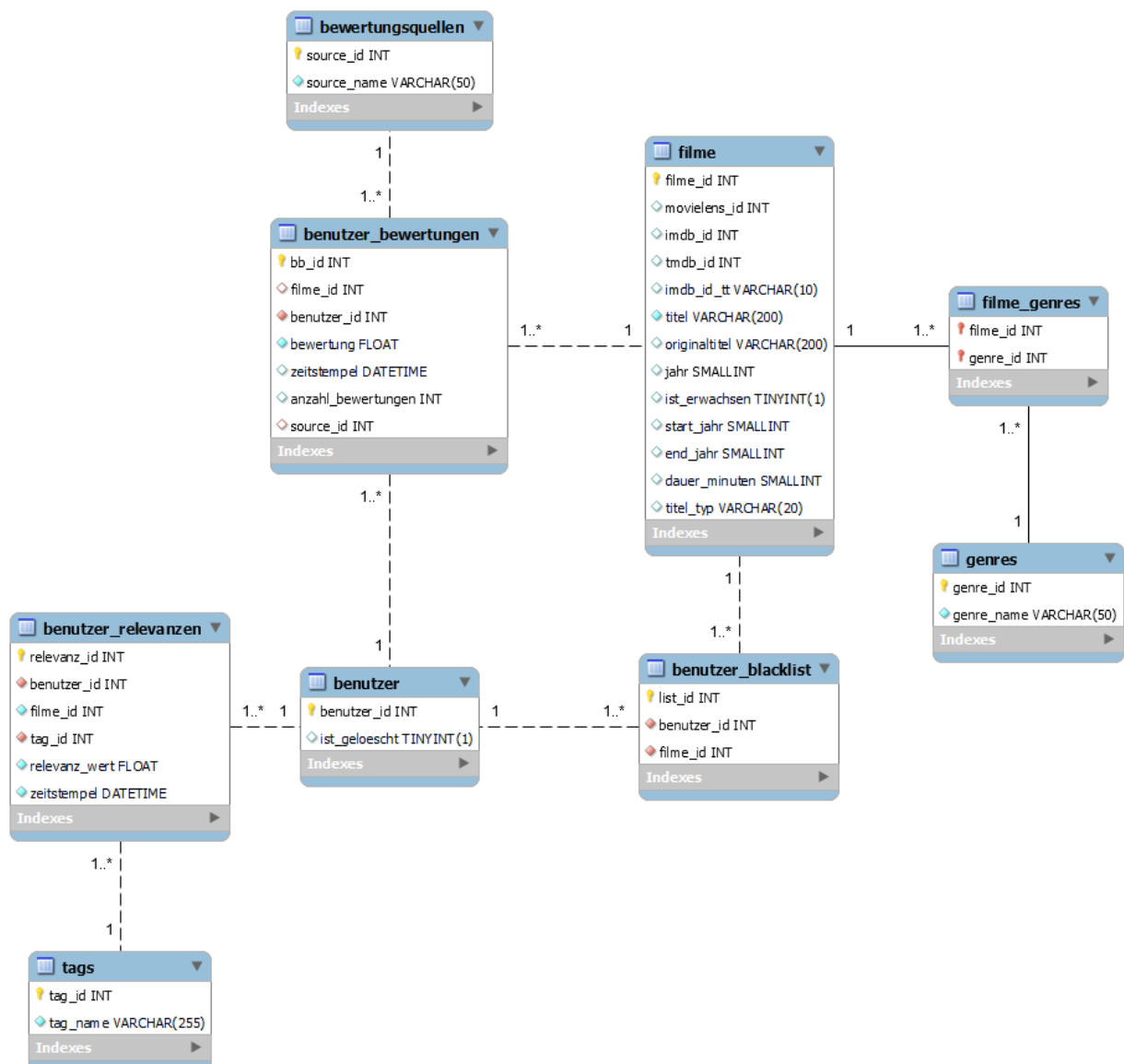


Abbildung 2 - Workbench EERD

3.1 Zusammenhang Modell und Schema

Beim Datenmodell geht es nicht darum, wie die echte Datenbank angewendet werden wird. Es funktioniert auf einem hohen Niveau und übersetzt Probleme der realen Welt und die Use Case Anforderungen in eine konzeptionelle Datenstruktur, die einfach zu verstehen und deshalb auch für Personen, die keine Datenbankexperten sind, verständlich ist. Es konzentriert sich darauf, diejenigen Einheiten zu identifizieren, die Teil der Datenbank sein werden und der Beziehungen/den Verbindungen zwischen ihnen, unabhängig von spezifischer Software, Systemen zum Datenbankmanagement oder Strukturen zur Datenspeicherung.

Datenmodell Filmdatenbank: Ausgehend von den Datenquellen und unseren Use Case Kennzahlen ordnen wir die Informationen unserer beiden Datenquellen in Entitäten und Beziehungen. Dazu trennen

wir die Datenstrukturen der beiden Filmdatenbanken Movielens und IMDB und erhalten eine Abbildung der realen Datenstrukturen (der realen Welt):

- Wir identifizieren die Entitäten: Benutzer, Movielens Filme, IMDB Filme, Movielens Bewertungen, IMDB Bewertungen und die Movielens Benutzerrelevanz (der Tags).
- Wir ergänzen die Entitäten mit den relevanten Attributen.
- Wir bestimmen die Beziehungen zwischen den Entitäten und setzen die Kardinalitäten.

Das Datenbankschema geht einen Schritt weiter und ist der erste Schritt zur Datenbankanwendung. Ausgehend vom Datenmodell wird die Sichtweise vom hohen Niveau aus in ein formelles Datenbankschema übersetzt. Die umsetzungsrelevanten Datenobjekte für ein herstellerspezifisches Datenbankmanagementsystem stehen dabei im Zentrum. Bei unserer relationalen Filmdatenbank sind dies die Tabellen und Spalten (inkl. Namen, Datentypen, Standardwerte, Primär-Fremdschlüsselangaben), Datenbankindizes und Berechtigungen. Im weiteren Projektverlauf können Views, Stored Procedures und Triggers in unserer MySQL Datenbank definiert werden.

Unterschiede zwischen Datenmodell und Datenbankschema

- Im Datenmodell werden Kardinalitäten und Assoziationstypen verwendet, um die Beziehungen zwischen Entitäten zu beschreiben, während im Datenbankschema die Beziehungen durch Fremdschlüssel und Tabellenstruktur dargestellt werden.
- Das Datenmodell konzentriert sich auf die logischen Aspekte der Datenbankstruktur und deren Beziehungen, während das Schema die tatsächliche Implementierung dieser Struktur in der Datenbank zeigt.
- Die Entitäten und Beziehungen im Modell werden im Schema als Tabellen dargestellt. Zum Beispiel wird die Entität "Benutzer" im Schema als Tabelle "benutzer" dargestellt.
- Die Attribute der Entitäten und Beziehungen im Modell werden im Schema als Spalten der entsprechenden Tabellen definiert. Zum Beispiel hat die Entität "Movielens Filme" die Attribute "Filmtitel", "genres" und "movielens_id", die im Schema als Spalten der Tabelle "filme" dargestellt werden.
- Die Kardinalitäten der Beziehungen im Modell werden im Schema durch Fremdschlüsselbeziehungen dargestellt. Zum Beispiel hat die Beziehung "IMDb Durchschnittsbewertung" die Kardinalität 1...1:n, und diese Beziehung wird im Schema durch die Fremdschlüsselbeziehungen zwischen den Tabellen "benutzer_bewertungen" und "filme" dargestellt.

Datenbankschema Filmdatenbank: Wir untersuchen das Datenmodell und übertragen die Elemente der konzeptionellen Sicht in die Tabellen und Spalten unserer MySQL Filmdatenbank:

- Dabei lösen wir die strikte Trennung der IMDB und Movielens Entitäten in einer gemeinsamen/generischen Sicht auf.
- Die Entitäten Movielens Filme, IMDB Filme und Filmdatenbankreferenzen werden in einer Tabelle 'filme' gespeichert. Somit können alle 1:1 Kardinalitäten in einer gemeinsamen Tabelle 'filme' gehalten werden und wir stellen sicher, dass alle notwendigen Filmattribute in nur einer Tabelle konsistent gehalten werden.
- Unsere Filmentscheidungen basieren auf Durchschnittsbewertungen sowie der Anzahl der Bewertungen. Während wir bei der IMDB Filme diese Werte direkt beziehen, werden wir beim Datenload der Movielens Benutzerbewertungen ebenfalls die Durchschnittsbewertung/Anzahl Bewertungen pro Film berechnen. Dadurch werden alle Bewertungen konsistent in derselben Form dem Entscheidungssystem zur Verfügung gestellt.
- Damit die Herkunft der Bewertungsinformation nicht verloren geht, werden diese Durchschnittsbewertungen mittels 'source_Id' separat pro Filmdatenbank in der Tabelle 'benutzer_bewertungen' gespeichert.

- Die Benutzerrelevanz wird wie im Datenmodell beschrieben in die tabellen ‚benutzer‘, ‚tags‘ und ‚Benutzerrelevanz‘ übernommen. Die Relevanz wird via ‚fime_id‘ (Fremdschlüssel) der tabelle ‚filme‘ zugeteilt.
- Normalisierung: Sowohl das DDL-Schema als auch das ERD sind normalisiert und erfüllen die Anforderungen der 1. bis 3. Normalform. Die Attribute sind von den Primärschlüsseln abhängig, und es gibt keine wiederholenden Gruppen oder partiellen Abhängigkeiten.
- Primärschlüssel: Im Allgemeinen stimmen die Primärschlüssel in beiden Entwürfen (ERD und DDL) überein. Jede Entität im ERD hat einen Primärschlüssel, und jede Tabelle im DDL hat ebenfalls einen Primärschlüssel.
- Unterschiede in den Attributnamen der Primärschlüssel: Es gibt einige Unterschiede in den Attributnamen der Primärschlüssel zwischen den beiden Entwürfen, aber die Bedeutung bleibt erhalten.

Die wichtigsten Datenklassen und Attribute unserer MySQL Datenbank:

Unsere Datenklassen können in drei Datenkategorien gegliedert werden: A) Film Informationen, B) Benutzer Informationen und C) Bewertungen

A) Film Informationen

Datenklasse ‚filme‘: Alle bewerteten Filme aus den Datenquellen IMDb und MovieLens. Neben der ‚filme_id‘ ist jeweils mindestens eine zusätzliche Film ID gespeichert.	
filme_id	Eindeutige Identifikationsnummer für jeden Film (PK)
movielens_id	Eindeutige Identifikationsnummer für jeden Film in der MovieLens DB (optional)
imdb_id	Eindeutige Identifikationsnummer für jeden Film in der IMDb (optional)
tmdb_id	Eindeutige Identifikationsnummer für jeden Film in der tMDb (optional)
titel	Der Titel des Films
originaltitel	Der Titel des Films in seiner Originalsprache
jahr	Veröffentlichungsjahr des Films
ist_erwachsen	Jugenschutz 0 oder 1
start_jahr	Startjahr der ersten Staffel einer Serie
end_jahr	Endjahr der letzten Staffel einer Serie
dauer_minuten	Filmlänge in Minuten
titel_typ	Art des Filmes: tvEpisode, shortmovie, movie,

Tabelle ‚genres‘: Referenztabelle aller möglichen Filmgenres (z.B. ‚Crime‘, ‚Sci-Fi‘, ‚Horror‘)	
genre_id	Eindeutige Identifikationsnummer für jeden Genre (PK)
genre_name	Bezeichnung des Genres

Tabelle ‚filme_genres‘: Jeder Film kann mehreren Genres zugeteilt werden. z.B. ‚Alien‘ wird dem Genre ‚Sci-Fi‘ und ‚Horror‘ zugeteilt. Jeder Film kann nur einmal einem Genre zugeteilt werden.	
filme_id	Zuteilung zum Film (PK1)
genre_id	Zuteilung zum Genre (PK2)

B) Benutzer Informationen:

Tabelle ‚benutzer‘: Ziel des Projektes ist es keine privaten Informationen über den Benutzer zu speichern.	
benutzer_id	Eindeutige Identifikationsnummer für jeden Benutzer (PK)

Tabelle ,benutzer_blacklist': Filme welche für den Benutzer nicht mehr empfohlen werden, z.B. weil der Benutzer den Film bereits gesehen hat.	
benutzer_id	Zuteilung zum Benutzer (PK1)
filme_id	Zuteilung zum Film (PK2)

C) Bewertungen:

Tabelle ,benutzer_bewertungen': Pro Film werden die aktuellen Durchschnittsbewertung und die Anzahl der Bewertungen gespeichert.	
bewertungs_id	Eindeutige Identifikationsnummer für jede Bewertung (PK)
filme_id	Zuteilung zum Film (FK)
benutzer_id	Zuteilung zum Benutzer (FK)
source_id	Zuteilung zur Bewertungsquelle (FK)
durchschnittsbewertung	Bewertungsskala 1-10 in 0.1 Steps
anzahl_bewertungen	Anzahl der Bewertungen
zeitstempel	Datum der letzten Bewertung

Tabelle ,bewertungsquellen': Referenztabelle zu den verwendeten Filmdatenbanken. Aktuell werden , MovieLens' (ID=01) und ,IMDb' (ID=02) verwendet.	
source_id	Eindeutige Identifikationsnummer der Datenquelle (PK)
source_name	Name der Datenquelle

Tabelle ,benutzer_relevanzen': Information über die Relevanz (wichtig/unwichtig) des Filmes für den Benutzer.	
relevanz_id	Eindeutige Identifikationsnummer für jede Benutzerrelevanz (PK)
benutzer_id	Zuteilung zum Benutzer (FK)
filme_id	Zuteilung zum Film (PK2)
tag_id	Zuteilung zu einem Schlüsselwert (FK)
relevanzwert	Benutzer definierte Taggewichtung, zum Beispiel: 0.35,0.178,...
zeitstempel	Datum der letzten Bewertung
Tabelle ,tag': Die ,tags' erlauben es für jeder Benutzerrelevanz zusätzliche Schlüsselwerte (Keywords) zu erfassen: Künstler, Dialog, Mood, Umgebung, Cinematics, oder weitere Filminformationen.	
tag_id	Eindeutige Identifikationsnummer für jeden Schlüsselwert (PK)
tag_bezeichnung	Bezeichnung des Schlüsselwertes

3. Daten laden & transformieren

- Systemkomponenten, Zusammenhänge und Datenflüsse
- Datenimport
- Datentransformationen

3.1. Systemkomponenten

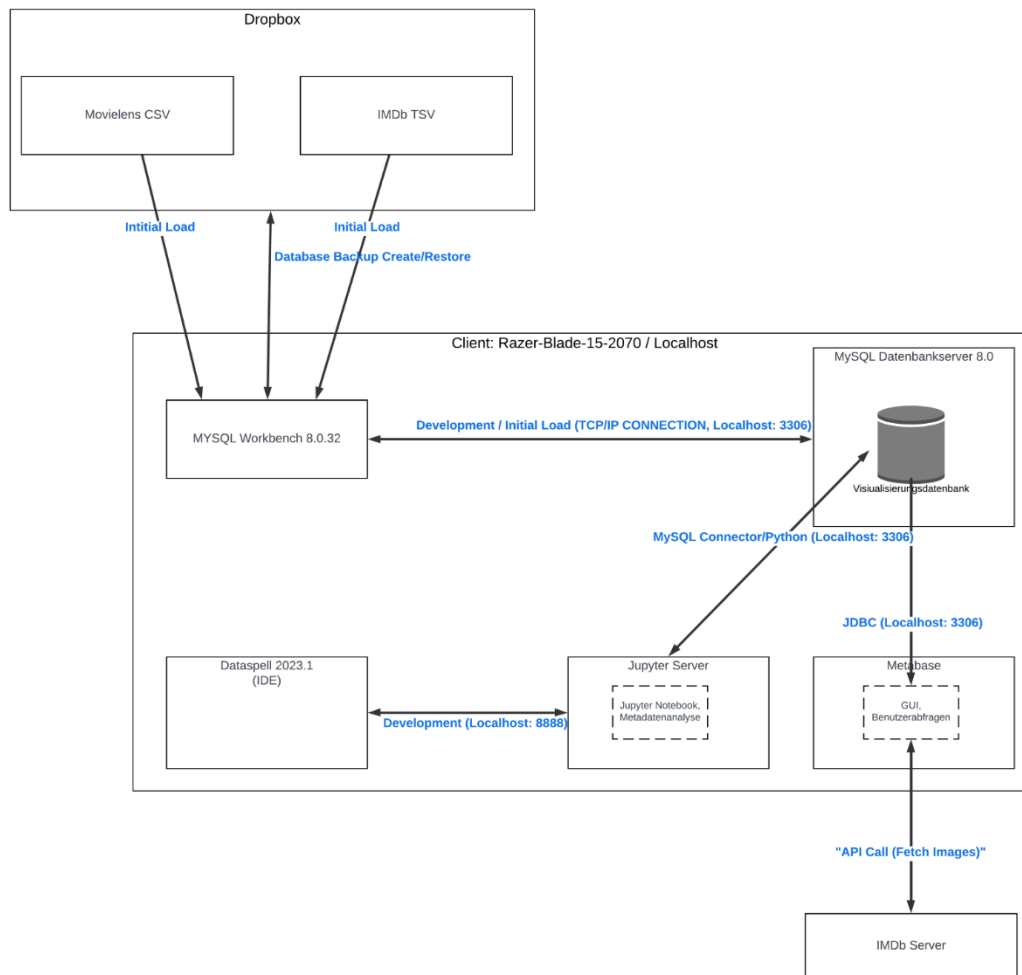


Abbildung 3 - Systemkomponenten

Das vorliegende DB-Projekt wird nicht als Gruppenarbeit, sondern wird als Einzelarbeit durchgeführt und somit entschied ich mich, alle eingesetzten Applikation lokal auf meinem Laptop zu installieren:

- Dropbox wird zur Datensicherung und zur Bereitstellung der IMDb und MovieLens Dateien verwendet.
- Die MySQL Workbench 8.0.32 sowie der MySQL Server sind lokal installiert und ermöglichen einerseits eine schnelle DB-Entwicklung und andererseits unterstützt die MySQL Workbench den Initial-Load der Filmdatenbank.
- Für die Metadatenanalyse während der Datenmigration haben wir Python verwendet.
- Für die Datenanalyse sowie die visuelle Darstellung der Daten verwenden wir Metabase.

