Bericht Datenbanken
Team 5

1. Vorwort

Dies ist der Bericht, der als Dokumentation für das Datenbankprojekt von Team 5 dient. In dieser Dokumentation werden die getroffenen Entscheidungen sowie die Gründe für diese Entscheidungen ausführlich erläutert. Dafür werden das ER-Modell, das relationale Schema, die Normalisierung, die Implementierung, das Füllen der Tabellen mit Daten sowie die Evaluation aufgeführt.

Im Zuge unseres Projekts haben wir eine Datenbank für eine Filmkritik-Website geplant und umgesetzt. Dabei sollen Nutzer die Möglichkeit haben, Filme zu bewerten, Kritiken zu lesen und Filme zu suchen.

2. User Stories

2.1 Maximilian Sgodin:

2.1.1 Als Kritiker möchte ich auf der Seite die Bewertungen von anderen Kritikern zu Filmen sehen, um mir eine Einsicht einzuholen, ob es sich rentiert, den Film zu schauen.

Akzeptanzkriterien:

- Eine durchschnittliche Kritikerbewertung wird angezeigt, basierend auf allen vorhandenen Kritikerbewertungen für den Film.
- Nutzer können auf den Namen des Kritikers klicken, um alle Bewertungen dieses Kritikers zu sehen.
- Kritikerbewertungen können nach Datum, Bewertung (höchste zuerst, niedrigste zuerst) oder Alphabet (Name des Kritikers) sortiert werden.

SQL:

Durchschnittliche Bewertung für einen bestimmten Film anzeigen:

```
SELECT
    f.Titel,
    AVG(b.Bewertungszahl) AS Durchschnittsbewertung,
    COUNT(b.Bewertungszahl) AS Bewertungsanzahl
FROM
    Film f
JOIN
    Bewertung b ON f.Film_ID = b.Film_ID
WHERE
    f.Film_ID = [Film_ID]  -- Setze die Film-ID entsprechend
GROUP BY
    f.Titel;
```

Alle Bewertungen für einen bestimmten Film anzeigen:

```
SELECT
    k.UserName AS KritikerName,
    b.Bewertungszahl,
    b.Kommentar,
    b.Verfassungsdatum
FROM
    Bewertung b
JOIN
    Kritiker k ON b.Kritiker_ID = k.Kritiker_ID
WHERE
    b.Film_ID = [Film_ID]  -- Setze die Film-ID entsprechend
```

```
ORDER BY
    b.Verfassungsdatum DESC;  -- Sortierung nach Datum, absteigend (jüngste zuerst)
```

Oder:

```
ORDER BY
    b.Bewertungszahl DESC;  -- Sortierung nach Bewertungspunktzahl (höchste zuerst)


ORDER BY
    b.Bewertungszahl ASC;  -- Sortierung nach Bewertungspunktzahl (niedrigste zuerst)


ORDER BY
    k.UserName ASC;  -- A-Z  Sortierung nach Kritikername (alphabetisch)
```

Dies trifft auch auf die folgenden Anfragen zu, bei denen es Sortiermöglichkeiten gibt.

Kritikerprofil-Anzeigen:

```
SELECT
    b.Film_ID,
    f.Titel,
    b.Bewertungszahl,
    b.Kommentar,
    b.Verfassungsdatum
FROM
    Bewertung b
JOIN
    Film f ON b.Film_ID = f.Film_ID
WHERE
    b.Kritiker_ID = [Kritiker_ID]  -- Setze die Kritiker-ID entsprechend
```

ORDER BY

b.Verfassungsdatum DESC; -- Sortierung nach Datum, absteigend (jüngste zuerst)

Erklärung:

Durchschnittliche Bewertung anzeigen:

- Diese Abfrage berechnet die durchschnittliche Bewertung und die Anzahl der Bewertungen für einen bestimmten Film. Sie zeigt den Titel des Films, die Durchschnittsbewertung und die Bewertungsanzahl an.
- Ersetze `[Film_ID]` mit der tatsächlichen ID des Films.

Alle Bewertungen anzeigen und sortieren:

- Diese Abfrage zeigt alle Bewertungen für einen bestimmten Film an, einschließlich des Namens des Kritikers, der Bewertungspunktzahl, des Kommentars und des Verfassungsdatums.
- Die Ergebnisse können nach Verfassungsdatum, Bewertungspunktzahl oder Kritikername sortiert werden. Die Sortierreihenfolge kann durch Anpassen der `ORDER BY`-Klausel geändert werden.

Kritikerprofil-Anzeige:

- Diese Abfrage zeigt alle Bewertungen eines bestimmten Kritikers an, wenn auf den Kritikername geklickt wird.
- Ersetze `[Kritiker_ID]` mit der tatsächlichen ID des Kritikers.

