Projektarbeit: Datenbank zur Filmverwaltung

1. Fachlicher Anwendungsbezug

Grundidee und Szenario

Grundidee der Datenbank ist die Entwicklung eines zentralen, digitalen Verzeichnisses zur Verwaltung der gemeinsamen Filmsammlung eines privaten Haushalts (z.B. einer Familie oder WG).

Als Ausgangsproblem besitzt ein filminteressierter Haushalt eine über Jahre gewachsene, gemeinsame Sammlung an Filmen und Serien. Diese sind auf diversen Medien (z.B. Blu-rays, DVDs, digital) verteilt, was zu einem zunehmenden Verlust der Übersicht führt.

Nutzen und Sinn

- **Primäres Ziel:** Die Datenbank soll Ordnung, Struktur und eine klare Übersicht in die gemeinsame Mediensammlung des Haushalts bringen und die Planung zukünftiger Filmabende oder -käufe erleichtern.
- **Zentrale Katalogisierung:** Alle Filme und Serien werden an einem einzigen Ort erfasst. Dies entkoppelt die Verwaltung vom physischen oder digitalen Speicherort der Medien.
- **Leistungsfähige Such- und Filterfunktionen:** Die Sammlung kann gezielt nach einer Vielzahl von Kriterien durchsucht werden, darunter Genre, Erscheinungsjahr oder Regisseur. Zusätzlich kann jeder Nutzer nach seiner eigenen, persönlichen Bewertung filtern.
- **Verwaltung einer "Watchlist":** Jeder Nutzer kann eine eigene, persönliche Wunschliste führen, um Filmempfehlungen oder Kaufwünsche systematisch zu erfassen.
- **Verwaltung bereits gesehener Filme:** Jeder Nutzer kann für sich markieren, welche Filme er bereits gesehen hat. Dabei können individuelle Informationen wie das Datum des Sehens und eine persönliche Bewertung hinterlegt werden.

Einschränkungen/Abgrenzung

- **Kein Streaming-Dienst oder Medien-Player:** Die Datenbank dient ausschließlich der Verwaltung von Metadaten. Sie beinhaltet nicht die eigentlichen Filmdateien und bietet keine integrierte Funktion zum Abspielen der Medien. Es werden keine kommerziellen Aspekte (Shopsystem, Lizenzverwaltung etc.) abgebildet.
- **Fokus auf private Nutzung im kleinen Kreis:** Die Benutzer- und Rollenverwaltung ist für einen privaten, überschaubaren Personenkreis ausgelegt und nicht auf Skalierbarkeit für tausende Nutzer ausgelegt.
- **Manuelle Dateneinpflege:** Es wird keine Schnittstelle zu externen, öffentlichen Filmdatenbanken implementiert. Alle Filminformationen müssen manuell eingegeben werden.
- **Kein Verleih- oder Bestandsmanagement:** Die Datenbank erfasst, welche Filme vorhanden sind, beinhaltet aber keine Funktion zur Verwaltung eines Verleihs an andere Personen.

---

2. Anforderungsanalyse

Funktionale Anforderungen

1. Verwaltung der Filmsammlung:

- Jedes `Mitglied` (und der `Administrator`) kann neue Filme, Serien und Personen in die Datenbank eintragen.
- Jedes `Mitglied` (und der `Administrator`) kann die Daten bereits existierender Einträge bearbeiten.
- Ausschließlich ein `Administrator` kann Einträge endgültig aus den Stammdaten (Filme, Personen etc.) löschen.
- Filme können einer übergeordneten `Filmreihe` zugeordnet werden.
- Die Beziehung zwischen Filmen und Personen wird über eine (n:m) Verknüpfungstabelle `Film_Beteiligungen` realisiert. Diese Tabelle speichert `filmID`, `personID` und die Rollen der Person (`istRegisseur`, `istSchauspieler`).

2. Personalisierte Nutzerfunktionen:

- Jeder registrierte Nutzer (`Administrator` oder `Mitglied`) kann Filme zu seiner persönlichen `Watchlist` hinzufügen oder davon entfernen.
- Jeder registrierte Nutzer (`Administrator` oder `Mitglied`) kann Filme als "gesehen" markieren und eine persönliche Bewertung (1-10) sowie ein Datum hinterlegen.
- **Gewährleistung der Privatsphäre:** Das System (via `VIEW`s) stellt sicher, dass ein Nutzer ausschließlich auf seine eigenen personalisierten Einträge zugreifen kann.