3.1.1. Rohdaten

Die Rohdaten entsprechen den ermittelten Datenquellen in Kapitel 3.3.

Die MovieLens Dateien sind in CSV (Comma Separated Values) formatiert und in UTF-8 kodiert. Die erste Reihe ist die Titelzeile:

```
movies.csv X
C: > Users > nico > Dropbox > HSLU > AIML > Semester_6 > DBS > DBS 23 > Projekt > DATA > NEW > ml-latest > movies.csv
1  movieId,title,genres
2  1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy
3  2,Jumanji (1995),Adventure|Children|Fantasy
4  3,Grumpier Old Men (1995),Comedy|Romance
5  4,Waiting to Exhale (1995),Comedy|Drama|Romance
6  5,Father of the Bride Part II (1995),Comedy
7  6,Heat (1995),Action|Crime|Thriller
8  7,Sabrina (1995),Comedy|Romance
9  8,Tom and Huck (1995),Adventure|Children
10 9,Sudden Death (1995),Action
11 10,GoldenEye (1995),Action|Adventure|Thriller
```

Abbildung 4 - Rohdaten: MovieLens, movies.csv

IMDb Dateien sind in tsv (Tab Separated Values) formatiert und in UTF-8 kodiert. Die erste Reihe ist die Titelzeile.

```
imdb_ratings.tsv X
C: > Users > nico > Dropbox > HSLU > AIML > Semester_6 > DBS > DBS 23 > Projekt > DATA > NEW > imdb_ratings.tsv
1  tconst  averageRating  numVotes
2  tt0000001  5.7 1958
3  tt0000002  5.8 263
4  tt0000003  6.5 1791
5  tt0000004  5.6 179
6  tt0000005  6.2 2594
7  tt0000006  5.1 177
8  tt0000007  5.4 812
9  tt0000008  5.4 2096
10 tt0000009  5.3 204
```

Abbildung 5 - Rohdaten : IMDb, imdb_ratings.tsv

4.2 Datenimport und Datentransformation

Um die Rohdaten in die Tabellen unserer Filme Datenbank zu laden, sind wir wie folgt vorgegangen:

1. Erstellen der Datenbank und aller Tabellen mittels 'Forward Engineering' via MySQL Workbench basierend auf dem Datenbankschema.
2. Zunächst werden alle Datensets der Rohdaten mittel dem LOAD-Befehl von MySQL in temporäre Tabellen geladen.
3. Die geladenen Daten werden nun so transformiert, dass sie der Form und den Regeln unserer Zieldatenbank entsprechen (UPDATE-Befehl).
4. Die aufbereiteten Daten werden aus den temporären Tabelle mittels 'INSERT INTO SELECT FROM tmp_' in unsere Zieldatenbank übertragen.
5. Die temporären Daten können zu einem späteren Zeitpunkt wieder gelöscht werden.

Dieses ETL (Extract, Transform.Load) Verfahren wird in den nächsten Kapiteln mittels den entsprechenden SQL Skripts beschrieben.

Da sich die zu ladenden CSV-Dateien lokal auf der Festplatte befinden muss erst noch die Erlaubnis für das Laden lokaler Dateien auf Clientseite über die MySQL-Connection im Feld Others hinzugefügt werden. Diese ist standardgemäss aus Sicherheitsgründen auf 0 gesetzt.

```
SET GLOBAL local_infile = 1;  
SET SQL_SAFE_UPDATES = 0;
```

4.2.1 Erstellen der Zieldatenbank

Unser Datenbankschema wurde mittels der MySQL Workbench erstellt (siehe Kapitel 2). Das erlaubt uns, die notwendigen SQL-Skripts basierend auf dem graphischen Entwurf zu generieren (Forward Engineering). Das Script enthält die Erstellung der DB, alle Tabellen mit ihren Attributen und alle Beziehungen (Anhang A) DDL Filmdatenbank).

4.2.2 Erstellen der temporären Tabellen

Im zweiten Schritt übertragen wir die Rohdaten aus den csv und tsv Quelldateien in temporäre Tabellen. Dazu erstellen wir zunächst die temporären Tabellen in unserer Datenbank:

```
-- Movielens temp_bewertungen  
CREATE TEMPORARY TABLE temp_bewertungen (  
  benutzer_id INT,  
  filme_id INT,  
  rating FLOAT,  
  timestamp INT  
);  
-- IMDb temp_imdb_bewertungen  
CREATE TEMPORARY TABLE temp_imdb_bewertungen (  
  imdb_id VARCHAR(255),  
  durchschnittsbewertung FLOAT,  
  anzahl_bewertungen INT  
);  
-- temp_filme für MovieLens Filme  
CREATE TEMPORARY TABLE temp_filme (  
  movielens_id INT,
```

```

        titel VARCHAR(200),
        genres VARCHAR(200),
        jahr INT
    );
-- temp_imdb_filme für IMDb Filme Jahr und ID Transformation
CREATE TEMPORARY TABLE temp_imdb_filme (
    imdb_id INT UNSIGNED,
    titel_typ VARCHAR(20),
    titel VARCHAR(200),
    originaltitel VARCHAR(200),
    ist_erwachsen BOOLEAN,
    start_jahr YEAR,
    end_jahr YEAR,
    dauer_minuten INT UNSIGNED,
    genres VARCHAR(255)
);
-- temp_imdb_genres
CREATE TEMPORARY TABLE temp_imdb_genres (
    imdb_id INT,
    genre_name VARCHAR(255)
);
-- temp_ml_genres (neu)
CREATE TEMPORARY TABLE temp_ml_genres (
    movielens_id INT,
    genre_name VARCHAR(255)
);
-- temp_links
DROP TEMPORARY TABLE IF EXISTS temp_links;
CREATE TEMPORARY TABLE temp_links (
    movielens_id INT,
    tt_imdbId VARCHAR(255),
    tmdbId INT
);
CREATE TEMPORARY TABLE temp_genome_tags (
    tagId INT,
    tag VARCHAR(255)
);
CREATE TEMPORARY TABLE temp_genome_scores (
    movieId INT,
    tagId INT,
    relevance FLOAT
);
-- temp_benutzer_tag_relevanz
CREATE TEMPORARY TABLE temp_benutzer_tag_relevanz (
    benutzer_id INT,
    movielens_id INT,
    tag_name VARCHAR(255),
    timestamp INT
);

```

4.2.3 Laden der temporären Tabellen

Nun laden wir die Datenfiles in die temporären (tmp) Tabellen. Dazu kann der 'LOAD Befehl' entsprechend konfiguriert werden:

- Separator ist TAB oder COMMA. Bei den MovieLens Files handelt es sich ausschließlich um **CSV**-Dateien:
FIELDS TERMINATED BY ','
Feldeinträge, die über ein Komma verfügen, können durch optionales Anführungs- und Schlusszeichen (""") als zusammengehöriger Wert interpretiert (escaped) werden, e.g.
"*Cry, the Beloved Country* (1995)":
FIELDS TERMINATED BY ','
- Zeilen werden durch '\n' getrennt:
LINES TERMINATED BY '\n'
- Die erste Linie mit den Feldbezeichnungen kann ignoriert werden:
IGNORE 1 LINES;

Laden der Movielens Bewertungen

```
LOAD DATA LOCAL INFILE 'C:/Users/nico/Dropbox/HSLU/AI ML/Semester_6/DBS/DBS
23/Projekt/DATA/NEW/ml-latest/ratings.csv'
INTO TABLE temp_bewertungen
FIELDS TERMINATED BY ','
ENCLOSED BY ''
LINES TERMINATED BY '\n'
IGNORE 1 LINES;
```

Laden der IMDb Bewertungen

```
LOAD DATA LOCAL INFILE 'C:/Users/nico/Dropbox/HSLU/AI ML/Semester_6/DBS/DBS
23/Projekt/DATA/NEW/imdb_ratings.tsv'
INTO TABLE temp_imdb_bewertungen
FIELDS TERMINATED BY '\t'
ENCLOSED BY ''
LINES TERMINATED BY '\n'
IGNORE 1 LINES
(imdb_id, durchschnittsbewertung, anzahl_bewertungen);
```

Laden der Movielens Filme

```
LOAD DATA LOCAL INFILE 'C:/Users/nico/Dropbox/HSLU/AI ML/Semester_6/DBS/DBS
23/Projekt/DATA/NEW/ml-latest/movies.csv'
INTO TABLE temp_filme
FIELDS TERMINATED BY ','
ENCLOSED BY ''
LINES TERMINATED BY '\n'
IGNORE 1 LINES
(movielens_id, titel, genres);
```

IMDB-Filme Transformation der Nullwerte und IMDb-ID (=tconst)

```
LOAD DATA LOCAL INFILE 'C:/Users/nico/Dropbox/HSLU/AI ML/Semester_6/DBS/DBS
23/Projekt/DATA/NEW/imdb_title_basics.tsv'
INTO TABLE temp_imdb_filme
FIELDS TERMINATED BY '\t'
```

```

LINES TERMINATED BY '\n'
IGNORE 1 LINES
(@imdb_id, titel_typ, @titel, originaltitel, ist_erwachsen, @start_jahr, @end_jahr,
@dauer_minuten, genres)
SET imdb_id = CAST(REGEXP_REPLACE(@imdb_id, 'tt', '') AS UNSIGNED),
    titel = @titel,
    start_jahr = NULLIF(@start_jahr, '\\N'),
    end_jahr = NULLIF(@end_jahr, '\\N'),
    dauer_minuten = NULLIF(@dauer_minuten, '\\N');

```

Laden von MovieLens Links

```

LOAD DATA LOCAL INFILE 'C:/Users/nico/Dropbox/HSLU/AIML/Semester_6/DBS/DBS
23/Projekt/DATA/NEW/ml-latest/links.csv'
INTO TABLE temp_links
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES
(movieLens_id , tt_imdbId, tmdbId);

```

Laden der MovieLens Genome Tags

```

LOAD DATA LOCAL INFILE 'C:/Users/nico/Dropbox/HSLU/AIML/Semester_6/DBS/DBS
23/Projekt/DATA/NEW/ml-latest/genome_tags.csv'
INTO TABLE temp_genome_tags
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES;

```

Laden der MovieLens Genome Scores

```

LOAD DATA LOCAL INFILE 'C:/Users/nico/Dropbox/HSLU/AIML/Semester_6/DBS/DBS
23/Projekt/DATA/NEW/ml-latest/genome_scores.csv'
INTO TABLE temp_genome_scores
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES;

```

Laden der MovieLens benutzer_tag_relevanz

```

-- TagID und Tagnamen in temporäre Tabelle temp_benutzer_tag_relevanz
LOAD DATA LOCAL INFILE 'C:/Users/nico/Dropbox/HSLU/AIML/Semester_6/DBS/DBS
23/Projekt/DATA/NEW/ml-latest/tags.csv'
INTO TABLE temp_benutzer_tag_relevanz
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES;

```

4.2.4 Daten Transformationen

Nachdem die Daten geladen wurden müssen die Daten bereinigt oder transformiert werden, damit unsere Datenbankabfragen auf einem aussagekräftigen und konsistenten Datenbestand erfolgen. Die folgende Transformationen werden dazu benötigt:

Titeljahr

Damit das Jahr in einem separaten Attribut (jahr, INT) geführt werden kann, extrahieren wir das Jahr aus dem Filmtitel, welcher diese Jahresinformation in Klammern enthält. Der folgende Code sucht zuerst nach Jahreszahlen in Klammern und entfernt danach die Jahresinformation.

```
UPDATE temp_filme
SET jahr = CASE
    WHEN SUBSTRING_INDEX(SUBSTRING_INDEX(titel, '(', -1), ')', 1) REGEXP '^[0-9]{4}$' THEN
        CAST(SUBSTRING_INDEX(SUBSTRING_INDEX(titel, '(', -1), ')', 1) AS SIGNED)
    ELSE
        NULL
    END,
    titel = TRIM(BOTH ' ' FROM RTRIM(SUBSTRING(titel, 1, CHAR_LENGTH(titel) -
        CHAR_LENGTH(SUBSTRING_INDEX(titel, '(', -1)) - 1)));
```

Filtern von Sonderzeichen für temp_filme, temp_imdb_filme

```
UPDATE temp_filme
SET titel = LOWER(REPLACE(REPLACE(REPLACE(titel, ' ', ''), '.', ''), ',', ''));

UPDATE temp_imdb_filme
SET titel = LOWER(REPLACE(REPLACE(REPLACE(titel, ' ', ''), '.', ''), ',', ''));
```

Zeitmarken

Die Rohdaten der Tabellen 'bewertungen' und 'tags' enthalten das Unix-Zeitmarkenformat. Diese müssen in den *Datetime* Datentyp umgewandelt werden. Dazu bietet MySQL die folgende Funktion:

```
SET zeitstempel = FROM_UNIXTIME(@unix_timestamp);
```

Eindeutige IMDb ID tconst

Da sich die *tconst* (eindeutige IMDb-Filme ID) im MovieLens und im IMDb Datenset um den Prefix „tt“ unterscheiden, muss dieser mittels dem Befehl *REGEXP_REPLACE(@imdb_id, 'tt', '')* bereinigt werden, sodass die Tabelle *imdb_bewertungen* und *filme* via dem transformiertem Integer-Fremdschlüssels *tconst* verbunden werden können:

```
SET imdb_id = CAST(REGEXP_REPLACE(@imdb_id, 'tt', '') AS UNSIGNED)
```

Genres

Die *MovieLens* Genre-Tabelle extrahiert Genres aus der **temp_filme** Tabelle, die mit Kommas getrennt sind, und fügt sie in die **temp_imdb_genres** Tabelle ein.

```
INSERT INTO temp_ml_genres (movielens_id, genre_name)
WITH RECURSIVE genre_splitting (movielens_id, genres, genre, rest) AS (
    SELECT movielens_id, genres,
           TRIM(SUBSTRING_INDEX(genres, ',', 1)) AS genre,
           SUBSTRING_INDEX(genres, ',', -1) AS rest
    FROM temp_ml_filme
    UNION ALL
    SELECT movielens_id, genres,
           TRIM(SUBSTRING_INDEX(rest, ',', 1)) AS genre,
           SUBSTRING_INDEX(rest, ',', -1) AS rest
    FROM genre_splitting
    WHERE rest <> genre
)
SELECT DISTINCT movielens_id, genre
FROM genre_splitting;
```

Tags

Für die Tags wollen wir den Benutzer, die Tag-ID und den Relevanzwert verbinden und die Tagnamen mit Tag-IDs in eine separate Referenztabelle namens *Tag* ablegen, um die 3. Normalform zu gewährleisten, in der keine transitiven Abhängigkeiten vorkommen. Auch ermöglicht sie ein einfaches Erneuern von Tagnamen mit Beibehaltung der Konsistenz.

-- Zeitstempeltransformation in *benutzer_tag_relevanz*-Tabelle:

```
INSERT INTO benutzer_tag_relevanz (benutzer_id, film_id, tag, zeitstempel)
SELECT benutzer_id, filme_id, tag, FROM_UNIXTIME(timestamp)
FROM temp_benutzer_tag_relevanz;
```

-- Matchen der Tagnamen und Aktualisieren mit Tag-IDs

```
ALTER TABLE temp_benutzer_tag_relevanz
ADD COLUMN tag_id INT;

UPDATE temp_benutzer_tag_relevanz t1
JOIN temp_genome_tags t2 ON t1.tag = t2.tag
SET t1.tag_id = t2.tagId;
```

Weitere Datenbereinigungen und -transformationen sind im Anhang B)

4.2.5 Migration in die Zieldatenbank

Nach dem Bereinigen der Daten müssen die Informationen der temporären Tabellen in die Zieldatenbank migriert werden. Dabei spielt die Reihenfolge der INSERT INTO Befehle eine wichtige Rolle, die Daten müssen in hierarchischer Ordnung eingefügt werden, z.B. kann eine 'bewertung' erst migriert werden nachdem die 'bewertungsquellen' sowie die 'filme' in die Zieldatenbank übertragen worden sind.