2.1.2 Als ein Bespaucher der Webseite möchte ich detaillierte Informationen über einen Film anzeigen können, damit ich entscheiden kann, ob ich den Film ansehen möchte.

Akzeptanzkriterien:

- Die Filmseite zeigt den Titel, das Erscheinungsdatum und das Genre an.
- Eine Liste der Hauptdarsteller und des Regisseurs wird angezeigt.
- Die durchschnittliche Benutzerbewertung und die Anzahl der Rezensionen sind sichtbar.

SQL:

Filmdetails anzeigen

```
SELECT
	f.Titel,
	f.Erscheinungsjahr,
	GROUP_CONCAT(g.Name SEPARATOR ', ') AS Genre,
	AVG(b.Bewertungszahl) AS Durchschnittsbewertung,
	COUNT(b.Bewertungszahl) AS Bewertungsanzahl
FROM
	Film f
LEFT JOIN
	ist_Genre ig ON f.Film_ID = ig.Film_ID
LEFT JOIN
	Genre g ON ig.Genre_ID = g.Genre_ID
LEFT JOIN
	Bewertung b ON f.Film_ID = b.Film_ID
WHERE
	f.Film_ID = [Film_ID]  -- Setze die Film-ID entsprechend
GROUP BY
	f.Titel, f.Erscheinungsjahr;
```

Liste der Hauptdarsteller und des Regisseurs anzeigen

```
-- Hauptdarsteller anzeigen
SELECT
	p.Name AS SchauspielerName,
	si.Rolle,
	si.Gage
FROM
	spielt_in si
JOIN
	Person p ON si.Person_ID = p.Person_ID
```

```
WHERE
        si.Film_ID = [Film_ID];  -- Setze die Film-ID entsprechend

-- Regisseur anzeigen
SELECT
        p.Name AS RegisseurName,
        ri.Gage
FROM
        regie_in ri
JOIN
        Person p ON ri.Person_ID = p.Person_ID
WHERE
        ri.Film_ID = [Film_ID];  -- Setze die Film-ID entsprechend
```

Erklärung:

Filmdetails anzeigen:

- Diese Abfrage zeigt den Titel des Films, das Erscheinungsjahr, das Genre (als kommagetrennte Liste), die durchschnittliche Bewertung und die Anzahl der Bewertungen an.
- `LEFT JOIN` wird verwendet, um sicherzustellen, dass alle Filme angezeigt werden, auch wenn keine Bewertung oder Genre-Zuordnung vorliegt.
- Ersetze `[Film_ID]` mit der tatsächlichen ID des Films.

Liste der Hauptdarsteller und des Regisseurs anzeigen:

- **Hauptdarsteller**: Diese Abfrage zeigt die Namen der Hauptdarsteller, ihre Rollen und ihre Gagen an.
- **Regisseur**: Diese Abfrage zeigt den Namen des Regisseurs und seine Gage an.
- Ersetze `[Film_ID]` mit der tatsächlichen ID des Films.

2.1.3 Als Benutzer der Webseite möchte ich eine Liste der am besten bewerteten Filme in einem bestimmten Genre sehen, damit ich entscheiden kann, welche Filme ich in diesem Genre anschauen möchte.

Akzeptanzkriterien:

- Eine Liste der Filme mit dem Titel, Erscheinungsjahr und durchschnittlichen Benutzerbewertungen wird angezeigt.
- Die Filme werden nach Genre und durchschnittlicher Benutzerbewertung gefiltert.
- Die Ergebnisse werden nach der durchschnittlichen Bewertung sortiert (höchste zuerst).

SQL:

```
SELECT
    f.Titel,
    f.Erscheinungsjahr,
    AVG(b.Bewertungszahl) AS Durchschnittsbewertung
FROM
    Film f
JOIN
    Bewertung b ON f.Film_ID = b.Film_ID
JOIN
    ist_Genre ig ON f.Film_ID = ig.Film_ID
JOIN
    Genre g ON ig.Genre_ID = g.Genre_ID
WHERE
    g.Name = '[Genre]'  -- Setze hier das gewünschte Genre ein
GROUP BY
    f.Titel, f.Erscheinungsjahr
HAVING
    AVG(b.Bewertungszahl) IS NOT NULL
ORDER BY
    Durchschnittsbewertung DESC;
```

Erklärung:

Filme und Bewertungen verknüpfen:

- Die Tabelle `Film` wird mit der Tabelle `Bewertung` verknüpft, um die Bewertungen der Filme zu erhalten.