3. Datenauswertung und Suche:

- Alle Nutzer (`Gast` eingeschlossen) können die Sammlung nach Kriterien wie Titel, Genre, Regisseur oder Filmreihe durchsuchen.
- Ein `Gast` hat ausschließlich Lesezugriff auf öffentliche Filmdaten und kann keine personalisierten Listen einsehen oder bearbeiten.
- Das System kann Detailinformationen zu einem Film anzeigen, inklusive aller beteiligten Personen.
- Ein `Mitglied` oder `Admin` kann seine persönliche Watchlist und seine Liste der gesehenen Filme einsehen.

4. Benutzer- und Rollenverwaltung:

- Ein `Administrator` kann neue Benutzer anlegen und ihnen eine Rolle (`Administrator`, `Mitglied`, `Gast`) zuweisen.

Nicht-funktionale Anforderungen

- **Datenkonsistenz und -integrität:** Die Datenbank sichert durch `PRIMARY KEY`, `FOREIGN KEY` und `CHECK`-Constraints (z.B. `chk_bewertung`) die Stimmigkeit der Daten.
- **Bedienbarkeit und Zuverlässigkeit:** Die Datenbank ist durch das bereitgestellte SQL-Skript (`main.sql`) und die `README.md` auf einem anderen System lauffähig und reproduzierbar.
- **Sicherheit:** Der Zugriff ist über ein Berechtigungskonzept geregelt. Die Zuweisung von Rechten (`GRANT`) erfolgt auf Basis von Rollen (`rolle_admin`, `rolle_mitglied`, `rolle_gast`).

---

3. Begründungen des Entwurfs

Der Entwurf der Datenbank basiert auf zwei Ebenen: Dem strukturellen Aufbau (ERM) und der technischen Implementierung der Sicherheitslogik.

1. Begründungen zum ERM-Modell (Struktur)

Diese Entscheidungen betreffen direkt den Aufbau der Tabellen und Beziehungen, wie sie im ERM-Diagramm dargestellt sind.

- **Generalisierung der Entität "Personen":** Schauspieler und Regisseure werden nicht als getrennte Entitäten modelliert, sondern in einer generalisierten Entität `Personen` zusammengefasst.
    - *Grund:* In der Filmindustrie übernehmen Personen oft beide Rollen (z. B. Clint Eastwood). Getrennte Entitäten würden zu Redundanzen führen. Eine zentrale Tabelle vereinfacht die Pflege der Stammdaten massiv.
- **Modellierung der Film-Beteiligungen (n:m mit Attributen):** Die Beziehung zwischen `Filme` und `Personen` ist als eine einzige n:m-Beziehung modelliert. Anstatt zwei separate Beziehungen (`führt_regie`, `spielt_mit`) zu zeichnen, nutzen wir Attribute (`istRegisseur`, `istSchauspieler`) direkt an der Beziehungsraute.
    - *Grund:* Dies reduziert die Komplexität des Diagramms. Eine Person kann in einem einzigen Datensatz präzise einem Film zugeordnet werden, auch wenn sie mehrere Funktionen gleichzeitig ausübt.
- **Personalisierte Listen als Beziehungstabellen:** Die Funktionen "Watchlist" und "Gesehene Filme" sind als n:m-Beziehungen zwischen `Benutzer` und `Filme` realisiert.
    - *Grund:* Diese Daten gehören logisch weder allein zum Nutzer noch zum Film. Besonders bei `GeseheneFilme` sind Attribute wie `persoenlicheBewertung` und `gesehenAm` zwingend Eigenschaften der *Beziehung* selbst, was im ERM durch Attribute an der Raute dargestellt wird.
- **Explizite Entität "Benutzer":** Das ERM beinhaltet eine eigene Entität `Benutzer`, obwohl die Datenbank (MariaDB) technisch eigene User verwaltet.
    - *Grund:* Um fachliche Beziehungen (wie "Nutzer X hat Film Y auf der Watchlist") im ERM modellieren zu können, benötigen wir eine referenzierbare ID (`benutzerID`). System-User haben keine solche ID, daher ist diese Entität im Datenmodell zwingend erforderlich.
- **Auslagerung der Rollen in eine eigene Entität:** Die Benutzerrollen (`Administrator`, `Mitglied`, `Gast`) werden nicht als einfaches Textfeld in der Tabelle `Benutzer` gespeichert, sondern in einer eigenständigen Tabelle `Rollen` verwaltet und über einen Fremdschlüssel referenziert.
    - *Grund:* Dies gewährleistet die Datenintegrität und Normalisierung. Es verhindert inkonsistente Schreibweisen (z.B. "Admin" vs. "Administrator") und stellt sicher, dass Benutzern nur gültige, vordefinierte Rollenbezeichnungen zugewiesen werden können.
- **Normalisierung (Genres und Filmreihen):** Attribute wie `Genre` und `Filmreihe` wurden in eigenständige Entitäten ausgelagert (3. Normalform).
    - *Grund:* Vermeidung von Redundanz und Inkonsistenz (z. B. Schreibfehler). Änderungen an einer Bezeichnung wirken sich so sofort global auf alle verknüpften Filme aus.