Migration der Film Informationen

-- Migrieren der Daten aus der temp_imdb_filme Tabelle in die filme Tabelle

```
INSERT IGNORE INTO filme (imdb_id, titel_typ, titel, originaltitel, ist_erwachsen,
start_jahr, end_jahr, dauer_minuten)
SELECT imdb_id, titel_typ, titel, originaltitel, ist_erwachsen, start_jahr, end_jahr,
dauer_minuten
FROM temp_imdb_filme;
```

-- Hinzufügen von MovieLens Links (Datenbankreferenzen) zur filme-Tabelle:

```
UPDATE filme
JOIN temp_links ON filme.filme_id = temp_links.movielens_id
SET filme.imdb_id_tt = temp_links.tt_imdbId;
```

-- MovieLens und IMDb Filme via imdb_id verbinden (! Inkonsistenzen, Benötigt weitere, noch offene Schritte)

```
SELECT temp_imdb_filme.imdb_id, filme.imdb_id_tt
FROM temp_imdb_filme
JOIN filme ON temp_imdb_filme.imdb_id = filme.imdb_id_tt;
```

-- Transformieren und Migrieren der genres-Tabelle aus der IMDB Tabelle temp_imdb_filme

```
INSERT INTO genres (genre_name)
WITH RECURSIVE genre_splitting (imdb_id, genres, genre, rest) AS (
    SELECT imdb_id, genres,
        TRIM(SUBSTRING_INDEX(genres, '|', 1)) AS genre,
        SUBSTRING_INDEX(genres, '|', -1) AS rest
    FROM temp_imdb_filme
    WHERE genres <> '(no genres listed)'
    UNION ALL
    SELECT imdb_id, genres,
        TRIM(SUBSTRING_INDEX(rest, '|', 1)) AS genre,
        SUBSTRING_INDEX(rest, '|', -1) AS rest
    FROM genre_splitting
    WHERE rest <> genre
)
SELECT DISTINCT imdb_id, genre
FROM genre_splitting;
```

-- Füllen der genres-Tabelle mit noch nicht vorhandenen Genrenamen aus der Movielens Tabelle temp_ml_genres (! Inkonsistenzen, Benötigt weitere, noch offene Schritte)

```
INSERT IGNORE INTO genres (genre_name)
SELECT DISTINCT genre_name
FROM temp_ml_genres
WHERE LOWER(genre_name) NOT IN (SELECT LOWER(genre_name) FROM genres);
```

-- Verknüpfung von MovieLens genres zu Filme

```
INSERT INTO filme_genres (filme_id, genre_id)
SELECT filme.filme_id, genres.genre_id
FROM temp_ml_genres
JOIN filme ON temp_ml_genres.movielens_id = filme.movielens_id
JOIN genres ON temp_ml_genres.genre_name = genres.genre_name;
```

-- Verknüpfung von IMDb genres zu Filme

```
INSERT INTO filme_genres (filme_id, genre_id)
SELECT filme.filme_id, genres.genre_id
FROM temp_imdb_filme
JOIN filme ON temp_imdb_filme.imdb_id = filme.imdb_id
JOIN genres ON FIND_IN_SET(genres.genre_name, REPLACE(temp_imdb_filme.genres, ',', '|'))
WHERE temp_imdb_filme.genres IS NOT NULL;
```

-- Migration der Benutzer Informationen

-- Movielens benutzer IDs aus temp_bewertungen migrieren

```
INSERT INTO benutzer (benutzer_id)
SELECT DISTINCT benutzer_id
FROM temp_bewertungen;
```

-- Migration der Bewertungen

-- Bewertungsquellen hinzufügen

```
INSERT INTO bewertungsquellen (source_id, source_name)
VALUES (1, 'Movielens'), (2, 'IMDb');
```

-- Migrieren der bewertungen-Tabelle aus temp_bewertungen:

```
INSERT INTO bewertungen (filme_id, source_id, durchschnittsbewertung, anzahl_bewertungen,
zeitstempel)
SELECT f.filme_id, 1 as source_id, AVG(r.rating) as durchschnittsbewertung,
COUNT(r.rating) as anzahl_bewertungen, FROM_UNIXTIME(AVG(r.timestamp)) as zeitstempel
FROM temp_bewertungen r
JOIN filme f ON r.filme_id = f.movielens_id
GROUP BY f.filme_id;
```

-- Migrieren der bewertungen-Tabelle aus temp_imdb_bewertungen:

```
INSERT INTO bewertungen (filme_id, source_id, durchschnittsbewertung, anzahl_bewertungen)
SELECT f.filme_id, 2 as source_id, t.durchschnittsbewertung, t.anzahl_bewertungen
FROM temp_imdb_bewertungen t
JOIN filme f ON t.imdb_id = f.imdb_id;
```

-- Migrieren der temp_genome_tags in die Tagstabelle


```
INSERT INTO tags (tag_id, tag_name)
SELECT tagId, tag
FROM temp_genome_tags;
```

-- Migrieren der temp_benutzer_tag_relevanz und temp_genome_scores Tabellen in die Benutzerrelevanztabelle

```
INSERT INTO benutzerrelevanzen (benutzer_id, filme_id, tag_id, relevanz_wert,
zeitstempel)
SELECT t1.benutzer_id, t1.moviels_id, t1.tag_id, t3.relevance,
FROM_UNIXTIME(t1.timestamp)
FROM temp_benutzer_tag_relevanz AS t1
JOIN temp_genome_scores AS t3 ON t1.moviels_id = t3.movieId AND t1.tag_id = t3.tagId;
```

4. Daten analysieren & auswerten

- Queries in Datenbank-Syntax (z.B. SQL)
- Verbale Beschreibung der Queries
- Zusammenhang Queries mit Use Case aufzeigen

Auf der Basis unserer Daten ist es nun möglich, unterschiedlichste Fragen zu stellen, indem die Basistabellen mittels einer Abfrage (Query) miteinander verbunden werden. Diese logische Relation ist im DBMS eine Sicht (View), welche auch gespeichert werden kann.

Die Aufgabe einer Sicht ist, den Zugriff auf die Datenbank zu vereinfachen. Das Bereitstellen 'Use Case' spezifischer Sichten erlaubt somit einen einfacheren Datenzugriff ohne die Verletzung der Normalisierung des Datenbankschemas. Sichten werden nach der Speicherung vom DBMS analysiert und vereinfacht/optimiert.

Nachdem wir die ersten Sichten für unsere 'Use Cases' erstellt haben lernten wir, dass es eine weitere Möglichkeit gibt, die Abfrage zu beschleunigen und auch die Netzwerklast zu verringern: die materialisierten Sichten (auch indexed Views). Sie eignen sich bei Aufgaben, welche grössere Datenmengen zu einem bestimmten Zeitpunkt dem Benutzer zur Verfügung gestellt wird.

Wir verwenden diese 'Materialized Views' bei der Implementierung unserer 'Haupt Use-Cases', um die Zugriffe auf die Datenmenge zu beschleunigen.

4.1. Empfehlungen über ähnliche Benutzer – Kollaborative Filterung

Die SQL-Abfrage für den UC02 "Wähle Empfehlungen über ähnliche Benutzer – Kollaborative Filterung" wählt Filme aus, die von Benutzern bewertet wurden, die unseren Ausgangsfilm (Film ID 325) hoch bewertet haben (siehe auch 2.1.1). Dabei werden die Kennzahlen für den UC02 mittels den folgenden Materialized Views vorbereitet:

Pre-Processing I (Berechnung des Gewichteten Durchschnittes für alle Datenquellen):

1. Wir reduzieren die Filmmenge der ermittelten Benutzer (Bewertungen 3,5 oder höher)

```
CREATE TABLE bewertungen_35_und_hoehere AS
SELECT *
FROM materialized_view_benutzer_bewertungen_summe_und_anzahl
WHERE benutzer_bewertung >= 3.5;
```

2. Wir bestimmen die Gesamtbewertungen pro Datenquelle (IMDB) um den 'Gewichteten Durchschnitt' zu ermitteln. Dazu erstellen wir eine temporäre Tabelle mit den Gesamtbewertungen pro Datenquelle.

```
CREATE TEMPORARY TABLE total_ratings AS
SELECT source_id, COUNT(*) as total
FROM benutzer_bewertungen
WHERE source_id = 1
GROUP BY source_id;
```

3. Wir berechnen den Gewichteten Durchschnitt pro Film und Datenquelle, und speichern das Ergebnis in einer temporären Tabelle. Den gewichteten Durchschnitt berechnen wir nur für die Movie Lens Filme, die IMDB Filme sind bereits gewichtet, somit können wir die Bewertung ohne Berechnung übernehmen.

```
CREATE TEMPORARY TABLE benutzer_bewertungen_mit_gd AS
SELECT
    bb.benutzer_id,
    bb.filme_id,
    f.titel,
    bb.anzahl_bewertungen,
    CASE
        WHEN bb.source_id = 1 AND bb.benutzer_id != 283229 THEN
            (bb.anzahl_bewertungen * bb.bewertung) / tr.total
        ELSE
            bb.bewertung
    END AS gewichteter_durchschnitt,
    CASE
        WHEN bb.benutzer_id = 283229 THEN 2
        ELSE 1
    END AS source_id
FROM
    benutzer_bewertungen bb
JOIN
    filme f ON f.filme_id = bb.filme_id
JOIN
    total_ratings tr ON bb.source_id = tr.source_id;
```

Pre-Processing II (Bestimmung des Filmranges)

4. Wir Berechnen den Rang aufgrund des gewichteten Durchschnitts sowie der Anzahl der Bewertungen

```
CREATE TEMPORARY TABLE benutzer_bewertungen_mit_rank AS
SELECT
    bbg.*,
    RANK() OVER (
        PARTITION BY bbg.source_id
        ORDER BY (bbg.gewichteter_durchschnitt / SQRT(bbg.anzahl_bewertungen))
        DESC
    ) AS quellen_rang
FROM
    benutzer_bewertungen_mit_gd bbg;
```

5. Wir bestimmen die Anzahl der Bewertungen pro Film

```
CREATE TEMPORARY TABLE filme_source_counts AS
SELECT
    filme_id,
    COUNT(DISTINCT source_id) as source_count
FROM
    benutzer_bewertungen
GROUP BY
    filme_id;
```

6. Wir berechnen den kombinierten Rang über alle Datenquellen pro Film

```
CREATE TABLE materialized_view_benutzer_bewertungen_summe_und_anzahl AS
SELECT
    bbr.*,
    fsc.source_count,
    CASE
        WHEN fsc.source_count = 1 THEN
            bbr.quellen_rang
        ELSE
            (SELECT AVG(bbr.quellen_rang) FROM benutzer_bewertungen_mit_rank WHERE
             filme_id = bbr.filme_id)
    END AS kombinierter_rang,
    CASE
        WHEN fsc.source_count = 1 THEN
            bbr.quellen_rang
        ELSE
            (SELECT AVG(bbr.quellen_rang) FROM benutzer_bewertungen_mit_rank WHERE
             filme_id = bbr.filme_id)
    END AS new_quellen_rang,
    NULL AS benutzer_bewertung
FROM
    benutzer_bewertungen_mit_rank bbr
JOIN
    filme_source_counts fsc ON bbr.filme_id = fsc.filme_id;
```

7. Erstelle einen Index für die Benutzer- und Filmtabellen

```
#CREATE INDEX idx_benutzer_filme
#ON materialized_view_benutzer;
```

Main Query für UC02 basierend auf den 'Materialized views'

Nach der Bereitstellung aller Kennzahlen mittels 'Materialized views' kann jede Filmempfehlung schnell und mittels der folgenden einfachen Abfrage bestimmt werden:

```
-- v4.5
WITH benutzer_die_start_film_bewertet AS (
    SELECT
        benutzer_id
    FROM
        mat_view_bewertungen_35_und_hoeher_subs_index #Entferne "_subs" für vollständige Tabelle
    WHERE
        filme_id = 325
),
gefilterte_filme AS (
    SELECT
        mv.filme_id,
        mv.titel,
        mv.kombinierter_rang,
        COUNT(*) AS anzahl_benutzer
    FROM
        mat_view_bewertungen_35_und_hoeher_subs_index mv
    JOIN
        benutzer_die_start_film_bewertet u ON mv.benutzer_id = u.benutzer_id
    WHERE
        mv.filme_id != 325
    GROUP BY
        mv.filme_id,
        mv.titel,
        mv.kombinierter_rang
    HAVING
        COUNT(*) >= 10
)
SELECT
    gefilterte_filme.titel,
    gefilterte_filme.kombinierter_rang
FROM
    gefilterte_filme
ORDER BY
    anzahl_benutzer DESC,
    kombinierter_rang DESC
LIMIT 20;
```

Kollaboratives Filtern Abfrage mit dem Ausgangsfilm 'Inception', sortiert nach 1. Anzahl Bewertungen und 2. Kombinerter Rang:

V1

titel	kombinierter_rang
The Matrix	1302831.9895
The Shawshank Redemption	550554.9196
The Dark Knight	3000742.2188
Fight Club	1690366.0110
The Lord of the Rings: The Return of the King	2530340.9590
The Lord of the Rings: The Fellowship of the Ring	2312969.2588
Forrest Gump	1137773.0217
Pulp Fiction	1196952.8039
The Lord of the Rings: The Two Towers	2591391.8613
Star Wars: Episode IV - A New Hope	1540054.1025
The Silence of the Lambs	1133857.1971
Memento	2963097.3975
Star Wars: Episode V - The Empire Strikes Back	1955235.7752
Interstellar	6435646.2162
WALL·E	5562033.5465
Schindler's List	1409713.3060
The Godfather	1735208.1681
Gladiator	3205593.4226
The Usual Suspects	1569818.8345
Batman Begins	5059699.7942

Abbildung 6 – Resultat Kollaboratives Filtern

4.2. Empfehlungen über Filmtag

Die SQL-Abfrage für den UC03 "Wähle Empfehlungen über Filmtag" (siehe 2.1.1) besteht aus zwei Ausführungsschritten:

1. Der erste Schritt erstellt eine Ansicht welche die Filme und ihre zugehörigen Tags, sowie den durchschnittlichen Relevanzwert jedes Tags, enthält ("movie_tags_grouped"). Dazu verbinden wir die beiden Tabellen "benutzerrelevanzen" und "tags" mittels der "tag_id" und gruppieren nach "filme_id" und "tag_id".

```
CREATE TABLE mat_view_gruppierte_film_tags (
  filme_id int unsigned NOT NULL,
  tag_id int unsigned NOT NULL,
  tag_name varchar(255) NOT NULL,
  relevanz double DEFAULT NULL,
  KEY idx_gruppierte_film_tags_filme_id_tag_id (filme_id, tag_id) )
ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

INSERT INTO mat_view_gruppierte_film_tags (filme_id, tag_id, tag_name, relevanz)
SELECT br.filme_id, br.tag_id, t.tag_name, AVG(br.relevanz_wert) as relevanz
FROM benutzerrelevanzen br JOIN tags t ON br.tag_id = t.tag_id
GROUP BY br.filme_id, br.tag_id, t.tag_name;
WITH input_movie_tags AS (
  SELECT *
```

```

FROM mat_view_gruppierte_film_tags
WHERE filme_id = 299
)

```

2. Im zweiten Teil verwenden wir diese Ansicht ("movie_tags_grouped"), um Filme mit ähnlicher Tag-Verteilung zu dem vom Benutzer ausgewählten Ausgangsfilm zu finden:

- Zuerst wird eine temporäre Tabelle "input_movie_tags" erstellt, die alle Tags des ausgewählten Films enthält.
- Dann wird die Gesamtdifferenz der Relevanzwerte zwischen den Tags des Eingabefilms und den Tags jedes anderen Films berechnet. Die Filme werden nach dieser Gesamtdifferenz aufsteigend sowie mit der Anzahl der Tags absteigend sortiert. Für den Benutzer werden die ersten 10 Filme angezeigt.

```

SELECT
    mtg.filme_id,
    f.titel,
    SUM(ABS(mtg.relevanz - imt.relevanz)) AS total_difference
FROM
    mat_view_gruppierte_film_tags mtg
JOIN
    input_movie_tags imt ON mtg.tag_id = imt.tag_id
JOIN
    filme f ON mtg.filme_id = f.filme_id
WHERE
    mtg.filme_id <> 299
GROUP BY
    mtg.filme_id,
    f.titel
ORDER BY
    total_difference ASC,
    COUNT(mtg.tag_id) DESC
LIMIT 10;

```

Tagabfrage (Version 1) für den Ausgangsfilm *Poltergeist* 'ohne Bestrafung' von nicht abgedeckten Tags:

V1	filme_id	titel	total_difference
	8972	The Story on Page One	0
	38038	This Time Around	0.00025018814775192716
	2490	Death Valley Rangers	0.00025018814775192716
	4840	Soft Matter	0.0005003762955038508
	2700	The Hunters	0.0005003762955038543
	8655	Whiskey Tango Foxtrot	0.0005003762955038543
	2106	Practical Magic	0.0007505625792125943
	6441	His Name Was Holy Ghost	0.0007505625792125943
	26386	Youth of the Son	0.0010007488629213412
	946	A Perfect World	0.001250937010673265

Abbildung 7 – Resultat Tagabfrage 'ohne Bestrafung'

Version 2 / Tagabfrage Verbesserung

In Version 1 wird nur die absolute Differenz zwischen den Relevanzen der gemeinsamen Tags verglichen. Tags, die in einem Film vorhanden sind, aber nicht im anderen, werden in dieser Version nicht berücksichtigt.

Version 2 bringt eine Reihe von Verbesserungen mit sich:

1. **Einbeziehung aller Tags:** In der verbesserten Version werden alle Tags der Filme berücksichtigt, nicht nur die, die beide Filme gemeinsam haben. Dies wird durch die Verwendung von LEFT JOIN anstatt JOIN erreicht. Dadurch werden auch die Filme berücksichtigt, die einige spezifische Tags des Eingabefilms nicht haben.
2. **Bestrafung für fehlende Tags:** In Version 2 wird ein neues Merkmal namens "fehlend" eingeführt. Dieses Merkmal ist auf 1 gesetzt, wenn ein Tag im betrachteten Film fehlt, der im Eingabefilm vorhanden ist. Diese zusätzliche "Bestrafung" führt dazu, dass Filme mit fehlenden Tags in der Gesamtdifferenz höher eingestuft werden.
3. **Normalisierte Differenzberechnung:** In der neuen Version wird die Differenz zwischen den Relevanzen der Tags nicht nur absolut, sondern normalisiert berechnet. Dies bedeutet, dass der Unterschied zwischen den Relevanzen durch den größeren der beiden Relevanzwerte dividiert wird. Dies stellt sicher, dass die Differenz zwischen den Relevanzen in einem angemessenen Verhältnis steht und große Unterschiede stärker gewichtet werden.