Genre hinzufügen:

- Die Tabelle `ist_Genre` wird genutzt, um die Genre-Informationen mit den Filmen zu verknüpfen.
- Die Tabelle `Genre` liefert den Namen des Genres.

Nach Genre filtern:

- Die `WHERE`-Bedingung filtert die Filme nach dem gewünschten Genre. Ersetze `[Genre]` mit dem Genre, z.B., `'Action'`.

Durchschnittliche Bewertung berechnen:

- `AVG(b.Bewertungszahl)` berechnet die durchschnittliche Bewertung der Filme.

Ergebnisse gruppieren:

- `GROUP BY f.Titel, f.Erscheinungsjahr` gruppiert die Ergebnisse nach Titel und Erscheinungsjahr der Filme.

Nur Filme mit Bewertungen anzeigen:

- `HAVING AVG(b.Bewertungszahl) IS NOT NULL` stellt sicher, dass nur Filme mit Bewertungen angezeigt werden.

Nach Bewertung sortieren:

- `ORDER BY Durchschnittsbewertung DESC` sortiert die Filme nach der durchschnittlichen Bewertung, beginnend mit der höchsten.

2.1.4 Als Benutzer möchte ich eine Liste der Filme sehen, die in einem bestimmten Zeitraum die meisten Bewertungen erhalten haben, um zu erfahren, welche Filme aktuell am meisten diskutiert werden.

Akzeptanzkriterien:

- Eine Liste der Filme mit den meisten Bewertungen in einem bestimmten Zeitraum wird angezeigt.
- Die Filme werden nach der Anzahl der Bewertungen in diesem Zeitraum sortiert, von den meisten zu den wenigsten.
- Die Liste zeigt den Filmtitel, die Anzahl der Bewertungen in diesem Zeitraum und die durchschnittliche Bewertung an.
- Die Abfrage berücksichtigt auch die Genre-Filterung.

SQL:

```
SELECT
    f.Titel,
    COUNT(b.Bewertungszahl) AS AnzahlBewertungen,
    AVG(b.Bewertungszahl) AS Durchschnittsbewertung
FROM
    Film f
JOIN
    Bewertung b ON f.Film_ID = b.Film_ID
JOIN
    ist_Genre ig ON f.Film_ID = ig.Film_ID
JOIN
    Genre g ON ig.Genre_ID = g.Genre_ID
WHERE
    b.Verfassungsdatum BETWEEN '[Startdatum]' AND '[Enddatum]'  -- Setze hier den gewünschten Zeitraum ein
    AND g.Name = '[Genre]'  -- Optional: Filter nach Genre, entferne diese Zeile, wenn keine Genre-Filterung nötig ist
```

GROUP BY

    f.Titel

ORDER BY

    AnzahlBewertungen DESC;

Erklärung:

Filme und Bewertungen verknüpfen:

- Die Tabelle `Bewertung` wird mit der Tabelle `Film` verknüpft, um die Bewertungen der Filme zu erhalten.
- Die Tabelle `ist_Genre` wird genutzt, um die Genre-Informationen mit den Filmen zu verknüpfen.
- Die Tabelle `Genre` liefert den Namen des Genres.

Zeitraum und optional Genre filtern:

- Die `WHERE`-Bedingung filtert die Bewertungen nach einem bestimmten Zeitraum und optional nach einem bestimmten Genre. Ersetze `[Startdatum]`, `[Enddatum]` und `[Genre]` mit den gewünschten Werten.

Filme gruppieren:

- `GROUP BY f.Titel` gruppiert die Ergebnisse nach Filmtitel, um die Anzahl der Bewertungen und die durchschnittliche Bewertung für jeden Film zu berechnen.

Anzahl der Bewertungen und Durchschnitt berechnen:

- `COUNT(b.Bewertungszahl) AS AnzahlBewertungen` zählt die Anzahl der Bewertungen für jeden Film.
- `AVG(b.Bewertungszahl) AS Durchschnittsbewertung` berechnet die durchschnittliche Bewertung für jeden Film.

Ergebnisse sortieren:

- `ORDER BY AnzahlBewertungen DESC` sortiert die Filme nach der Anzahl der Bewertungen, beginnend mit den meisten Bewertungen.

2.1.5 Als Kritiker möchte ich eine Liste der Filme sehen, die ich bewertet habe, um einen Überblick über meine bisherigen Rezensionen zu erhalten.

Akzeptanzkriterien:

- Kritiker können eine Liste der von ihnen bewerteten Filme sehen.
- Die Liste zeigt den Filmtitel, die Bewertung, den Kommentar und das Datum der Bewertung an.
- Kritiker können die Liste nach Erscheinungsdatum oder Bewertung sortieren.