2. Begründungen zur technischen Umsetzung (Logik & Sicherheit)

Diese Entscheidungen betreffen die Art und Weise, wie das Datenmodell mittels SQL technisch abgesichert und genutzt wird.

- **Implementierung von Sicherheit durch VIEWs:** Standardmäßig bietet MariaDB keine native Beschränkung auf Zeilenebene für denselben Benutzerkreis. Um dennoch sicherzustellen, dass Mitglieder nur ihre *eigenen* Listen bearbeiten können, wurden die VIEWs `MeineWatchlist` und `MeineGesehenenFilme` implementiert.
    - *Grund:* Der direkte Zugriff auf die Basistabellen `Watchlist` und `GeseheneFilme` wird Mitgliedern entzogen. Der Zugriff erfolgt ausschließlich über die VIEWs, welche die Daten dynamisch filtern.
- **Verwendung von `USER()` im VIEW-Filter:** Die Filterung erfolgt über die Bedingung `WHERE benutzerID = ... SUBSTRING_INDEX(USER()...)`.
    - *Grund:* Die Funktion `USER()` gibt den Benutzer zurück, der die Verbindung aufgebaut hat (den "Invoker"). Dies ist notwendig, damit der VIEW dynamisch auf den jeweils eingeloggten Benutzer reagiert.
- **Datenintegrität durch `WITH CHECK OPTION`:** Die VIEWs wurden mit der Klausel `WITH CHECK OPTION` definiert.
    - *Grund:* Dies verhindert, dass ein Benutzer Datensätze über den VIEW einfügt oder manipuliert, die nicht der Filterbedingung entsprechen. Konkret wird so technisch erzwungen, dass ein Benutzer (`benutzerID 4`) keinen Eintrag für einen anderen Benutzer (`benutzerID 1`) anlegen kann, da dies vom DBMS sofort blockiert wird.
- **Entkopplung von Authentifizierung und Datenhaltung:** Das System nutzt eine "doppelte Buchführung": Der technische Zugang erfolgt über MariaDB-User (Sicherheit), die fachliche Logik über die Tabelle `Benutzer` (Daten).
    - *Grund:* Diese Trennung erlaubt es, die strikte Rechteverwaltung des DBMS zu nutzen, ohne die fachlichen Daten mit Systeminterna zu vermischen. Die dynamische Verknüpfung erfolgt über den übereinstimmenden Benutzernamen.

3. Begründung des Rollenkonzepts

Das Berechtigungskonzept folgt dem Prinzip der geringsten Privilegien ("Principle of Least Privilege"), um die Datensicherheit zu maximieren.

- **Administrator:**
    - *Funktion:* Besitzt uneingeschränkten Vollzugriff.
    - *Grund:* Eine zentrale Instanz ist notwendig, um Stammdatenpflege (z.B. das endgültige Löschen von Filmen oder Personen) durchzuführen. Das Recht zum Löschen (`DELETE`) auf Stammdaten ist exklusiv dieser Rolle vorbehalten, um versehentlichen Datenverlust durch normale Nutzer zu verhindern.
- **Mitglied:**
    - *Funktion:* Darf die Sammlung erweitern und pflegen, sowie eigene Listen verwalten.

- *Grund:* Um eine kollaborative Pflege der Sammlung im Haushalt zu ermöglichen, benötigen Mitglieder Schreibrechte (`INSERT`, `UPDATE`) auf Katalogdaten. Das explizite Fehlen des `DELETE`-Rechts auf Stammdaten dient dem Schutz der gemeinsamen Sammlung. Der Zugriff auf `Watchlist` und `GeseheneFilme` ist auf die eigene `benutzerID` beschränkt, um die Privatsphäre zu wahren.