Zusammengefasst ermöglicht die verbesserte Abfrage in Version 2 eine gründlichere und präzisere Analyse der Tag-Übereinstimmungen zwischen Filmen. Die Einbeziehung aller Tags und die Bestrafung für fehlende Tags sorgen für eine umfassendere Übereinstimmungsanalyse, während die normalisierte Differenzberechnung eine differenzierte Bewertung der Abweichungen ermöglicht.

*Tagabfrage (Version 2/Finale Version) für den Ausgangsfilm **Poltergeist** 'mit Bestrafung' von nicht abgedeckten Tags.*

V2

	filme_jd	titel	gesamte_differenz
►	2856	Die Totale Therapie	0.040650427918820606
	6736	Judgment Day	0.10569108835940352
	3395	Grand Piano	0.10975607663484269
	6837	The Hillz	0.11454361503573715
	69574	Breathless (Ddongpari)	0.14345989523410652
	26616	Princess Iron Fan	0.20934962200726195
	6329	The Egyptian	0.23396228257183369
	6228	Song of the Saddle	0.27218936363122315
	32316	50 Million Frenchmen	0.28462344802286926
	5754	The Hagstone Demon	0.2865853684218133

Abbildung 8 – Resultat Tagabfrage 'mit Bestrafung'

5. Effizienz & Performance

- Datenmengen und Engpässe
- Massnahmen zur Leistungsoptimierung
- Ausführungspläne und Laufzeiten vor und nach der Optimierung

Die Effizienz und Performance einer Datenbank hängen primär von einem guten Design der Datenbank ab. Zu viel Redundanz und unnötige Beziehungen führen zu komplexen Abfragen, welche wiederum meistens über eine schlechte Performance verfügen. Da hilft uns unser Datenbankdesign, welcher unterschiedliche Datenquellen (IMDB und Movie Lens) in einer gemeinsamen, übersichtlichen und einfacher Datenstruktur zusammenhält. Aber auch dieser Lösungsansatz verlangt eine zusätzliche Laufzeitanalyse, gerade weil die Datenmengen in diesen gemeinsamen Tabellen hoch sind und somit die Abfragen wesentlich von weiteren Optimierungs-Massnahmen profitieren können.

5.1. Datenmengen

Tabellenname	TABLE_ROWS
avg_quellen_rang	54'075
benutzer	276'233
benutzer_bewertungen	27'507'168
benutzer_blacklist	0
benutzer_relevanzen	438'344
bewertungsquellen	2
filme	4'253'202
filme_genres	2'496'561
filme_source_counts	937'980
genres	32
tags	64'247

Datenmengen und mögliche Engpässe: Die Datenbank enthält Tabellen wie Filme, Genres, Benutzerbewertungen mit grossen Datenmengen. Engpässe können auftreten, wenn komplexe JOIN-Operationen oder Gruppierungen auf diesen Tabellen durchgeführt werden, insbesondere bei Berechnungen mit mehreren Tabellen.

Bei grossen Datenbankmengen ist es zudem sinnvoll die 'Globalen Variablen' der Datenbank gemäss der Datenanalyse anzupassen, in unserem Fall ist dies:

```
SET GLOBAL innodb_buffer_pool_size = 8589934592;
-- Sets the size of the InnoDB buffer pool to 8 gigabytes (in bytes and 50% of the available
memory).
-- The buffer pool is used to cache data and indexes in memory for faster access.

SET GLOBAL innodb_buffer_pool_instances = 8

SET GLOBAL tmp_table_size = 8589934592;
-- Sets the size of the temporary table in-memory storage engine to 8 gigabytes (in bytes).
-- Temporary tables are used to store intermediate results during query execution.
-- For 8GB (50% of 16GB): 8589934592

SET GLOBAL wait_timeout = 40800;
```


5.2. Massnahmen zur Leistungsoptimierung

In unserem Projekt haben wir Materialized Views und Indizes verwendet, um die Abfragegeschwindigkeit zu optimieren. Hier ist eine Zusammenfassung der Massnahmen zur Leistungsoptimierung:

1. **Materialized Views:** Wir haben Materialized Views erstellt, um einige Berechnungen vorab durchzuführen und die Ergebnisse zu speichern. Zum Beispiel haben wir die Materialized View **benutzer_bewertungen_summe_und_anzahl** erstellt, um die Summe und Anzahl der normalisierten Bewertungen pro Benutzer und Film zu speichern. Dadurch werden Berechnungen wie **SUM()** und **COUNT()** im Voraus durchgeführt, wodurch die Laufzeit der Abfragen, die diese Ergebnisse benötigen, erheblich verkürzt wird.
2. **Indizes:** Wir haben Indizes auf den wichtigsten Attributen der Tabellen erstellt, um die Geschwindigkeit der JOIN-Operationen und Filterungen zu erhöhen. Zum Beispiel haben wir Indizes auf **filme_id**, **bewertungs_id**, und **benutzer_id** erstellt, um die Abfrageleistung zu verbessern.

Damit wir diese Massnahmen verifizieren können, benötigen wir entsprechende Tools und Funktionen für die Messungen und Vergleiche:

3. **Ausführungspläne:** Wir haben den EXPLAIN-Befehl in MySQL verwendet, um die Ausführungspläne der Abfragen vor und nach unseren Optimierungs-Massnahmen zu analysieren. Dies hilft uns, den Einfluss von Indizes und Materialized Views auf den 'Datenbank Optimizer' zu beurteilen.
4. **Laufzeitmessungen:** Durch Messen der Laufzeiten vor und nach der Optimierung können wir quantifizieren, wie effektiv unsere Optimierungsmaßnahmen sind. Dies kann zum Beispiel durch die Verwendung von SQL-Timing-Funktionen oder externen Tools zur Messung der Abfragezeiten erfolgen.
5. **Subsampling:** Da wir mit sehr vielen Benutzerbewertungen arbeiten (1.5 Gigabyte), macht es Sinn, die Laufzeiten der Abfragen zunächst mit kleinerer Menge zu vergleichen. Dazu wird mit Hilfe der Funktion **SUB()** zufällig ¼ der Datenmenge ausgewählt.

Insgesamt haben wir durch den Einsatz von Materialized Views und Indizes die Abfragegeschwindigkeit in unserer Datenbank optimiert und somit die Leistung verbessert.

5.2.1. Indizes

Unser Datenbank-Design verwendet ausschliesslich eindeutige IDs als Primary Keys. Diese PKs sind indiziert und werden bei JOIN-Operationen oder WHERE-Filterungen verwendet. Zusätzlich können (nach der Migration) Index gesetzt werden um die Performance einzelner Abfragen zu unterstützen. Da das Setzen von Indexes Speicher und Zeit (beim INSERT) kostet, muss der Erfolg eines zusätzlichen Index bei der spezifischen Abfrage untersucht werden. Das Beispiel zeigt den zusätzlichen Index 'idx_bewertungen_benutzer_filme' auf unserer tabelle 'bewertungen_35_und-hoeher' welcher die kombinierten ID's 'benutzer_ID' und 'Filme_ID' enthält.

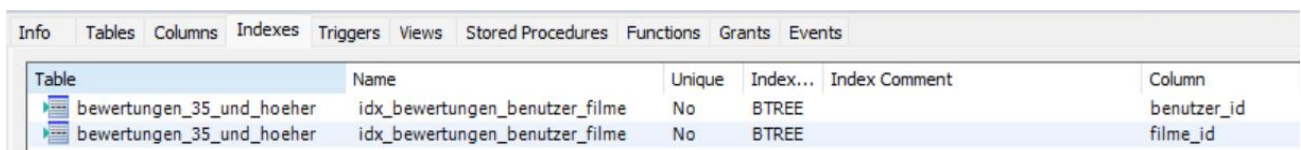


Table	Name	Unique	Index...	Index Comment	Column
bewertungen_35_und_hoeher	idx_bewertungen_benutzer_filme	No	BTREE		benutzer_id
bewertungen_35_und_hoeher	idx_bewertungen_benutzer_filme	No	BTREE		filme_id

Die folgenden Executionplans zeigen das Resultat mit und ohne Index auf. Wir sehen im Resultat mit Index, dass der kombinierte Index erfolgreich verwendet wird (grüner Non Unique Key Lookup). Die Ausführungszeit zeigt jedoch nur einen kleinen Gewinn auf: 116.534678 vs 117.2000483. Wir belassen den Index und werden ihn für die Leistungsoptimierung im UC02 einsetzen.

Executionplan		Resultat
A	<p>Query cost: 286391587168.49</p> <p>query_block #1</p> <p>ORDER</p> <p>filesort</p> <p>286391587168.49</p> <p>2545.70G rows</p> <p>Full Table Scan</p> <p>gefilterte_filme (materialized)</p> <p>GROUP</p> <p>tmp table</p> <p>25457 rows</p> <p>25.45G rows</p> <p>hash join</p> <p>1706824 rows</p> <p>16.82M rows</p> <p>Full Table Scan</p> <p>25458164 rows</p> <p>25.45G rows</p> <p>Full Table Scan</p> <p>bewertungen_35_und_hoeherbewertungen_35_und_hoeh2</p> <p>bewertungen_35_und_hoeh2</p>	<p>Resultat ohne Index</p> <p>Status,Duration</p> <p>starting,0.000234</p> <p>"Executing hook on transaction ",0.000022</p> <p>starting,0.000011</p> <p>"checking permissions",0.000007</p> <p>"checking permissions",0.000004</p> <p>"Opening tables",0.000383</p> <p>init,0.000012</p> <p>"System lock",0.000044</p> <p>optimizing,0.000009</p> <p>optimizing,0.000023</p> <p>statistics,0.000200</p> <p>preparing,0.000028</p> <p>"Creating tmp table",0.000103</p> <p>statistics,0.000010</p> <p>preparing,0.000033</p> <p>executing,117.200483</p> <p>end,0.000016</p> <p>"query end",0.000007</p> <p>"waiting for handler commit",0.204210</p> <p>"closing tables",0.000043</p> <p>"freeing items",0.000569</p> <p>"cleaning up",0.000785</p>
B	<p>Query cost: 1702857.29</p> <p>query_block #1</p> <p>ORDER</p> <p>filesort</p> <p>1702857.29</p> <p>15.14M rows</p> <p>Full Table Scan</p> <p>gefilterte_filme (materialized)</p> <p>GROUP</p> <p>tmp table</p> <p>18356 rows</p> <p>18.35M rows</p> <p>nested loop</p> <p>1706824 rows</p> <p>16.82M rows</p> <p>Full Table Scan</p> <p>16650136.36</p> <p>1 row</p> <p>Non-Unique Key Lookup</p> <p>bewertungen_35_und_hoeh2</p> <p>bewertungen_35_und_hoeh2_idx_bewertungen_benutzer_filme</p>	<p>Resultat mit Index</p> <p>Status,Duration</p> <p>starting,0.000141</p> <p>"Executing hook on transaction ",0.000005</p> <p>starting,0.000008</p> <p>"checking permissions",0.000006</p> <p>"checking permissions",0.000009</p> <p>"Opening tables",0.000171</p> <p>init,0.000007</p> <p>"System lock",0.000013</p> <p>optimizing,0.000004</p> <p>optimizing,0.000021</p> <p>statistics,0.000051</p> <p>preparing,0.000229</p> <p>"Creating tmp table",0.000067</p> <p>statistics,0.000010</p> <p>preparing,0.000031</p> <p>executing,116.534678</p> <p>end,0.000015</p> <p>"query end",0.000005</p> <p>"waiting for handler commit",0.182301</p> <p>"closing tables",0.000032</p> <p>"freeing items",0.000627</p> <p>"cleaning up",0.000183</p>

5.2.2. Leistungsoptimierung bei der Kollaborativen Filterung (UC02)

	Abfrageart	Index	Mat	Laufzeit	Bemerkungen
A	Ohne Views, Subsampling < 0.25	No	No	91.625 sec	SQL, direkt auf die Basistabellen.
B	Standardview, Subsampling < 0.25	No	No	17.704 sec	Ausführung der View bei jedem Aufruf
C	Materialized View 1, Subsampling < 0.25	No	Yes	9.922 sec	Zusätzliche Materialized Views für Kombiniertes Rang und Durchschnittl. Bewertung. Dadurch auch Reduktion von JOINS
D	Materialized View 2, Subsampling < 0.25	No	Yes	5.844 sec	Zusätzliche Materialized View mit Filterung der Filme > Bewertung 3.5

	Executionplan	SQL
A		<pre> WITH gewichteter_durchschnitt_berechnung AS (SELECT bb.filme_id, bb.source_id, bb.benutzer_id, AVG(CASE WHEN bb.source_id = 1 AND bb.benutzer_id != 283229 THEN (bb.anzahl_bewertungen * bb.bewertung) / (SELECT COUNT(*) FROM benutzer_bewertungen_subsample WHERE source_id = 1) ELSE bb.bewertung END) AS gewichteter_durchschnitt FROM benutzer_bewertungen_subsample bb WHERE bb.benutzer_id != 283229 AND bb.bewertung >= 3.5 GROUP BY bb.filme_id, bb.source_id, bb.benutzer_id), rank_berechnung AS (SELECT g.filme_id, g.source_id, g.benutzer_id, g.gewichteter_durchschnitt, RANK() OVER (PARTITION BY g.source_id ORDER BY g.gewichteter_durchschnitt DESC) AS quellen_rang FROM gewichteter_durchschnitt_berechnung g), benutzer_die_start_film_bewertet AS (SELECT benutzer_id FROM bewertungen_35_und_hoeher_subs WHERE filme_id = 325), gefilterte_filme AS (SELECT r.filme_id, r.source_id, r.gewichteter_durchschnitt, r.quellen_rang, COUNT(*) AS anzahl_benutzer FROM rank_berechnung r JOIN benutzer_die_start_film_bewertet u ON r.benutzer_id = u.benutzer_id WHERE r.filme_id != 325 GROUP BY r.filme_id, r.source_id, r.gewichteter_durchschnitt, r.quellen_rang HAVING COUNT(*) >= 10) SELECT gefilterte_filme.filme_id, gefilterte_filme.source_id, gefilterte_filme.gewichteter_durchschnitt, gefilterte_filme.quellen_rang FROM gefilterte_filme ORDER BY anzahl_benutzer DESC, quellen_rang DESC LIMIT 20 </pre> <p>20 row(s) returned 91.625 sec / 0.000 sec</p>

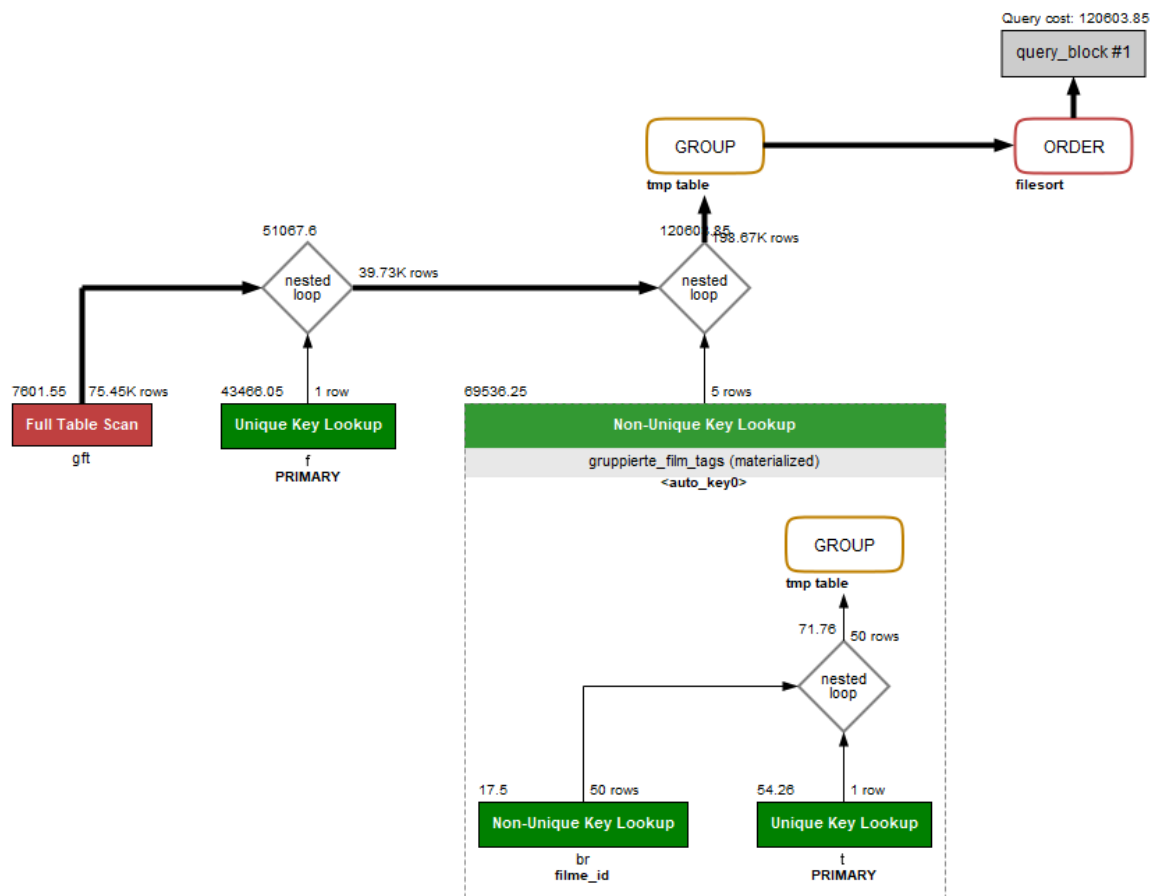
C		<p>WITH benutzer_die_start_film_bewertet AS (SELECT benutzer_id FROM subsampled_mat_view_bewertungen_summe_und_anzahl #VIEW 2 :mat_view_bewertungen_35_und_hoeher_subs # Entferne "_subs" für vollständige Tabelle WHERE filme_id = 325 AND benutzer_bewertung >= 3.5 #(>= 3.5 already in VIEW 2)), gefilterte_filme AS (SELECT mv.filme_id, mv.titel, mv.kombinierter_rang, COUNT(*) AS anzahl_benutzer FROM subsampled_mat_view_bewertungen_summe_und_anzahl mv #mat_view_bewertungen_35_und_hoeher_subs JOIN benutzer_die_start_film_bewertet u ON mv.benutzer_id = u.benutzer_id WHERE mv.filme_id != 325 GROUP BY mv.filme_id, mv.titel, mv.kombinierter_rang HAVING COUNT(*) >= 10) SELECT gefilterte_filme.titel, gefilterte_filme.kombinierter_rang FROM gefilterte_filme ORDER BY anzahl_benutzer DESC, kombinierter_rang DESC LIMIT 20</p> <p>20 row(s) returned 9.922 sec / 0.000 sec</p>
D		<p>WITH benutzer_die_start_film_bewertet AS (SELECT benutzer_id FROM mat_view_bewertungen_35_und_hoeher_subs # Entferne "_subs" für vollständige Tabelle WHERE filme_id = 325 # AND benutzer_bewertung >= 3.5 (already in view)), gefilterte_filme AS (SELECT mv.filme_id, mv.titel, mv.kombinierter_rang, COUNT(*) AS anzahl_benutzer FROM mat_view_bewertungen_35_und_hoeher_subs mv JOIN benutzer_die_start_film_bewertet u ON mv.benutzer_id = u.benutzer_id WHERE mv.filme_id != 325 GROUP BY mv.filme_id, mv.titel, mv.kombinierter_rang HAVING COUNT(*) >= 10) SELECT gefilterte_filme.titel, gefilterte_filme.kombinierter_rang FROM gefilterte_filme ORDER BY anzahl_benutzer DESC, kombinierter_rang DESC LIMIT 20</p> <p>20 row(s) returned 5.844 sec / 0.000 sec</p>