SQL:

```
SELECT
    f.Titel,
    b.Bewertungszahl,
    b.Kommentar,
    b.Verfassungsdatum
FROM
    Bewertung b
JOIN
    Film f ON b.Film_ID = f.Film_ID
WHERE
    b.Kritiker_ID = [Kritiker_ID]  -- Setze hier die Kritiker-ID ein
ORDER BY
    b.Verfassungsdatum DESC;  -- Standard-Sortierung nach Datum (jüngste zuerst)
```

Erklärung:

Filme und Bewertungen verknüpfen:

- Die Tabelle `Bewertung` wird mit der Tabelle `Film` verknüpft, um die Filminformationen zu den Bewertungen zu erhalten.

Nach Kritiker filtern:

- Die `WHERE`-Bedingung filtert die Bewertungen nach der Kritiker-ID. Ersetze `[Kritiker_ID]` mit der ID des Kritikers.

Ergebnisse anzeigen:

- `SELECT f.Titel, b.Bewertungszahl, b.Kommentar, b.Verfassungsdatum` wählt die benötigten Informationen aus: Filmtitel, Bewertung, Kommentar und das Datum der Bewertung.

Standard-Sortierung:

- `ORDER BY b.Verfassungsdatum DESC` sortiert die Ergebnisse standardmäßig nach dem Datum der Bewertung, beginnend mit der jüngsten Bewertung.

2.2 Denis Francesco Volpe:

2.2.1 Als Administrator möchte ich sehen, wie der Bewertungsstil der Kritiker ist, nachdem sie Kommentare verfasst habe

Akzeptanzkriterien:

- Wenn die Durchschnittsbewertung <= 4 ist, wird der Bewertungsstil auf 'Bad' gesetzt.
- Wenn die Durchschnittsbewertung zwischen 5 und 7 liegt, wird der Bewertungsstil auf 'Neutral' gesetzt.

- Wenn die Durchschnittsbewertung > 7 ist, wird der Bewertungsstil auf 'Good' gesetzt.

```sql
-- Beginn der Transaktion
START TRANSACTION;

-- wir machen eine temporöre Tabelle um auf der rechen/arbeiten zu können
CREATE TEMPORARY TABLE Kritiker_Bewertungsdurchschnitt (
    Kritiker_ID int(11) NOT NULL,
    Durchschnittsbewertung float NOT NULL,
    PRIMARY KEY (Kritiker_ID)
) ENGINE=InnoDB;

-- füge in die Tabelle eine berechnete Durschnittszahl von allen Kritiken eines Kritikers ein,
-- sowie seine ID
INSERT INTO Kritiker_Bewertungsdurchschnitt (Kritiker_ID, Durchschnittsbewertung)
SELECT
    Kritiker_ID,
    AVG(Bewertungszahl) AS Durchschnittsbewertung
FROM
    Bewertung
GROUP BY
    Kritiker_ID;

-- Update die Kritiker Tabelle mit neuen Bewertungsstil, Enum wird durch untere
-- Bewertungschema geändert
UPDATE Kritiker k
JOIN Kritiker_Bewertungsdurchschnitt kbd ON k.Kritiker_ID = kbd.Kritiker_ID
SET k.Bewertungsstil = CASE
    WHEN kbd.Durchschnittsbewertung <= 4 THEN 'Bad'
    WHEN kbd.Durchschnittsbewertung BETWEEN 5 AND 7 THEN 'Neutral'
    ELSE 'Good'
END;

-- Entfernen der temporären Tabelle
DROP TEMPORARY TABLE Kritiker_Bewertungsdurchschnitt;

-- Transaktion erfolgreich abschließen
COMMIT;
```

Erklärung:
Der Code startet mit der Anweisung `START TRANSACTION;`, um eine neue Transaktion zu beginnen. Dies stellt sicher, dass alle nachfolgenden SQL-Anweisungen als eine atomare Einheit ausgeführt werden.
Danach erstellen wir eine temporäre Tabelle um auf der rechnen zu können. Diese Tabelle enthält zwei Spalten: `Kritiker_ID` (vom Datentyp `int(11)`, nicht null) und `Durchschnittsbewertung` (vom Datentyp `float`, nicht null). Die `Kritiker_ID` ist der Primärschlüssel dieser Tabelle.
In die temporäre Tabelle werden die Kritiker-IDs und ihre Durchschnittsbewertungen eingefügt. Diese Daten werden aus der `Bewertung`-Tabelle abgerufen. Die Durchschnittsbewertung wird für jeden Kritiker berechnet, indem der Durchschnitt der Bewertungszahlen (`Bewertungszahl`) für jede `Kritiker_ID` gebildet wird.
Die `Kritiker`-Tabelle wird basierend auf den Durchschnittsbewertungen aktualisiert. Hierbei wird die `Kritiker`-Tabelle mit der temporären Tabelle `Kritiker_Bewertungsdurchschnitt` verbunden (mittels JOIN) und der `Bewertungsstil` der Kritiker wird aktualisiert.