- **Gast:**
  - *Funktion:* Reiner Lesezugriff ("Read-Only").
  - *Grund:* Diese Rolle ermöglicht Dritten (z.B. Besuchern) einen Einblick in die Sammlung, ohne ein Risiko für die Datenintegrität darzustellen. Gäste können keine Daten manipulieren und haben keinen Zugriff auf personalisierte Nutzerdaten.

---

4. Technische Umsetzung

Inhalt der `main.sql` Datei

Das Skript `main.sql` ist in drei Hauptabschnitte unterteilt, die die gesamte Struktur und Sicherheit der Datenbank definieren:

1. **Abschnitt 1: Grundlegendes Datenbankschema**
   - Erstellt die Datenbank `filmverwaltung` (nachdem eine eventuell vorhandene Version gelöscht wurde).
   - Erstellt alle 9 Kerntabellen (`Filme`, `Personen`, `Benutzer`, `Rollen`, `Watchlist` etc.) mit den notwendigen Primärschlüsseln, Fremdschlüsseln, `UNIQUE`-Constraints und `CHECK`-Constraints.

2. **Abschnitt 2: Kernsystem und Berechtigungen**
   - Befüllt die Anwendungstabellen `Rollen` und `Benutzer` mit den Stammdaten für die Logik.
   - Erstellt die MariaDB-Systemrollen (`rolle_admin`, `rolle_mitglied`, `rolle_gast`).
   - Erstellt die MariaDB-Systembenutzer (z.B. 'julian', 'max', 'sophie') ohne Passwörter.
   - Erstellt die beiden Sicherheits-`VIEW`s (`MeineWatchlist`, `MeineGesehenenFilme`), die als "Brücke" zwischen den Systembenutzern und der Anwendungslogik dienen.
   - Vergibt detaillierte `GRANT`-Berechtigungen an die Rollen.
   - Weist den Benutzern ihre jeweiligen Rollen zu und setzt diese als `DEFAULT ROLE`, damit sie beim Login automatisch aktiv sind.

3. **Abschnitt 3: Datenbefüllung**
   - Befüllt die Tabellen `Genres` und `Filmreihen` mit den grundlegenden Kategorien.
   - Fügt Beispieldaten für `Personen` (Regisseure und Schauspieler) hinzu, die in den Filmen vorkommen.
   - Befüllt die Tabelle `Filme` mit einer umfangreichen Sammlung von Beispielfilmen, inklusive Metadaten.
   - Verknüpft Filme mit Personen über die Tabelle `Film_Beteiligungen` und legt dabei fest, ob die Person als Regisseur und/oder Schauspieler beteiligt war.
   - Befüllt die personalisierten Listen (`Watchlist`, `GeseheneFilme`) mit Beispieldaten für jeden Benutzer, sodass jeder Nutzer 3–5 Filme auf seiner Watchlist und seiner Liste gesehener Filme

hat. Dies ermöglicht das direkte Testen der personalisierten Funktionen und Abfragen.

---

5. SQL-Abfragen

Frage 1: "Welche Filme (Titel und Erscheinungsjahr) hat der Benutzer 'max' auf seiner persönlichen Watchlist?"

```
-- Frage 1:
-- Diese Abfrage kann nur als Benutzer mit Admin-Rechten ausgeführt werden!

SELECT
    F.titel,
    F.erscheinungsjahr
FROM
    Watchlist W

-- Verknüpfe die Watchlist-Einträge mit Filmen
JOIN
    Filme F ON W.filmID = F.filmID

-- Verknüpfe Watchlist-Einträge mit den Benutzern
JOIN
    Benutzer B ON W.benutzerID = B.benutzerID
WHERE
    B.benutzerName = 'max' -- Filtert auf Benutzer
ORDER BY
    F.titel;
```

Diese Abfrage dient dazu, die Inhalte der Watchlist für einen Benutzer auszugeben. Da die Tabelle `Watchlist` lediglich die IDs der Filme und Benutzer speichert, ist eine Verknüpfung (`JOIN`) mit den Stammdaten-Tabellen `Filme` und `Benutzer` notwendig. Nur so können Titel und Erscheinungsjahr angezeigt werden. Der abschließende `WHERE`-Filter auf den Benutzernamen 'max' schränkt die Ergebnismenge gezielt ein, sodass nur persönliche Einträge geliefert werden.