5.2.3. Leistungsoptimierung bei den Empfehlungen über Filmtag (UC03)

	Abfrageart	Index	Mat	Laufzeit	Bemerkungen
A	Ohne Views	No	No	0.625 sec	SQL, direkt auf die Basistabellen.
B	Materialized View, Subsampling < 0.25	No	Yes	0.265 sec	Zusätzliche Materialized View gruppierte_film_tags

Executionplan und SQL	
A	<div><p>Query cost: 288792.49 query_block #1</p><p>Full Table Scan eft (materialized) 8.120000000000001 50 rows</p><p>GROUP tmp table 71.78 50 rows</p><p>nested loop</p><p>Non-Unique Key Lookup fr (materialized) <auto_key1> 70001.14 4.00K rows</p><p>GROUP tmp table 33347.88 288.67K rows</p><p>nested loop</p><p>Unique Key Lookup f PRIMARY 218783.228998998</p><p>Full Table Scan br filme_id 17.5 50 rows</p><p>Unique Key Lookup PRIMARY 54.26 1 row</p><p>Full Table Scan br 44058.45 437.30K rows</p><p>Unique Key Lookup PRIMARY 289417.54000000004</p></div> <p>WITH eingabe_film_tags AS (SELECT br.filme_id, br.tag_id, AVG(br.relevanz_wert) as relevanz FROM benutzer_relevanzen br JOIN tags t ON br.tag_id = t.tag_id WHERE br.filme_id = 299 GROUP BY br.filme_id, br.tag_id), filme_relevanzen AS (SELECT br.filme_id, br.tag_id, AVG(br.relevanz_wert) as relevanz FROM benutzer_relevanzen br JOIN tags t ON br.tag_id = t.tag_id WHERE br.filme_id <> 299 GROUP BY br.filme_id, br.tag_id) SELECT fr.filme_id, f.titel, SUM(ABS(fr.relevanz - eft.relevanz)) AS gesamt_unterschied FROM filme_relevanzen fr JOIN eingabe_film_tags eft ON fr.tag_id = eft.tag_id JOIN filme f ON fr.filme_id = f.filme_id GROUP BY fr.filme_id, f.titel ORDER BY gesamt_unterschied ASC, COUNT(fr.tag_id) DESC LIMIT 10</p> <p>10 row(s) returned 0.625 sec / 0.000 sec</p>

B



```

WITH eingabe_film_tags AS
(SELECT * FROM gruppierte_film_tags
WHERE filme_id = 299 )
SELECT gft.filme_id, f.titel, SUM(ABS(gft.relevanz - eft.relevanz)) AS gesamt_unterschied
FROM mat_view_gruppierte_film_tags gft
JOIN eingabe_film_tags eft
ON gft.tag_id = eft.tag_id
JOIN filme f ON gft.filme_id = f.filme_id
WHERE gft.filme_id <> 299
GROUP BY gft.filme_id, f.titel
ORDER BY gesamt_unterschied ASC, COUNT(gft.tag_id) DESC
LIMIT 10

```

10 row(s) returned 0.265 sec / 0.000 sec

6. Datenschutz & Datenbanksicherheit

- Analyse der Sicherheitsrisiken Ihrer Datenbank
- Verwendete Sicherheitsmassnahmen
- Zusammenhang Massnahmen und Reduktion der Risiken

6.1. Analyse der Sicherheitsrisiken

Informationssicherheit basiert auf drei grundlegenden Prinzipien: Vertraulichkeit, Integrität und Verfügbarkeit. Diese Prinzipien werden auch als CIA-Triade (Confidentiality, Integrity und Availability) bezeichnet. Je nach Anwendung kann eines dieser Prinzipien Vorrang haben. Bei unserer Filmdatenbank hat einerseits das Risiko der Datenveränderungen und andererseits die Verfügbarkeit des Systems hohe Priorität. Somit konzentrieren wir uns auf den Schutz und die Verfügbarkeit der Daten.

<i>Vertraulichkeit</i>		
Risiko	Auswirkungen	Sicherheitsmassnahmen
Unautorisierter Zugriff	Datenveränderung	(A) Benutzer- und Rollenmanagement: Erstellen spezifischer Rollen und Zuweisung angemessener Berechtigungen
		(B) Verwendung von starken Passwörtern
Datenlecks	Unerlaubte Verbreitung der Daten	(C) Verschlüsselung von Datenbanken und Verbindungen
		(D) Nutzung von SSL-Zertifizierung

<i>Integrität</i>		
Risiko	Auswirkungen	Sicherheitsmassnahmen
SQL-Injection	Unerlaubter Zugriff auf Daten, Modifizieren/Löschen von Daten	(E) Serverseite: Integritätsbedingungen. Clientseite: Einsatz BI Tool
Korrupte Daten	Falsche Entscheidungen und Fehlfunktion der Anwendung	(F) Hashing bei der Datenübertragung (G) Datenkonsistenz schützen

<i>Verfügbarkeit</i>		
Risiko	Auswirkungen	Sicherheitsmassnahmen
Datenverlust	Verlust von wertvollen Daten	(H) RAID Festplatten (I) Regelmässige Daten-Backups

6.2. Vertraulichkeit

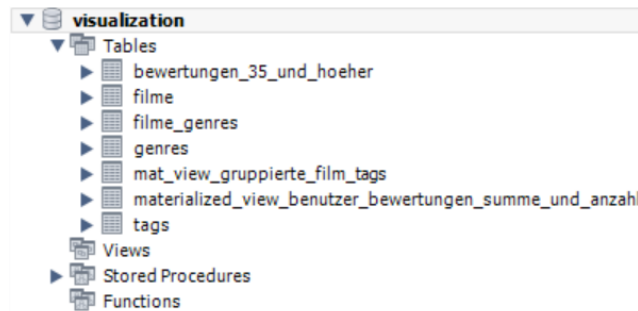
Zur Wahrung der Vertraulichkeit müssen unerlaubte Zugriffe auf unsere Filmdatenbank verhindert werden, beispielsweise wenn ein Hacker sich Zugang zum Netzwerk verschafft und Informationen verändert oder löscht. Da unsere Datenbank nur über wenig sensitive Benutzerdaten verfügt muss primär die Gefahr der Datenveränderung gebannt werden. Die zwei wichtigsten Methoden zur Gewährleistung der Vertraulichkeit sind Verschlüsselung und Zugriffssteuerung.

6.2.1. Zugriffssteuerung

Für unsere Filmdatenbank verwenden wir eine rollenbasierte Zugriffskontrolle. Jeder Mitarbeiter hat eine Rolle, die Ihm den Zugriff auf bestimmte Informationen ermöglicht.

Replikation

Um die Rechte des Benutzers auf Tabellenspalten zu regeln, wurde eine Replikation der filmdatenbank als visualization erstellt. Für diese Replikation wurden nur die für den Endbenutzer wichtigsten Tabellen übernommen:



Benutzer und Rollenmanagement

Rollen Beschreibungen		
A	Endbenutzer	Die 'Endbenutzer Rolle' regelt den Benutzerzugriff für das GUI (den Client): Der Endbenutzer besitzt die Berechtigung, gespeicherte Prozeduren/Views auszuführen (EXECUTE) welche Ihm das gewünschte GUI-Resultat anzeigen (z.B. die Filmempfehlungen). Auch im Use Case, in welchem der Benutzer Filme zur Blacklist hinzufügen kann, wird der Datenzugriff durch die gespeicherte Prozedur geregelt, somit benötigt der Benutzer keine direkte INSERT-Berechtigung für die Tabelle.

Host	User	Scope	Select	Insert	Update	Delete	Create	Drop	Grant	Refer...	Index	Alter	Creat...	Lock...	Creat...	Alter ...	Event	Tr
localhost	Nico	<global>	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
localhost	mysql.infoschema	<global>	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
localhost	root	<global>	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
%	Endbenutzer	visualization	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
localhost	benutzer1	visualization	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

Benutzer-Rollen und Benutzer - SQL-Befehle	
A	<pre>CREATE ROLE 'Endbenutzer'; GRANT SELECT ON visualization.* TO 'Endbenutzer'; GRANT USAGE ON visualization.* TO 'Endbenutzer';</pre>
B	<pre>CREATE USER 'benutzer1'@'localhost' IDENTIFIED BY 'Tn7@fK3UwZ'; GRANT Endbenutzer TO 'benutzer1'@'localhost'; FLUSH PRIVILEGES;</pre>

6.2.2. Verwendung von starken Passwörtern

Damit die Verwendung von starken Passwörtern sichergestellt ist, verwenden wir das folgende PLUGIN:

```
INSTALL COMPONENT 'file://component_validate_password';
SET GLOBAL validate_password.policy = 1;
SHOW VARIABLES LIKE 'validate_password.%';
```

```
# Variable_name, Value
'validate_password.check_user_name', 'ON'
'validate_password.dictionary_file', ''
'validate_password.length', '8'
'validate_password.mixed_case_count', '1'
'validate_password.number_count', '1'
'validate_password.policy', 'MEDIUM'
'validate_password.special_char_count', '1'
```

6.2.3. Verschlüsselung von Datenbanken und Verbindungen

Datenbank verschlüsseln

MySQL erlaubt uns in wenigen Schritten die Datenbank Tabellen zu verschlüsseln:

1. Plugin zum Verschlüsseln von Tabellen installieren:
`INSTALL PLUGIN keyring_file SONAME 'keyring_file.dll';`
2. Tabellen mit dem Befehl `ALTER TABLE ... ENCRYPTION = 'Y'` verschlüsseln.

Beispiel der nicht verschlüsselten Tabelle *bewertungsquellen*. Die Werte „IMDB“ und „Movielens“ lassen sich hier noch klar lesen



Abbildung 9 – unverschlüsselte Tabelle *bewertungsquellen*

Wohingegen auf der verschlüsselten Version keine Wörter mehr erkennbar sind:

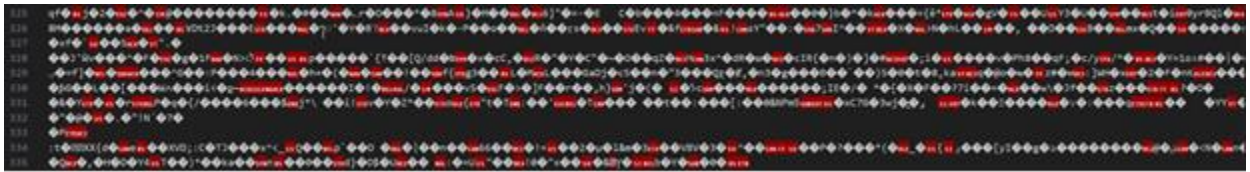
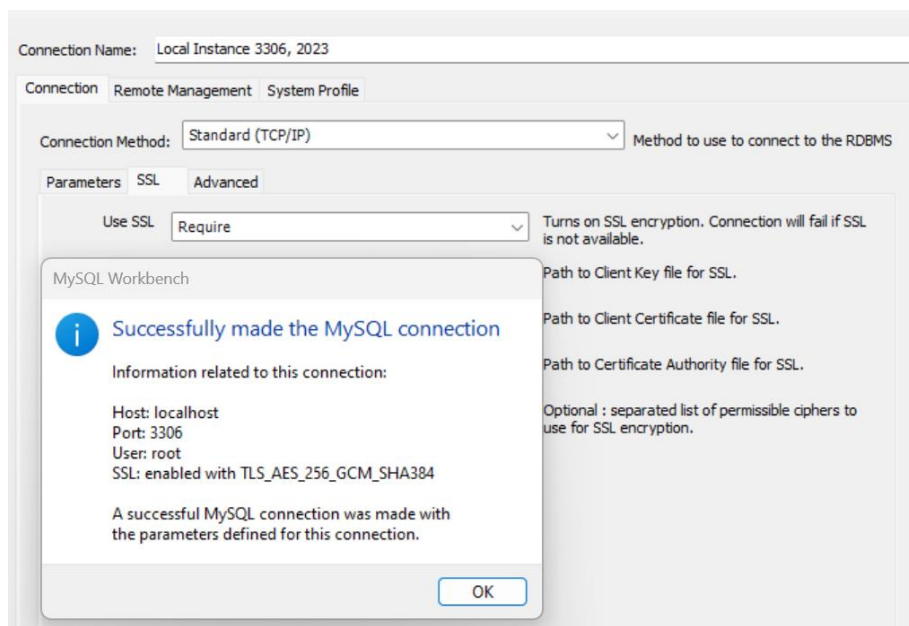


Abbildung 10 – verschlüsselte Tabelle bewertungsquellen

SSL/TLS-Konfiguration



In unserer Datenbankanwendung setzen wir SSL (Secure Sockets Layer) zur Verschlüsselung unserer Verbindungen ein. SSL ist dabei der Vorläufer von TLS (Transport Layer Security). Das bedeutet, dass sämtliche Datenübertragungen zwischen Server und Client verschlüsselt werden und dadurch für Dritte unlesbar sind.

Unser System verwendet hierfür das TLS-Protokoll. Dieses Protokoll bietet Datenschutz und Datenintegrität zwischen zwei kommunizierenden Anwendungen. Es gewährleistet, dass die ausgetauschten Informationen während der Übertragung nicht manipuliert werden können.

Die Verschlüsselungssuite, die wir in unserem System einsetzen, ist AES-256-GCM. Hierbei steht AES für Advanced Encryption Standard und verwendet eine Schlüssellänge von 256 Bit zur Verschlüsselung der Daten. GCM steht für Galois/Counter Mode, der sowohl Vertraulichkeit als auch Authentizität der Datenquelle bietet.

Um die Datenintegrität zu gewährleisten (siehe 7.3.), verwenden wir SHA-384, einen Teil der SHA-2 (Secure Hash Algorithm 2) Familie. SHA-384 ist eine kryptographische Hash-Funktion, die dazu dient zu überprüfen, ob die Daten während der Übertragung unverändert geblieben sind.

Nutzung von SSL-Zertifizierung

Bei symmetrischen Algorithmen müssen der Absender und der Empfänger einer verschlüsselten Nachricht denselben Schlüssel und dieselben Verarbeitungsalgorithmen verwenden. Symmetrische Algorithmen erzeugen einen symmetrischen Schlüssel (auch als „geheimer Schlüssel“ oder „privater Schlüssel“ bezeichnet), der geschützt werden muss.

Unsere Server-Zertifikatsdatei, `server-cert.pem`, ist das digitale Zertifikat, das wir verwenden, um unsere Server-Identität zu bestätigen. Unser Server-Schlüssel, `server-key.pem`, ist der private Schlüssel, den unser Server zur Entschlüsselung der SSL/TLS-Kommunikation verwendet. Es ist von größter Wichtigkeit, diesen Schlüssel sicher zu bewahren, da er das Vertrauen in unsere Server-Identität gewährleistet.

Wir verwenden zur Passwortverschlüsselung den Algorithmus SHA-256. Der private Schlüssel (`private_key.pem`) und der öffentliche Schlüssel (`public_key.pem`) sind dabei Teil des Verschlüsselungsprozesses. Der private Schlüssel wird zur Erstellung einer digitalen Signatur verwendet, während der öffentliche Schlüssel zur Überprüfung dieser Signatur dient.

Der Speicherort der CA-Dateien

Die CA-Dateien (Certificate Authority) sind auf unserem Windows-PC in einem schreibgeschützten Ordner unter folgendem Pfad gespeichert: `C:\ProgramData\MySQL\MySQL Server 8.0\Data`. Dieser Pfad ist nicht für das Netzwerk freigegeben, was zusätzliche Sicherheit bietet. Der Administrator (in diesem Fall wir selbst), das System (unser System für die Anwendung) und der Netzwerkdienst haben Lese-, Schreib-, Änderungs- und Ausführungsrechte. Diese Einstellungen gewährleisten die Integrität und Sicherheit unserer Zertifikatsdateien.