- Wenn die Durchschnittsbewertung eines Kritikers kleiner oder gleich 4 ist, wird der `Bewertungsstil` auf 'Bad' gesetzt.
- Wenn die Durchschnittsbewertung zwischen 5 und 7 liegt, wird der `Bewertungsstil` auf 'Neutral' gesetzt.

- Wenn die Durchschnittsbewertung größer als 7 ist, wird der `Bewertungsstil` auf 'Good' gesetzt.

Danach wird die temporäre Tabelle gelöscht und die Transaktion mit `COMMIT;` abgeschlossen.

2.2.2 Als Kritiker möchte ich eine Bewertung abgeben

Akzeptanzkriterien:

- Bewertungen sind zwischen 1-10
- der Filmstatus (Seen, Unseen, Dropped) muss ebenfalls angegeben werden
- Voraussgesetzt ich kenne meinen UserName und den Filmnamen

```
DELIMITER //

-- Hier muss eine Prozedur erstellt werden weil MySQl keine IF,ELSE sonst erkennt
CREATE PROCEDURE InsertBewertung()
BEGIN
    DECLARE kritikerID INT;
    DECLARE filmID INT;

    -- Transaktion starten
    START TRANSACTION;

    -- Kritiker_ID ermitteln
    SELECT Kritiker_ID INTO kritikerID
    FROM Kritiker
    WHERE UserName LIKE '%FilmFanatic%'
    LIMIT 1;

    -- Film_ID ermitteln
    SELECT Film_ID INTO filmID
    FROM Film
    WHERE Titel LIKE '%Memento%'
    LIMIT 1;

    -- Prüfen, ob beide IDs gefunden wurden
    IF kritikerID IS NOT NULL AND filmID IS NOT NULL THEN
        -- Bewertung hinzufügen wenn kritikerID und filmID existieren
        INSERT INTO Bewertung (Kritiker_ID, Film_ID, Bewertungszahl, Kommentar, Verfassungsdatum, Filmstatus)
        VALUES (kritikerID, filmID, 8, 'A thrilling movie with an unexpected twist.', '2024-06-21', 'Seen');

        -- Transaktion abschließen
        COMMIT;
    ELSE
        -- Transaktion rückgängig machen
        ROLLBACK;
    END IF;
END //

DELIMITER ;

-- Stored Procedure aufrufen
CALL InsertBewertung();
```

Erklärung:

Der Code beginnt mit der Erstellung einer Prozedur namens `InsertBewertung`. Dies ist notwendig, weil MySQL keine `IF-ELSE`-Anweisungen außerhalb von Prozeduren unterstützt.

Wir starten dann erneut eine Transaktion, um eigenschaften einer konsistenten Datenbank zu gewährleisten, wie in 2.2.1 erklärt worden ist.

Die Prozedur sucht nach der `Kritiker_ID` des Kritikers, dessen Benutzername den Teilstring 'FilmFanatic' enthält. Das Ergebnis wird in die Variable `kritikerID`

gespeichert. Hierbei wird die `Kritiker`-Tabelle durchsucht und nur das erste Ergebnis (`LIMIT 1`) berücksichtigt. Das bedeutet das wir nur nach maximal einem passenden Kritiker Username suchen.
Diese gleiche Suche nach der `Film_ID` erfolgt nach dem gleichen Schema wie bei `Kritiker_ID`.
Die gespeicherten Film- und Kritiker-ID´s werden in den Variablen `kritikerID` als auch `filmID`. Diese dürfen nicht `NULL` sein.
Falls diese nicht `NULL` sind, kann eine Bewertung eingefügt werden.
Eine Bewertung kann folgende Informationen enthalten:

- `Kritiker_ID`: Die ID des Kritikers.
- `Film_ID`: Die ID des Films.
- `Bewertungszahl`: Die Bewertungszahl (hier 8).
- `Kommentar`: Ein Kommentar zur Bewertung ('A thrilling movie with an unexpected twist.').
- `Verfassungsdatum`: Das Datum der Bewertung ('2024-06-21').
- `Filmstatus`: Der Status des Films ('Seen').

Falls entweder `kritikerID` oder `filmID` nicht gefunden wurde (also `NULL` ist), wird die Transaktion mit `ROLLBACK` abgebrochen. Dadurch werden alle bis dahin durchgeführten Änderungen rückgängig gemacht.
Danach wird Die Prozedur wird mit `END` beendet.
Der Code beendet die Prozedurdefinition und setzt den SQL-Delimiter wieder auf das Standardzeichen “`;`”.
Zum Schluss wird die gespeicherte Prozedur `InsertBewertung` mit `CALL InsertBewertung();` aufgerufen.