Frage 2: "Welche 5 Personen sind in der gesamten Sammlung am häufigsten als Schauspieler vertreten? Zeige den Namen der Person und die Anzahl der Filme, in denen sie mitspielt."

```
-- Frage 2:
-- Abfrage nutzt Aggregation (COUNT) und Filter (WHERE istSchauspieler).

SELECT
    -- Kombiniere Vor- und Nachname für Ausgabe
    CONCAT(P.vorname, ' ', P.name) AS personName,

    -- Zähle Anzahl der Filmeinträge für die Person
    COUNT(FB.filmID) AS anzahlFilme
FROM
    Personen P
JOIN
    Film_Beteiligungen FB ON P.personID = FB.personID
```

```
WHERE
    -- Stelle sicher, dass die Person auch Schauspieler ist
    FB.istSchauspieler = TRUE
GROUP BY
    P.personID, personName -- Gruppiere die Zählung pro Person
ORDER BY
    anzahlFilme DESC -- Sortiere von der höchsten zur niedrigsten Anzahl
LIMIT 5; -- Zeige nur die Top 5 an
```

Ziel dieser Abfrage ist eine Auswertung der Sammlung, um die am häufigsten vertretenen Schauspieler herauszufinden. Die Basis bildet die Tabelle `Film_Beteiligungen`, wobei hier der Filter `istSchauspieler = TRUE` wichtig ist, um Regisseure auszuschließen. Mithilfe der Gruppierung (`GROUP BY`) werden alle Filmeinträge einer Person zusammengefasst, während `COUNT` die Anzahl dieser Einträge berechnet. Durch die Sortierung nach Häufigkeit (`DESC`) und die Begrenzung auf fünf Datensätze (`LIMIT 5`) entsteht die Top-5-Rangliste.

Frage 3: "Liste für jeden Benutzer (ausgenommen 'Gast') seine Top 3 am besten bewerteten Filme auf. Die Abfrage soll den Benutzernamen, den Filmtitel und die persönliche Bewertung anzeigen."

```
-- Frage 3:
-- Abfrage nutzt CTE (WITH...) und Window Function (ROW_NUMBER()).

-- 1. CTE definieren 'RankedFilme'
WITH RankedFilme AS (
    SELECT
        B.benutzerName,
        F.titel,
        GF.persoenlicheBewertung,

        -- Window Function: Erstellt eine separate Rangliste (rang) für jeden
Benutzer (PARTITION BY)
        -- sortiert nach der Bewertung von hoch nach niedrig (ORDER BY ... DESC)
        ROW_NUMBER() OVER(
            PARTITION BY B.benutzerName
            ORDER BY GF.persoenlicheBewertung DESC, F.titel ASC
        ) AS rang
    FROM
        GeseheneFilme GF
    JOIN
        Benutzer B ON GF.benutzerID = B.benutzerID
    JOIN
        Filme F ON GF.filmID = F.filmID
    JOIN
        Rollen R ON B.rollenID = R.rollenID
    WHERE
        R.rollenName != 'Gast' -- Schließt "Gast"-Benutzer aus
)

-- 2. Finale Abfrage:
-- Wähle nur Top 3 (rang <= 3) aus der CTE aus.
SELECT
    benutzerName,
```

```
    titel,
    persoenlicheBewertung,
    rang
FROM
    RankedFilme
WHERE
    rang <= 3
ORDER BY
    benutzerName, rang;
```

Diese Abfrage dient dazu, für jeden Benutzer separat eine Top-3-Liste mit Detailinfos zu erstellen. Da herkömmliche Gruppierungen hier nicht mehr ausreichen, wird eine **Common Table Expression (CTE)** zur Vorberechnung genutzt. Fensterfunktion `ROW_NUMBER()`: Durch die Anweisung `PARTITION BY` wird die Nummerierung für jeden Benutzer neu gestartet, während `ORDER BY` die Filme innerhalb dieser Benutzer-Partition nach ihrer Bewertung sortiert. So erhält jeder Film einen eindeutigen Rang (1, 2, 3...). Die eigentliche Hauptabfrage muss dann nur noch auf diese vorberechneten Ränge zugreifen und mittels `WHERE rang <= 3` die besten drei Filme pro Person herausfiltern.