6.3. Integrität

Integrität ist mit den folgenden Zielen verknüpft, die zur Datensicherheit beitragen:

1. Die Vermeidung der Änderung von Informationen durch unbefugte Benutzer
2. Die Vermeidung der unerlaubten oder unbeabsichtigten Änderung von Informationen durch autorisierte Benutzer
3. Die Wahrung interner und externer Konsistenz

Hashing

Mithilfe verschiedener Verschlüsselungsmethoden kann diese Integrität erreicht werden, indem sichergestellt wird, dass eine Nachricht nicht während der Übertragung geändert wurde. Eine solche Änderung könnte dazu führen, dass die Nachricht unlesbar oder gar fehlerhaft ist. Wenn Änderungen im Genre oder der Bewertungen in unserer Filmdatenbank nicht erkannt werden, werden falsche Empfehlungen ermittelt. Wird eine Nachricht manipuliert, sollte das Verschlüsselungssystem darauf hinweisen, dass die Nachricht kompromittiert oder geändert wurde.

Um die Datenintegrität zu gewährleisten (siehe 7.2.2), verwenden wir SHA-384, einen Teil der SHA-2 (Secure Hash Algorithm 2) Familie. SHA-384 ist eine kryptographische Hash-Funktion, die dazu dient zu überprüfen, ob die Daten während der Übertragung unverändert geblieben sind.

Die Wahrung interner und externer Konsistenz

Interne Konsistenz: sorgt dafür, dass Daten intern konsistent sind. In unser Filmdatenbank muss für jede Filmbewertung ein entsprechender Film erfasst und kategorisiert sein. Filme dürfen nicht einfach gelöscht werden können und die entsprechenden Beurteilungen stehenbleiben und 'verweisen'.

Externe Konsistenz: sorgt dafür, dass die in der Datenbank gespeicherten Daten der Realität entsprechen. Ein Film in unserer Datenbank muss existieren und erhältlich sein oder sein Erscheinungsjahr darf nicht vor der Erfindung des Kinofilmes sein.

Nachfolgend Beispiele welche die Konsistenz unserer Filmdatenbank sicherstellen.

<i>Statische Integritätsbedingungen</i>	
A	Die Filmjahre dürfen nicht vor 1880 und nicht in der Zukunft liegen.
	<pre>ALTER TABLE filme ADD CONSTRAINT check_jahr CHECK (jahr >= 1880 AND jahr <= YEAR(CURDATE())), ADD CONSTRAINT check_start_jahr CHECK (jahr >= 1880 AND jahr <= YEAR(CURDATE())), ADD CONSTRAINT check_end_jahr CHECK (jahr >= 1880 AND jahr <= YEAR(CURDATE())), ADD CONSTRAINT relevanz_wert CHECK (relevanz_wert >= 0.00000 and relevanz_wert <= 1.00000);</pre>
B	Die Bewertungen müssen zwischen 0.5 und 5.0 in 0.5 Schritten erfasst sein.
	<pre>ALTER TABLE benutzer_bewertungen ADD CONSTRAINT check_bewertungen CHECK (bewertungen >= 0.5 AND bewertungen <= 5.0 AND MOD(bewertungen * 2, 1) = 0);</pre>

<i>Dynamische Integritätsbedingungen</i>	
A	Prozedurale Integritätsbedingung: Falls der Benutzer sein Konto löscht, wollen wir seine anonymisierten Bewertungen behalten.
	<pre>DELIMITER // CREATE TRIGGER set_geloescht_benutzer BEFORE DELETE ON benutzer FOR EACH ROW BEGIN UPDATE benutzer SET ist_geloescht = 1 WHERE benutzer_id = OLD.benutzer_id; END; // DELIMITER;</pre>

B	Fremdschlüssel Bedingungen: a) Kein Benutzer soll aus dem System gelöscht werden und b) Löschungen erfolgen Top Down. Falls ein Film aus dem System gelöscht wird, werden alle Bewertungen, Genres und Benutzerrelevanzen mitgelöscht.
	a) Für user_id auf tabelle benutzer: ON DELETE: RESTRICT b) Für alle Fremdschlüssel (excl benutzer_ID) in den tabellen benutzer_bewertungen, filme_genres, benutzer_relevanzen, benutzer_blacklist: ON UPDATE und ON DELETE : CASCADE

SQL Injection

Ein SQL-Injection-Angriff ist möglich, wenn eine Website keine ausreichende Eingabebereinigung aufweist. Mit dieser kann vermieden werden, dass Endbenutzer-Eingaben als ausführbarer Code auf den Server gelangen und dort Schäden verursachen. Beim Entwickeln bedeutet das mehr Aufwand, schützt aber letztendlich vor Datenverlust oder unerlaubter Datenmanipulation. SQL Injection ist möglich per Benutzereingabe, durch Cookie-Veränderungen, über Servicevariablen (URL im Browser).

Serverseitig schützen restriktive Integrity Checks die Datenbank. Es können auch Stored Procedure verwendet werden, Dynamic SQL darf nicht verwendet werden oder ein ORM Framework schützt die DB.

Clientseitig ist eine sichere Input Validation oder Character Escaping die Lösung. Unser Client Metabase konvertiert was immer aus der 'Search Box' kommt in einen String und verhindert somit das ausführbarer Code auf die DB gelangt: <https://www.metabase.com/docs/latest/questions/native-editor/sql-parameters>

Datenbank Auditing

Audit-Logs zu Systemereignissen sind immer aktiviert und können nicht deaktiviert werden.

Audit-Logs zu Administratoraktivitäten sind immer aktiviert. Sie können nicht deaktivieren werden.

Audit-Logs zum Datenzugriff sind standardmäßig deaktiviert und werden nur geschrieben, wenn sie explizit aktiviert werden (nur mit Enterprise Version möglich).

Auf der Workbench findet sich unter Help > Locate Logfiles eine Referenz auf den Ordner mit den Aktionslogs. Darin befindet sich unter anderem die Textdatei sql_actions_Local_Instance_3306_2023.txt, welche alle Aktionen (Befehle) auf der MySQL-Verbindung Local_Instance_3306 (für den User, der sich einloggt) trackt:

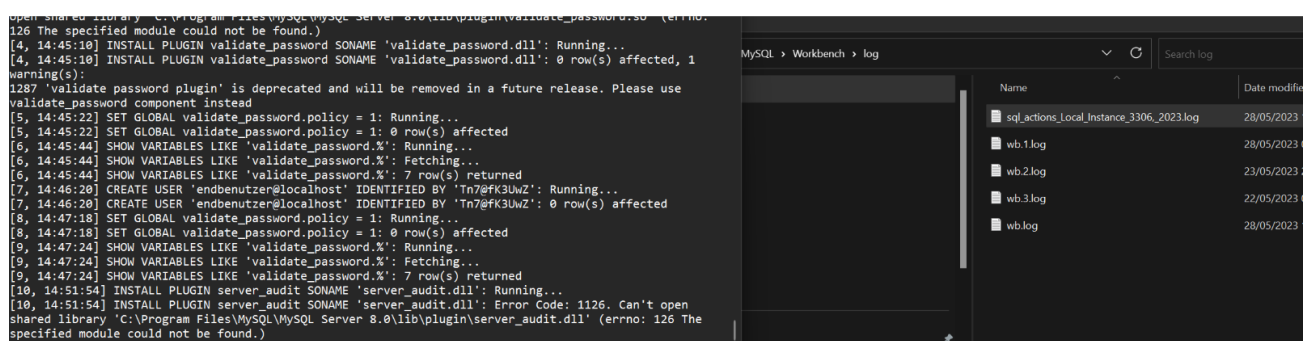


Abbildung 11 – Aktionslog MySQL txt-Datei

6.4. Verfügbarkeit

Verfügbarkeit bedeutet, dass die autorisierten Benutzer eines Systems zeitnah und ohne Unterbrechung auf die Informationen auf dem System und im Netzwerk zugreifen können. Mit einfachen und kostenschonenden Methoden lässt sich die Verfügbarkeit unserer Filmdatenbank sicherstellen:

Redundant Array of Independent Disks (RAID)

RAID ist eine Technologie, die Fehlertoleranz durch den Einsatz mehrerer Festplatten erreicht. Es gibt verschiedene RAID-Ebenen: RAID 0 (Striping), RAID 1 (Mirroring), RAID 3 oder 4 (Striping mit dedizierter Parität), RAID 5 (Striping mit verteilter Parität), RAID 6 (Striping mit doppelter Parität), RAID 1+0 (oder 10) und RAID 0+1.

Backup-Plan

Mit einem Disaster-Recovery-Plan können wir bei einem schwerwiegenden Problem oder Ausfall schnell geeignete Maßnahmen ergreifen. Hierzu gehören beispielsweise Systemausfälle, Netzausfälle, Ausfälle der Infrastruktur und Naturkatastrophen wie Wirbelstürme oder Erdbeben. Ein DR-Plan definiert die Methoden für eine schnellstmögliche Wiederherstellung von Services und den Schutz der Daten vor unzumutbaren Verlusten im Falle einer Katastrophe.

Ein Disaster-Recovery-Plan sollte den Zugriff auf und die Speicherung von Informationen regeln. Für unsere Filmdatenbank ist der Backup-Plan zentral.

Hier sind die Schritte, um eine Sicherung zu erstellen und sie unter Verwendung von MySQL auf einem Windows 11-System zu sichern:

1. Zunächst verwenden Sie den Befehl `mysqldump`, um eine Sicherung zu erstellen:
`mysqldump -u root -p[passwort] --databases filmdatenbank > backup_komplett.sql`
Ersetzen Sie [passwort] und backup_komplett.sql durch Ihr aktuelles MySQL-Passwort und den Namen der Sicherungsdatei. Dieser Befehl erstellt eine vollständige Sicherung Ihrer Datenbank in eine .sql-Datei.
2. Um eine Sicherung nur der Datenbankstruktur zu erstellen, fügen Sie die Option `-d` oder `--no-data` hinzu:
`mysqldump -u root -p[passwort] -d --databases filmdatenbank > backup_struktur.sql`
3. Und um eine Sicherung nur der Daten (ohne Struktur) zu erstellen, fügen Sie die Option `--no-create-info` hinzu:
`mysqldump -u root -p[passwort] --no-create-info --databases filmdatenbank > backup_daten.sql`
4. Um die Sicherungsdatei zu komprimieren, können Sie ein Tool wie 7-Zip verwenden. Sie können die Software von der offiziellen Website herunterladen und installieren.
`7z a backup_komplett.sql.7z backup_komplett.sql`
5. Zum Verschlüsseln der Sicherungsdatei können Sie das gleiche 7-Zip-Tool verwenden:
`7z a -p[Schlüssel] -mhe backup_komplett.sql.7z backup_komplett.sql`
6. Ersetzen Sie [Schlüssel] durch Ihren gewählten Verschlüsselungsschlüssel. Dieser Befehl verschlüsselt die .7z-Datei. Sie müssen sich an den Verschlüsselungsschlüssel erinnern, um die Datei in der Zukunft zu entschlüsseln.

```

E:\00 verschluesselt Filmedatenbankbackup>"C:\Program Files\7-Zip\7z.exe" a "E:\00 verschluesselt Filmedatenbankback
up\backup_daten.sql.7z" "E:\00 verschluesselt Filmedatenbankbackup\backup_daten.sql"

7-Zip 23.00 (x64) : Copyright (c) 1999-2023 Igor Pavlov : 2023-05-07

Scanning the drive:
1 file, 10073889651 bytes (9608 MiB)

Creating archive: E:\00 verschluesselt Filmedatenbankbackup\backup_daten.sql.7z
Add new data to archive: 1 file, 10073889651 bytes (9608 MiB)

6% + backup_daten.sql

```

Abbildung 12 – Komprimierung der Backupdatei via 7-zip

```

E:\00 verschluesselt Filmedatenbankbackup>"C:\Program Files\7-Zip\7z.exe" a -p"S&ecureP@ssw@rd#129" -mhe "E:\00 verschluesselt Filmedatenbankbackup\backup_daten.sql.
7z"

7-Zip 23.00 (x64) : Copyright (c) 1999-2023 Igor Pavlov : 2023-05-07

Open archive: E:\00 verschluesselt Filmedatenbankbackup\backup_daten.sql.7z
--
Path = E:\00 verschluesselt Filmedatenbankbackup\backup_daten.sql.7z
Type = 7z
Physical Size = 891960920
Headers Size = 138
Method = LZMA2:24
Solid = -
Blocks = 1

Scanning the drive:
7 files, 22193560627 bytes (21 GiB)

Updating archive: E:\00 verschluesselt Filmedatenbankbackup\backup_daten.sql.7z
Add new data to archive: 7 files, 22193560627 bytes (21 GiB)

8% U backup_daten.sql

```

Abbildung 13 – Verschlüsselung der Backupdatei via 7-zip

WICHTIG: In unserem Falle sollten die Daten täglich auf einem zusätzlichen externen Server (Dropbox) gesichert werden.

7. Visualisierung & Entscheidungsunterstützung

- Visualisierung der Abfrageergebnisse
- Entscheidungsempfehlungen
- Zusammenhang Visualisierung und ursprünglicher Use Case

Nachdem ich im Unterricht die Möglichkeiten von Metabase (Business Intelligence Tool) entdeckte, bin ich von meiner ursprünglichen Visual-Lösung mit Python zu diesem höchst flexiblen BI Tool gewechselt.

Bei dieser Entscheidung war es wichtig die Vor- und Nachteile eines BI Tools für unsere Lösung abzuwägen:

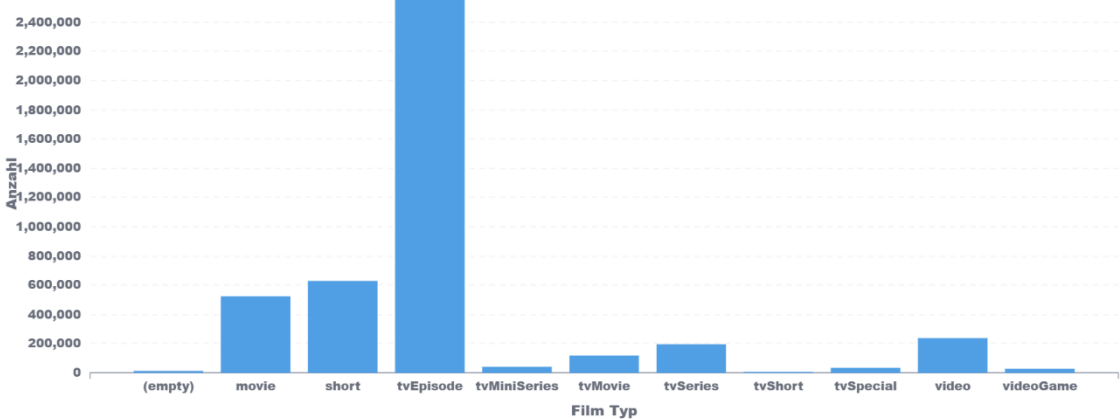
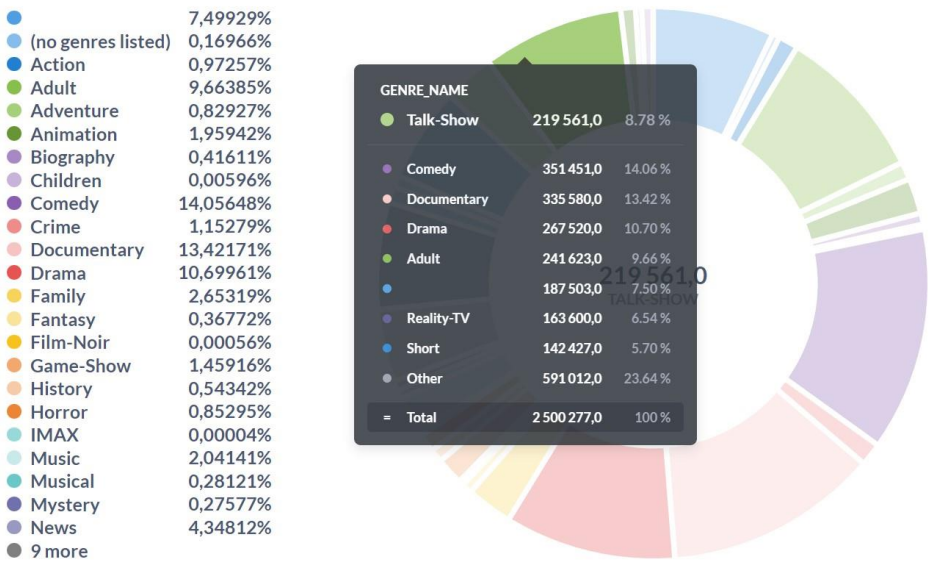
- **Metabase** ist eine gute Wahl für 'straightforward' Data Visualisierung basierend auf einer guten Datenbanklösung (einfache oder mehrere Tabellen auf dem gleichen Server) und unterstützt die Datenvisualisierung schnell und zweckmässig.
- **Python** ist die richtige Lösung bei Data-Streaming und bietet hilfreiche mathematische Bibliotheken, die bei der Datenanalyse helfen. Es können Skripte geschrieben und ausgeführt werden und für die Visualisierung bietet sich die Open-Source-Bibliothek **Plotly** an.
- **Benutzerfreundlichkeit:** Metabase gilt im Allgemeinen als benutzerfreundlicher als Plotly, insbesondere für Benutzer, die noch keine Erfahrung mit der Datenvisualisierung haben. Metabase verfügt über eine benutzerfreundliche Oberfläche und eine breite Palette vorgefertigter Visualisierungstypen, wodurch es sich gut für die schnelle Erstellung grundlegender Dashboards und Berichte eignet. Plotly hingegen ist ein leistungsfähigeres und flexibleres Tool, für dessen effektive Nutzung mehr Programmierkenntnisse erforderlich sind.
- **Anpassbarkeit:** Plotly ist im Allgemeinen anpassbarer als Metabase, da es Entwicklern ermöglicht, mithilfe der Plotly.js-Bibliothek benutzerdefinierte Visualisierungen und Interaktivität zu erstellen. Metabase ermöglicht zwar einige Anpassungen, ist in dieser Hinsicht jedoch eingeschränkter als Plotly.