2.2.3 Als Kritiker möchte ich meine Bewertung verändern weil ich meine meinung geändert habe

Akzeptanzkriterien:

- Bewertungszahl, Kommentar, und Filmstatus lassen sich verändern
- das datum der verfassung ändert sich auf den geänderten Zeitpunkt

```sql
START TRANSACTION;
UPDATE Bewertung
SET Bewertungszahl = 9,
    Kommentar = 'An even more thrilling movie with a deeper twist.',
    -- innodb schreibweise für current date (NOW() würde noch zeit angeben)
    -- in zukunft -> erfassen ob etwas bearbeitet wurde
    Verfassungsdatum = CURDATE(),
    Filmstatus = 'Seen'
WHERE Kritiker_ID = (
    SELECT Kritiker.Kritiker_ID
    FROM Kritiker
    WHERE Kritiker.UserName LIKE '%Suchstring%'
) AND Film_ID = (
    SELECT Film.Film_ID
    FROM Film
    WHERE Film.Titel LIKE '%Suchstring%'
);
COMMIT;
```

Erklärung:
Wir starten dann erneut eine Transaktion, um eigenschaften einer konsistenten Datenbank zu gewährleisten, wie in 2.2.1 erklärt worden ist.
Folgende Spalten der `Bewertung`-Tabelle werden aktualisiert:

- `Bewertungszahl`: Der Wert wird auf 9 gesetzt.
- `Kommentar`: Der Wert wird auf 'An even more thrilling movie with a deeper twist.' gesetzt.
- `Verfassungsdatum`: Das Datum der Bewertung wird auf das aktuelle Datum gesetzt. Dies wird mit der Funktion `CURDATE()` erreicht, die das aktuelle Datum ohne die Zeitangabe liefert, da Bewertungen mit aktuellen Attributen nur ein Verfassungsdatum speichern kann.
- `Filmstatus`: Der Wert wird auf 'Seen' gesetzt.

Die Bedingung für die Aktualisierung sind passend gefundene Kritiker UserNames sowie Filmnamen die ebenfalls per %Suchstring% ermittelt werden.

2.2.4 Als Benutzer möchte ich die durchschnittliche Bewertung und die Anzahl der Bewertungen pro Film für alle Filme eines bestimmten Genres sehen (basierend auf Kritiker-Bewertungen), um die beliebtesten Filme dieses Genres zu identifizieren. Die gezeigte Liste sollte absteigend sein.

Akzeptanzkriterien:

- Die Abfrage gibt die Film-ID, den Filmtitel, das Genre, die durchschnittliche Bewertung und die Anzahl der Bewertungen zurück.

- Es werden beispielsweise nur Filme des Genres "Action" angezeigt. (beliebige Genres möglich)
- Die Ergebnisse sind nach der durchschnittlichen Bewertung in absteigender Reihenfolge sortiert.

```
-- Hiermit wähle ich die Informationen aus die ich später in einer Zeile darstellen möchte
SELECT Film.Film_ID, Film.Titel, Genre.Name,
       AVG(Bewertung.Bewertungszahl) AS Durchschnittsbewertung,
       COUNT(Bewertung.Bewertungszahl) AS AnzahlBewertungen


-- INNER JOIN : combiniert Tabellen in dem die gleichen Werte auftreten

-- Daten werden aus Film entnommen
FROM Film
-- INNER JOIN : es werden nur Filme ausgewählt die eine Bewertung haben
   JOIN Bewertung ON Film.Film_ID = Bewertung.Film_ID
-- INNER JOIN : ordnet zu welcher Film welches Genre ist
   JOIN ist_Genre ON Film.Film_ID = ist_Genre.Film_ID
-- INNER JOIN : löst auf welche ist_Genre ID das tatsächliche Genre ist von der Genre Entität
   JOIN Genre ON ist_Genre.Genre_ID = Genre.Genre_ID

-- filtert nur wo der GenreName Action ist
WHERE Genre.Name = 'Action'
-- Gruppiert nach Filmtitel, Film_ID und GenreName fur AVG und COUNT
GROUP BY Film.Film_ID, Film.Titel, Genre.Name
-- FIlm muss mindestens eine Bewertung über 0 haben
HAVING COUNT(Bewertung.Bewertungszahl) > 0
-- Sortiere so dass es absteigend angezeigt wird
ORDER BY Durchschnittsbewertung DESC;
```

Der Code wählt Informationen über ein gesuchtes Genre aus der Datenbank aus und stellt sie in einer Zeile dar. Diese Informationen sind absteigend sortiert nach der Durchschnittsbewertung.
Die Daten werden aus mehreren Tabellen entnommen und miteinander verknüpft:

- Die Haupttabelle ist `Film`.
- Mit `INNER JOIN Bewertung` werden nur die Filme ausgewählt, die mindestens eine Bewertung haben. Die Verknüpfung erfolgt über die Spalte `Film_ID`.
- Mit `INNER JOIN ist_Genre` wird jedes Film-Genre zugeordnet. Auch hier erfolgt die Verknüpfung über die Spalte `Film_ID`.
- Mit `INNER JOIN Genre` wird das tatsächliche Genre des Films aus der `Genre`-Tabelle abgerufen. Diese Verknüpfung erfolgt über die Spalte `Genre_ID`.

Es werden nur die Filme ausgewählt, deren Genre 'Action' ist. Dies wird durch die Bedingung `WHERE Genre.Name = 'Action'` erreicht. Diese kann auch beliebig für andere Genres geändert werden.Die Ergebnisse werden nach `Film.Film_ID`, `Film.Titel` und `Genre.Name` gruppiert. Dies ist notwendig, um die Durchschnittsbewertung und die Anzahl der Bewertungen korrekt zu berechnen. Dies wird erreicht mit dem `GROUP BY …` erreicht.
Zusätzlich werden nur Filme berücksichtigt die mindestens eine Bewertung haben. Dies wird durch die Bedingung `HAVING COUNT(Bewertung.Bewertungszahl) > 0` sichergestellt.
Zum Schluss werden die Ergebnisse absteigend abhängig von der Durchschnittsbewertung dargestellt.
Dies wird durch die Anweisung `ORDER BY Durchschnittsbewertung DESC` erreicht.

2.2.5 Als Benutzer möchte ich verfolgen können welcher Kritiker noch aktiv ist oder nicht

Akzeptanzkriterien:

- Die Anzahl der Bewertungen jedes Kritikers wird angezeigt.
- Der Status jedes Kritikers (aktiv oder inaktiv) wird angezeigt. Mehr als 3 Monate ohne Bewertung wird als Inaktivität vermerkt
- Kritiker, die in den letzten sechs Monaten keine Bewertung abgegeben haben, werden als inaktiv markiert.

```sql
-- kreiere eine Temporäre tabelle für Berechnung
CREATE TEMPORARY TABLE Kritiker_Aktivität_Temp (
Kritiker_ID int(11) NOT NULL,
UserName varchar(255) NOT NULL,
Anzahl_Bewertungen int NOT NULL,
Status varchar(10) NOT NULL
);

-- die Tabelle soll Kritiker Aktivitäten anzeigen
INSERT INTO Kritiker_Aktivität_Temp (Kritiker_ID, UserName, Anzahl_Bewertungen, Status)
SELECT
    k.Kritiker_ID,
    k.UserName,

    -- zählt die insgesammte Anzahl an Bewertungen pro Kritiker_ID auf
    COUNT(b.Bewertungszahl) AS Anzahl_Bewertungen,

    -- checkt ob das datum älter als 3 monate ist falls ja dann inaktiv
    -- MAX gibt den größten Wert zurück also den "neuste" Bewertungsdatum um zu schauen wie weit dieser zurückliegt
    CASE
        WHEN MAX(b.Verfassungsdatum) < DATE_SUB(CURRDATE(), INTERVAL 3 MONTH) THEN 'Inaktiv'
        ELSE 'Aktiv'

    -- Ergebnis wird als Status gespeichert
    END AS Status

-- k als kurzform für Kritiker tabelle
FROM
    Kritiker k

-- berücksichtigt Datensätze wo Kritiker eine Bewertung abgegeben haben
JOIN
    Bewertung b ON k.Kritiker_ID = b.Kritiker_ID

-- gruppiert für Aggregationsfunktionen
GROUP BY
    k.Kritiker_ID, k.UserName;

-- Gebe mir alle Daten von der Temporären Tabelle zurück
SELECT * FROM Kritiker_Aktivität_Temp;
DROP TEMPORARY TABLE Kritiker_Aktivität_Temp;
```

Eine temporäre Tabelle namens `Kritiker_Aktivität_Temp` wird erstellt. Diese Tabelle hat folgende Spalten:

- `Kritiker_ID`: Eine Ganzzahl (int(11)), die die ID des Kritikers darstellt und nicht null sein darf.
- `UserName`: Ein VARCHAR-Feld mit einer maximalen Länge von 255 Zeichen, das den Benutzernamen des Kritikers enthält und nicht null sein darf.
- `Anzahl_Bewertungen`: Eine Ganzzahl, die die Anzahl der Bewertungen des Kritikers darstellt und nicht null sein darf.
- `Status`: Ein VARCHAR-Feld mit einer maximalen Länge von 10 Zeichen, das den Status des Kritikers (aktiv oder inaktiv) enthält und nicht null sein darf.