Ich entscheide mich für Metabase weil mich die Einfachheit des BI Tools überzeugt und wir keine speziellen, komplexen und benutzerdefinierten Visualisierungen umsetzen werden.

Metabase ist ein Open-Source-BI-Server, den wir einfach installiert haben und verschiedene Datenquellen wie unsere MySQL Datenbank lassen sich danach wirklich einfach integrieren. Die Benutzer des Tools benötigen keine umfangreichen technischen Kenntnisse (oder Programmierkenntnisse) und können in einer sicheren Systemumgebung mit einer intuitiven Benutzeroberfläche:

- a) nützliche Erkenntnisse aus der Filmdatenbank ziehen
- b) die Film-Empfehlungen aus unseren Use Cases in Form von Diagrammen und Dashboards einfach erstellen und danach an Freunde weitergeben.
- c) zusätzliche Filterkriterien erfassen und die Film-Empfehlung weiter einschränken, je nach Vorlieben der Person.

7.1. Analyse Filmdatenbank

Analyse Filmdatenbank																																																																										
A	Dass die Bedeutung von TV-Serien bei IDMB stark zugenommen hat, zeigt die folgende Übersicht der vorhandenen 'Film Typen' in unserer Filme-Datenbank deutlich.																																																																									
																																																																										
B	Eine Übersicht aller 'Film Genres' gibt uns einen schnellen Überblick über die Aufteilung nach Genre der Filmdatenbank und hilft auch den Benutzern bei weiteren Filterungen.																																																																									
	 <table><tr><td>●</td><td></td><td>7,49929%</td></tr><tr><td>●</td><td>(no genres listed)</td><td>0,16966%</td></tr><tr><td>●</td><td>Action</td><td>0,97257%</td></tr><tr><td>●</td><td>Adult</td><td>9,66385%</td></tr><tr><td>●</td><td>Adventure</td><td>0,82927%</td></tr><tr><td>●</td><td>Animation</td><td>1,95942%</td></tr><tr><td>●</td><td>Biography</td><td>0,41611%</td></tr><tr><td>●</td><td>Children</td><td>0,00596%</td></tr><tr><td>●</td><td>Comedy</td><td>14,05648%</td></tr><tr><td>●</td><td>Crime</td><td>1,15279%</td></tr><tr><td>●</td><td>Documentary</td><td>13,42171%</td></tr><tr><td>●</td><td>Drama</td><td>10,69961%</td></tr><tr><td>●</td><td>Family</td><td>2,65319%</td></tr><tr><td>●</td><td>Fantasy</td><td>0,36772%</td></tr><tr><td>●</td><td>Film-Noir</td><td>0,00056%</td></tr><tr><td>●</td><td>Game-Show</td><td>1,45916%</td></tr><tr><td>●</td><td>History</td><td>0,54342%</td></tr><tr><td>●</td><td>Horror</td><td>0,85295%</td></tr><tr><td>●</td><td>IMAX</td><td>0,00004%</td></tr><tr><td>●</td><td>Music</td><td>2,04141%</td></tr><tr><td>●</td><td>Musical</td><td>0,28121%</td></tr><tr><td>●</td><td>Mystery</td><td>0,27577%</td></tr><tr><td>●</td><td>News</td><td>4,34812%</td></tr><tr><td>●</td><td>9 more</td><td></td></tr></table>		●		7,49929%	●	(no genres listed)	0,16966%	●	Action	0,97257%	●	Adult	9,66385%	●	Adventure	0,82927%	●	Animation	1,95942%	●	Biography	0,41611%	●	Children	0,00596%	●	Comedy	14,05648%	●	Crime	1,15279%	●	Documentary	13,42171%	●	Drama	10,69961%	●	Family	2,65319%	●	Fantasy	0,36772%	●	Film-Noir	0,00056%	●	Game-Show	1,45916%	●	History	0,54342%	●	Horror	0,85295%	●	IMAX	0,00004%	●	Music	2,04141%	●	Musical	0,28121%	●	Mystery	0,27577%	●	News	4,34812%	●	9 more	
●		7,49929%																																																																								
●	(no genres listed)	0,16966%																																																																								
●	Action	0,97257%																																																																								
●	Adult	9,66385%																																																																								
●	Adventure	0,82927%																																																																								
●	Animation	1,95942%																																																																								
●	Biography	0,41611%																																																																								
●	Children	0,00596%																																																																								
●	Comedy	14,05648%																																																																								
●	Crime	1,15279%																																																																								
●	Documentary	13,42171%																																																																								
●	Drama	10,69961%																																																																								
●	Family	2,65319%																																																																								
●	Fantasy	0,36772%																																																																								
●	Film-Noir	0,00056%																																																																								
●	Game-Show	1,45916%																																																																								
●	History	0,54342%																																																																								
●	Horror	0,85295%																																																																								
●	IMAX	0,00004%																																																																								
●	Music	2,04141%																																																																								
●	Musical	0,28121%																																																																								
●	Mystery	0,27577%																																																																								
●	News	4,34812%																																																																								
●	9 more																																																																									
C	Aber auch einfache Listabfragen können über das BI Tool vorgenommen werden und ermitteln blitzschnell die (nicht unerwarteten) Gewinner z.B. nach 'Anzahl Bewertungen':																																																																									
	<table><tr><th>Titel</th><th>Gesamtbenutzerbewertungen</th><th>Anzahl Bewertungen</th></tr><tr><td>The Silence of the Lambs</td><td>1.133.857,2</td><td>87.899</td></tr><tr><td>The Matrix</td><td>1.302.831,99</td><td>84.545</td></tr><tr><td>Star Wars: Episode IV - A New Hope</td><td>1.540.054,1</td><td>81.815</td></tr><tr><td>Jurassic Park</td><td>2.251.632</td><td>76.451</td></tr></table>		Titel	Gesamtbenutzerbewertungen	Anzahl Bewertungen	The Silence of the Lambs	1.133.857,2	87.899	The Matrix	1.302.831,99	84.545	Star Wars: Episode IV - A New Hope	1.540.054,1	81.815	Jurassic Park	2.251.632	76.451																																																									
Titel	Gesamtbenutzerbewertungen	Anzahl Bewertungen																																																																								
The Silence of the Lambs	1.133.857,2	87.899																																																																								
The Matrix	1.302.831,99	84.545																																																																								
Star Wars: Episode IV - A New Hope	1.540.054,1	81.815																																																																								
Jurassic Park	2.251.632	76.451																																																																								

Dazu können 'vorgefertigte' Abfragen zur Verfügung gestellt werden oder mit wenig SQL-Wissen kann der Benutzer schnell weitere Abfragen selbst erstellen oder die bestehenden Abfragen anpassen.

Analyse Filmdatenbank – Einfache SQL Abfragen	
A	Abfrage Filmtyp
	<pre># Film Typ SELECT titeltyp, count(*) FROM filmdatenbank8.filme group by titel_typ;</pre>
B	Abfrage Filmgenres
	<pre># Genres SELECT genre_name,count(*) FROM filme JOIN filme_genres ON filme.film_id = filme_genres.film_id JOIN genres ON filme_genres.genre_id = genres.genre_id group by genres.genre_name;</pre>
C	Abfrage Top 4
	<pre># top 4 nach Kombierter Rang und Gesamtanzahl bewertungen SELECT DISTINCT filme.titel, kombierter_rang, anzahl_bewertungen FROM `materialized_view_benutzer_bewertungen_summe_und_anzahl` JOIN filme ON `materialized_view_benutzer_bewertungen_summe_und_anzahl`.film_id = filme.film_id group by filme.titel, kombierter_rang, anzahl_bewertungen order by anzahl_bewertungen DESC, kombierter_rang DESC LIMIT 4;</pre>

7.2. Suche Ausgangsfilm (UC01)

In jede Metabase Abfrage kann ein **Metabase-Feldfilter** verwendet werden, welcher dem Benutzer flexibel erlaubt innerhalb jeder Abfrage den Ausgangsfilm zu suchen und auszuwählen.

Das lässt sich einfach und flexibel mit den folgenden Schritten erledigen:

1. Zuerst müssen Sie eine Abfrage erstellen, die sich mit Ihrer Datenbank verbindet.
2. Wählen Sie die Option "Native Query" (SQL).
3. Im geöffneten SQL-Editor schreiben Sie Ihre SQL-Abfrage. Hier müssen Sie eine Variable mit der spezifischen Variablensyntax von Metabase angeben. Wenn Sie zum Beispiel eine SQL-Abfrage wie `SELECT * FROM filme WHERE filme_id = ?` verwenden, ersetzen Sie das Fragezeichen durch `{{film}}` wie folgt: `SELECT * FROM filme WHERE filme_id = {{film}}`
4. Jetzt ist `{{film}}` eine Variable, die von Metabase erkannt wird. Nachdem Sie die Abfrage geschrieben haben, öffnet sich eine Seitenleiste auf der rechten Seite, in der Sie den "Variablentyp" festlegen können.
5. Sie müssen den "Variablentyp" auf "Feldfilter" setzen und ihn mit dem entsprechenden Feld in Ihrer Datenbank verknüpfen, in diesem Fall dem Feld "filme_id" in der Tabelle "filme".
6. Danach stellt Metabase jedes Mal, wenn Sie diese Frage ausführen, ein benutzerfreundliches Widget bereit, mit dem Sie einen Film aus einer Dropdown-Liste, einer Suchbox oder einem anderen geeigneten Widget je nach Datentyp auswählen können.
7. Der ausgewählte Wert im Widget wird als "film_id" an die SQL-Abfrage übergeben und die Abfrage wird entsprechend ausgeführt.

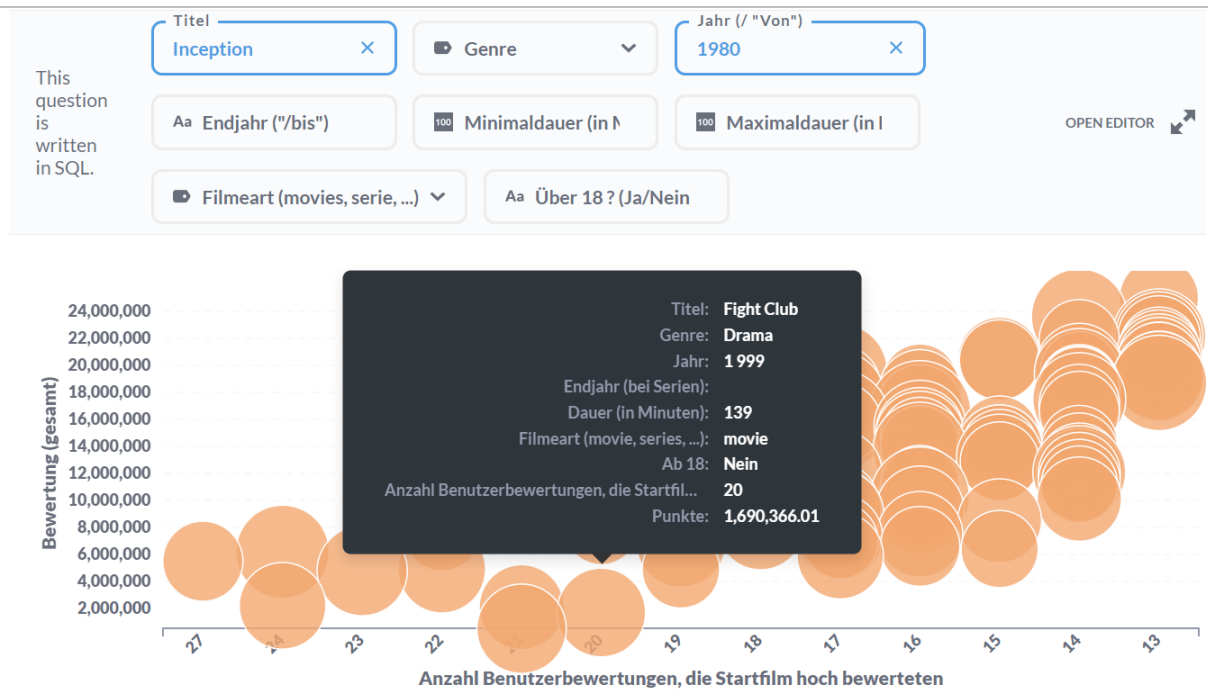
Diese einfache Feldfilter Funktion wird zur Bestimmung des Ausgangsfilms verwendet oder kann für weitere Filterfunktionen in den Benutzerabfragen verwendet werden (z.B. Genre).

7.3. Empfehlungen über ähnliche Benutzer (UC02 / UC04)

Kollaborative Filterung

A Kollaborative Filterung: Die Hauptidee besteht darin, Filme zu finden, die von Benutzern geschätzt werden, welche den Ausgangsfilm **'Inception'** geschätzt haben. Somit bestimmen wir zunächst alle Benutzer, welche 'Inception' höher als 3.5 bewertet haben und danach filtern wir die Filme welche von diesen Benutzern ebenfalls > 3.5 bewertet wurden.

Nolan ist ein '80er Jahre Filmfan' und schliesst bei seiner Anfrage sofort alle älteren Filme aus (ab Jahr 1980). Die Auswahl erfolgt über eine Pop-Up Anzeige, welche erscheint, wenn Nolan mit der Maus über die 'Film-Bubbles' fährt. Mittels dieser graphischen Anzeige entdeckt Nolan auf spielerische Art seine Lieblingsfilme und erhält mit jeder Pop-Up Anzeige zusätzliche Filminformationen.



B Nachdem sich Nolan die Filme mehrmals angeschaut hat, fällt seine Entscheidung auf die Filmempfehlung 'Fight Club'. Nolan sieht den zweiten Filter 'Genre' und startet eine zweite Suche bei der er mit dem Film-Genre 'Drama' (übernommen von 'Fight Club') die Anfrage nochmals stärker einschränkt.

	<div data-bbox="268 224 1497 347"> <p>This question is written in SQL.</p> <div> <div> <div>Titel</div> <div>Inception</div> <div>×</div> </div> <div> <div>Genre</div> <div>Drama</div> <div>×</div> </div> <div> <div>Jahr (/ "Von")</div> <div>1980</div> <div>×</div> </div> <div> <div>Aa</div> <div>Endjahr ("%bis")</div> </div> </div> <div> <div> <div>Minimaldauer (in)</div> <div>Maximaldauer (in)</div> <div>Filmeart (movies, serie, ...)</div> <div>Aa Über 18? (Ja/Nein)</div> </div> <div>OPEN EDITOR</div> </div> </div> <div data-bbox="268 358 1497 761"> <div data-bbox="379 421 810 645"> <p> Titel: American Beauty Genre: Drama Jahr: 1999 Endjahr (bei Serien): Dauer (in Minuten): 122 Filmeart (movie, series, ...): movie Ab 18: Nein Anzahl Benutzerbewertungen, die Startfil...: 21 Punkte: 2.118,594.63 </p> </div> </div>
	<p>Nolan fährt mit der Maus durch die 'vorderen Empfehlungen': Bei dieser zweiten Film-Empfehlung rückt der Film 'American Beauty' mehr in den Vordergrund – ein weiterer passender Vorschlag für Nolan. Dennoch entscheidet er für den Film 'Fight Club' und freut sich auf den bevorstehenden spannenden Filmabend.</p>

7.4. Empfehlungen über Film-Tag (UC03)

Empfehlung über Film-Tags	
	<p>Inhaltliche Filterung: Die Hauptidee besteht darin, Filme zu finden welche eine hohe Übereinstimmung mit den Stichwörtern (Tags) im Ausgangsfile aufweisen.</p> <p>Die Vorschläge werden aufsteigend nach der Differenz der Tag-Relevanz angezeigt. Dies ist eine Metrik, die den Unterschied in der Relevanz zwischen den Tags des Eingabefilms und den Tags aller anderen Filme misst: Je kleiner die gesamte Differenz, desto ähnlicher sind die Filme in Bezug auf ihre Tag-Verteilung.</p> <p>Dazu verwenden wir emotionale Stichwörter (Filmtags) welche den Film beschreiben: z.B. realistisch, gedankenanregend, humorvoll. Basierend auf solch emotional gesetzten Bewertungen, werde die Empfehlungen auch überraschender ausfallen als beim kollaborativen Filtern.</p> <p>Der 'Comic-Fan' Robin startet seine Filmsuche wie folgt:</p> <ol style="list-style-type: none"> Zunächst lässt er sich die Filmtags im Ausgangsfilem 'Batman Forever' anzeigen. Robin kann nun die Gewichtung aller vorgeschlagenen Filmtags via GUI justieren, z.B. weniger realistisch aber mehr humorvoll. Robin erhält drei Film Empfehlungen.
A	<p>Unser 'Comic und Batman-Fan' Robin schaut sich zunächst die vorhandenen Film-Tags im Ausgangsfile an. Robin bestimmt drei Tags, welche die Empfehlungen steuern werden. Da es sich bei den Tags mehr um eine emotionale Bewertung des Films handelt, wird Robin nicht immer die erwarteten Tags und Gewichtungen finden. So würde Robin nicht unbedingt den Tag 'overrated' beim Film 'Batman Forever' erwarten. Die Tags 'drama', 'emotional' und 'beautiful' passen jedoch für Ihn und auch 'friendship' klingt für jeder Comic-Fan interessant.</p>

B	<p>Robin wählt die folgenden drei Tags für seine erste Suche und übernimmt die vorgeschlagenen Relevanz-Werte für die Suche.</p>
C	<p>Metabase bietet viele Möglichkeiten die Diagramme anzupassen und Robin kann dabei aus einer breiten Palette vorgefertigter Visualisierungstypen auswählen. Er belässt das vorgegebene Balkendiagramm und erhält schnell ein erstes Diagramm mit drei Vorschlägen.</p>

D

Diese Filmempfehlungen bleiben in der Nähe der Tags des Ausgangsfilms. Die letzte Empfehlung ist für Robin jedoch zu 'beautiful' und er ersetzt das Tag mit dem Comic-freundlicheren 'friendship' für die nächste Suche. Die Gewichtung übernimmt er vom Ausgangsfilm.

Tag1 name

emotional

×

Tag1 weight

0.17

×

Tag2 name

drama

×

Tag2 weight

0.034

×

Tag3 name

friendship

×

Tag3 weight

0.1

×

Input film

Batman Forever

×

E

Robin ändert das Metabase Diagramm mit einigen Klicks einfach und schnell an (ohne Programmiererfahrung): breitere Säulen, %-Angaben und beim Klicken auf 'friendship' wird die grüne Säule im Vordergrund angezeigt.

mystery

love

friendship

drama

insanity

soundtrack

drugs

action

good acting

true story

22.6%

10.3%

6.68%

1.95%

5.95%

4.5%

2.88%

7.6%

6.58%

5.92%

7.05%

Gewicht

Tron

Hercule & Sherlock

Minotaur

Film und Tags

F

'Tron' überzeugt unseren 'Comic-Fan' Robin aber auch 'Minotaur' ist eine sehr gute Empfehlung. 'Hercule & Sherlock' ist Robin dann doch zu 'emotional'. Für die nächste Suche erhöht Robin die 'friendship' Bewertung von 0.1 auf 0.7.

	<div><div><div><div><div><div></div><div>mystery</div></div><div><div></div><div>love</div></div><div><div></div><div>friendship</div></div><div><div></div><div>drama</div></div><div><div></div><div>comedy</div></div><div><div></div><div>Funny</div></div><div><div></div><div>Nudity (Topless)</div></div><div><div></div><div>Family</div></div><div><div></div><div>true story</div></div></div></div><div><table><tr><th>Film</th><th>Tag</th><th>Gewicht</th></tr><tr><td rowspan="4">Tron</td><td>mystery</td><td>22.9%</td></tr><tr><td>friendship</td><td>10.3%</td></tr><tr><td>drama</td><td>6.83%</td></tr><tr><td>love</td><td>1.95%</td></tr><tr><td rowspan="6">The Eclipse</td><td>friendship</td><td>13.55%</td></tr><tr><td>drama</td><td>7.23%</td></tr><tr><td>comedy</td><td>3.05%</td></tr><tr><td>Funny</td><td>3.27%</td></tr><tr><td>Nudity (Topless)</td><td>8.15%</td></tr><tr><td>Family</td><td>8.15%</td></tr><tr><td rowspan="2">Minotaur</td><td>friendship</td><td>6.58%</td></tr><tr><td>drama</td><td>5.32%</td></tr><tr><td></td><td>true story</td><td>7.05%</td></tr></table></div></div></div>	Film	Tag	Gewicht	Tron	mystery	22.9%	friendship	10.3%	drama	6.83%	love	1.95%	The Eclipse	friendship	13.55%	drama	7.23%	comedy	3.05%	Funny	3.27%	Nudity (Topless)	8.15%	Family	8.15%	Minotaur	friendship	6.58%	drama	5.32%		true story	7.05%
Film	Tag	Gewicht																																
Tron	mystery	22.9%																																
	friendship	10.3%																																
	drama	6.83%																																
	love	1.95%																																
The Eclipse	friendship	13.55%																																
	drama	7.23%																																
	comedy	3.05%																																
	Funny	3.27%																																
	Nudity (Topless)	8.15%																																
	Family	8.15%																																
Minotaur	friendship	6.58%																																
	drama	5.32%																																
	true story	7.05%																																
G	<p>Sehr gut - 'Hercule & Sherlock' ist draussen und mit der sehr spannenden Empfehlung 'The Eclipse', sind nun die Vorschläge perfekt für Robin. Dennoch ein weiterer Versuch: Robin ersetzt 'drama' mit 'inspirational' und setzt die Gewichtung hoch auf 0.9.</p>																																	
	<div><div><div>Tag1 name <input type="text" value="emotional"/> <input type="button" value="x"/></div><div>Tag1 weight <input type="text" value="0.17"/> <input type="button" value="x"/></div><div>Tag2 name <input type="text" value="inspirational"/> <input type="button" value="x"/></div><div>Tag2 weight <input type="text" value="0.9"/> <input type="button" value="x"/></div><div>Tag3 name <input type="text" value="friendship"/> <input type="button" value="x"/></div><div>Tag3 weight <input type="text" value="0.7"/> <input type="button" value="x"/></div><div>Input film <input type="text" value="Batman Forever"/> <input type="button" value="x"/></div></div></div>																																	
H	<div><div><div><div><div><div></div><div>inspirational</div></div><div><div></div><div>boring</div></div><div><div></div><div>brilliant</div></div><div><div></div><div>high school</div></div><div><div></div><div>school drama</div></div><div><div></div><div>friendship</div></div><div><div></div><div>great acting</div></div><div><div></div><div>thought-provoking</div></div><div><div></div><div>documentary</div></div><div><div></div><div>car accident</div></div><div><div></div><div>sexuality</div></div><div><div></div><div>feel-good</div></div><div><div></div><div>emotional</div></div><div><div></div><div>true story</div></div><div><div></div><div>racism</div></div><div><div></div><div>based on a true story</div></div><div><div></div><div>Disney</div></div></div></div><div><table><tr><th>Film</th><th>Tag</th><th>Gewicht</th></tr><tr><td rowspan="5">King of the Mountain</td><td>inspirational</td><td>8.5%</td></tr><tr><td>boring</td><td>6.5%</td></tr><tr><td>brilliant</td><td>8.5%</td></tr><tr><td>high school</td><td>3.5%</td></tr><tr><td>school drama</td><td>7.5%</td></tr><tr><td rowspan="2">Coraline</td><td>friendship</td><td>5.5%</td></tr><tr><td>great acting</td><td>39.5%</td></tr><tr><td rowspan="5">All the Young Men</td><td>friendship</td><td>5.5%</td></tr><tr><td>great acting</td><td>31.5%</td></tr><tr><td>high school</td><td>4.5%</td></tr><tr><td>school drama</td><td>8.5%</td></tr><tr><td>emotional</td><td>3.5%</td></tr></table></div></div></div>	Film	Tag	Gewicht	King of the Mountain	inspirational	8.5%	boring	6.5%	brilliant	8.5%	high school	3.5%	school drama	7.5%	Coraline	friendship	5.5%	great acting	39.5%	All the Young Men	friendship	5.5%	great acting	31.5%	high school	4.5%	school drama	8.5%	emotional	3.5%			
Film	Tag	Gewicht																																
King of the Mountain	inspirational	8.5%																																
	boring	6.5%																																
	brilliant	8.5%																																
	high school	3.5%																																
	school drama	7.5%																																
Coraline	friendship	5.5%																																
	great acting	39.5%																																
All the Young Men	friendship	5.5%																																
	great acting	31.5%																																
	high school	4.5%																																
	school drama	8.5%																																
	emotional	3.5%																																
	<p>Wie erwartet erhält Robin mehr 'inspirational' Filmempfehlungen. Robin ist von 'Coraline' begeistert aber auch 'King of the Mountain' weckt seine Interesse. Der letzte Vorschlag ist Ihm zu '60s'. Aber mit insgesamt fünf spannenden Empfehlungen ist Robin zufrieden, beendet die Filmsuche und freut sich auf den 'Tron' Filmabend mit seinen Freunden.</p>																																	

8. Fazit & Lessons Learned

Das Hauptziel dieser Arbeit ist für mich erreicht: Ein tieferes praktisches Verständnis zu erlangen, sowohl für den Aufbau von Informationssystemen, für die Empfehlungsmethodik als auch für die Datenbanktechnik.

Was habe dazu gelernt?

- Ich bin mit nur spärlichen Datenbankerfahrung gestartet und habe im Laufe der Arbeit den Umgang mit der Datenbanktechnik praktisch erfahren/erlernt.
- Die Arbeit lernte mich, wie ein Use Case/Requirement/Entwurf Schritt für Schritt bis zur Implementierung verfolgt wird und wie ein Informationssystem praktisch aufgebaut wird.
- Was es heisst mit grossen Datenmengen umzugehen – Daten bereitstellen – Transformieren – Daten laden – Performance! – vielleicht ein guter Vorgeschmack auf Big Data.
- Trotz AI und Machine Learning hat SQL mit der Zeit mitgehalten und bleibt weiterhin eine mächtige Sprache für die Kommunikation mit Datenbanken, um die darin enthaltenen Daten zu verwalten.

Wo bin ich meinen Zielen nähergekommen, wo nicht?

- Viele der ursprünglichen Use Cases wurden während dieser Arbeit umgesetzt, aber nicht alle. Zum Beispiel fehlt der Schauspielernamen oder auch die Verbindung zur IMDB-Datenbank.
- Die beiden Datenquellen konnten integriert werden. Jedoch der Aufwand für die Datenbereinigung, der Abgleich der Daten (z.B. die Abstimmung der Namen) und die Migration der Daten in die gemeinsame Datenbank war hoch und am Ende rettete mich nur noch meine Python-Erfahrung, da sich komplexe Textbereinigungen nur schwierig mit SQL lösen lassen.
- Dadurch war es mir nicht möglich innerhalb der Projektzeit weitere Filmdatenbanken zu importieren, was notwendig ist, um die Empfehlungen weiter zu verbessern (Arthouse, Film Noire, 80ies). Auch weitere Filmattribute wie zum Beispiel Schauspieler und Regisseur wären hilfreich gewesen.
- Gerade während der Analyse-Phase wäre ein BI Tool wie Metabase sehr hilfreich gewesen. Ein gutes Datenanalyse-Tool würde ich bei einem nächsten DB-Projekt von Beginn weg einsetzen. Und je früher das Tool verwendet wird, je besser kann man es dann auch für die Visualisierung einsetzen.
- Als Python Entwickler, würde ich die Arbeit gerne auf eine NoSQL Lösung migrieren, um die beiden Ansätze zu vergleichen.

Wie habe ich mich dabei gefühlt?

- Das Durchsetzen einer einfachen Ausgangsidee bis zur komplexen Datenbankimplementierung verlangt eine permanente Fokussierung. Da kommt es schon mal vor, dass man sich tief im SQL-Syntax/Strudel befindet und die eigentliche Aufgabe zwischendurch verloren geht ... Da fühlt man sich alleine – und da ist das Zurückfinden in einem Team einfacher.
- Die gewohnten Entwicklungsumgebungen der objektorientierten Welt haben mir schmerzlich gefehlt und bei (zu) komplexen SQL-Statements dauerte die Fehlersuche zu lange (ewig).
- Das erreichte Resultat gibt mir ein gutes Gefühl, alleine die Klippen beim Aufbau eines Informationssystems bewältigt zu haben. Lässt mich aber auch erahnen, wie hoch die Komplexität bei der Weiterentwicklung und Verbesserung eines bestehenden Empfehlungssystem sein muss.

9. Anhänge

Anhang A) ‚DDL Filmdatenbank‘

```
DROP DATABASE IF EXISTS filmdatenbank;
CREATE DATABASE filmdatenbank;
USE filmdatenbank;

-- Tabelle für Filme erstellen
CREATE TABLE filme (
    filme_id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    movielens_id INT UNSIGNED UNIQUE,
    imdb_id INT UNSIGNED UNIQUE,
    tmdb_id INT UNSIGNED UNIQUE,
    imdb_id_tt VARCHAR(10) UNIQUE,
    titel VARCHAR(200) NOT NULL,
    originaltitel VARCHAR(200),
    jahr YEAR,
    ist_erwachsen BOOLEAN,
    start_jahr YEAR,
    end_jahr YEAR,
    dauer_minuten INT UNSIGNED,
    titel_typ VARCHAR(20)
);

-- Tabelle für Genres erstellen
CREATE TABLE genres (
    genre_id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    genre_name VARCHAR(50) NOT NULL UNIQUE
);

-- Tabelle für Filme-Genres-Verknüpfung erstellen
CREATE TABLE filme_genres (
    filme_id INT UNSIGNED,
    genre_id INT UNSIGNED,
    PRIMARY KEY (filme_id, genre_id),
    FOREIGN KEY (filme_id) REFERENCES filme(filme_id),
    FOREIGN KEY (genre_id) REFERENCES genres(genre_id)
);

-- Tabelle für Benutzer erstellen
CREATE TABLE benutzer (
    benutzer_id INT UNSIGNED PRIMARY KEY
);

CREATE TABLE benutzer_blacklist (
    list_id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    benutzer_id INT UNSIGNED NOT NULL,
    filme_id INT UNSIGNED NOT NULL,
    FOREIGN KEY (benutzer_id) REFERENCES benutzer(benutzer_id),
    FOREIGN KEY (filme_id) REFERENCES filme(filme_id)
```

```

);

-- Tabelle für Bewertungsquellen erstellen
CREATE TABLE bewertungsquellen (
    source_id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    source_name VARCHAR(50) NOT NULL UNIQUE
);

-- Tabelle für Bewertungen erstellen

CREATE TABLE benutzer_bewertungen (
    bewertungs_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    filme_id INT UNSIGNED NOT NULL,
    benutzer_id INT UNSIGNED NOT NULL,
    source_id INT UNSIGNED NOT NULL,
    durchschnittsbewertung FLOAT NOT NULL,
    anzahl_bewertungen INT UNSIGNED,
    zeitstempel DATETIME NOT NULL,
    FOREIGN KEY (filme_id) REFERENCES filme(filme_id),
    FOREIGN KEY (source_id) REFERENCES bewertungsquellen(source_id)
);

-- Tabelle für Tagnamen erstellen
CREATE TABLE tags (
    tag_id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    tag_name VARCHAR(70) NOT NULL UNIQUE
);

-- Tabelle für Benutzer-Tags und Tag-Relevanz erstellen
CREATE TABLE benutzer_relevanzen (
    relevanz_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    benutzer_id INT UNSIGNED NOT NULL,
    filme_id INT UNSIGNED NOT NULL,
    tag_id INT UNSIGNED NOT NULL,
    relevanz_wert FLOAT NOT NULL,
    zeitstempel DATETIME NOT NULL,
    FOREIGN KEY (benutzer_id) REFERENCES benutzer(benutzer_id),
    FOREIGN KEY (filme_id) REFERENCES filme(filme_id),
    FOREIGN KEY (tag_id) REFERENCES tags(tag_id));

```

Anhang B) Weitere Datenbereinigungen und -transformationen

Bereinigung der Filmtitel mittels Regular Expressions

```
-- Update MovieLens temp_filme

ALTER TABLE temp_filme ADD COLUMN normalized_titel VARCHAR(255);
UPDATE temp_filme
SET normalized_titel = REGEXP_REPLACE(
    TRIM(
        CASE
            WHEN SUBSTRING_INDEX(titel, ' ', -1) IN ('the', 'The', 'an', 'An', 'a', 'A')
            THEN CONCAT(SUBSTRING_INDEX(titel, ' ', -1), ' ', SUBSTRING_INDEX(titel, ' ', 1))
            ELSE titel
        END
    ),
    '[:.*]',
    ''
);

-- Update temp_imdb_filme
ALTER TABLE temp_imdb_filme ADD COLUMN normalized_titel VARCHAR(255);
UPDATE temp_imdb_filme
SET normalized_titel = REGEXP_REPLACE(
    TRIM(
        CASE
            WHEN SUBSTRING_INDEX(titel, ' ', -1) IN ('the', 'The', 'an', 'An', 'a', 'A')
            THEN CONCAT(SUBSTRING_INDEX(titel, ' ', -1), ' ', SUBSTRING_INDEX(titel, ' ', 1))
            ELSE titel
        END
    ),
    '[:.*]',
    ''
);
```

Check Match MovieLend and IMDB

```
-- Vereinte Filmdatentabelle (MovieLens und IMDB)
INSERT INTO temp_matched_filme (movielens_id, imdb_id, titel, originaltitel, jahr,
ist_erwachsen, start_jahr, end_jahr, dauer_minuten, titel_typ)
SELECT
    ml.movielens_id,
    imdb.imdb_id,
    ml.titel,
    imdb.originaltitel,
    ml.jahr,
    imdb.ist_erwachsen,
    imdb.start_jahr,
    imdb.end_jahr,
    imdb.dauer_minuten,
    imdb.titel_typ
FROM temp_filme AS ml
JOIN temp_imdb_filme AS imdb ON ml.normalized_titel = imdb.normalized_titel AND ml.jahr =
imdb.start_jahr;

-- Unvereinte MovieLenseinträge temp_unmatched_ml_filme
INSERT INTO temp_unmatched_ml_filme (movielens_id, titel, jahr)
SELECT
    ml.movielens_id,
    ml.titel,
    ml.jahr
FROM temp_filme AS ml
WHERE NOT EXISTS (
    SELECT 1
    FROM temp_imdb_filme AS imdb
    WHERE ml.normalized_titel = imdb.normalized_titel AND ml.jahr = imdb.start_jahr
);

-- Unvereinte IMDb Einträge in temp_unmatched_imdb_filme
INSERT INTO temp_unmatched_imdb_filme (imdb_id, titel, originaltitel, ist_erwachsen,
start_jahr, end_jahr, dauer_minuten, titel_typ)
SELECT
    imdb.imdb_id,
    imdb.titel,
    imdb.originaltitel,
    imdb.ist_erwachsen,
    imdb.start_jahr,
    imdb.end_jahr,
    imdb.dauer_minuten,
    imdb.titel_typ
FROM temp_imdb_filme AS imdb
WHERE NOT EXISTS (
    SELECT 1
    FROM temp_filme AS ml
    WHERE imdb.normalized_titel = ml.normalized_titel AND imdb.start_jahr = ml.jahr
);
```