Daten werden in die temporäre Tabelle `Kritiker_Aktivität_Temp` eingefügt, indem Informationen aus den Tabellen `Kritiker` und `Bewertung` abgefragt werden:

- `Kritiker_ID`: Die ID des Kritikers aus der Tabelle `Kritiker`.
- `UserName`: Der Benutzername des Kritikers aus der Tabelle `Kritiker`.
- `Anzahl_Bewertungen`: Die Anzahl der Bewertungen, die der Kritiker abgegeben hat, berechnet mit der Aggregatfunktion `COUNT(b.Bewertungszahl)`.
- `Status`: Der Status des Kritikers, der basierend auf dem Datum der letzten Bewertung berechnet wird. Wenn das Datum der letzten Bewertung (bestimmt durch `MAX(b.Verfassungsdatum)`) älter als drei Monate ist, wird der Status auf 'Inaktiv' gesetzt. Andernfalls wird der Status auf 'Aktiv' gesetzt

Die Daten werden aus den Tabellen `Kritiker` (`k`) und `Bewertung` (`b`) entnommen. Dabei wird die Tabelle `Kritiker` mit der Tabelle `Bewertung` verknüpft mit einem INNER JOIN.
Die Daten werden nach `Kritiker_ID` und `UserName` gruppiert, um Aggregatfunktionen wie `COUNT` und `MAX` korrekt berechnen zu können. Dies geschieht mit einem GROUP BY.
Zum Schluss wollen wir alle Daten der `Kritiker_Aktivität_Temp;` zurückgeben lassen um uns die Daten sichtbar zu machen.
Danach wird die temporäre Tabelle gelöscht, da diese nicht mehr benötigt wird.

2.3 Stephane Sandevski

2.3.1 Als Benutzer möchte ich wissen, wie viel die Schauspieler verdienen. Und sehen, wer am meisten verdient.

Akzeptanzkriterien:

- Liste mit meist-bezahlten Schauspielern.
- Möglichkeit nach Kriterien sortieren -> meist verdienter Schauspieler, Regisseur

SQL:

```
SELECT
    p.name, SUM(g.Gage) AS Gesamtgage --nimmt die Gage g summiert sie als Gesamtgage
FROM
    Person p  --nimmt es aus der tabelle Personen p
JOIN
    spielt_in g ON p.Person_ID = g.Person_ID --Joint tabelle mit Personen welche nur Schauspieler sind
GROUP BY
    p.name -- gruppiert Tabelle nach Name und Gesamtgage
ORDER BY
    Gesamtgage DESC; --Sortiert die Gesamtgage der Schauspieler aller Filme in der Datenbank von Höchsten nach niedrigsten
```

Erklärung:
SQL Code beginnt mit der Selektierung der Personen und deren Gesamtgage als Summe ihrer Gage, aus der Personen Tabelle.
Als nächstes Inner Joint man die Relation von Personen zu spielt_in, diese gibt alle Gagen der Personen die Schauspieler sind. Man erhält eine Tabelle von Schauspielern und ihrer Gage in all ihren mit gespielten Tabellen. Das

gleiche Schema lässt sich auch für die Regisseure implementieren indem man statt nach spielt_in nach regie_in filtert.

2.3.2 Als Benutzer möchte ich die am neusten erschienen Filme in einem bestimmten Genre sehen, damit ich interessante neue Filme finden kann.

Akzeptanzkriterien:

- Neuesten Filme in absteigender Reihenfolge ansehen
- Durchschnittsbewertung zu jedem Film

SQL:

```sql
SELECT
    f.Titel, f.Erscheinungsjahr, g.Name AS Genre, f.Bewertungsdurchschnitt -- nimmt Titel, Erscheinungsjahr und Bewetungsdurchschnitt FROM Film als f
FROM
    Film f -- von Tabelle Film
JOIN
    ist_Genre ig ON f.Film_ID = ig.Film_ID -- joint Tabelle Film mit der Tabelle ist_Genre basierend auf Spalte Film_ID
JOIN
    Genre g ON ig.Genre_ID = g.Genre_ID -- joint Tabelle ist_Genre mit der Tabelle Gerne basierend auf Spalte Genre_ID
WHERE
    g.Name = 'Action'-- -> Genre einfügen nach Interesse
ORDER BY
    g.Name, f.Erscheinungsjahr DESC; -- gibt anhand des Genres den die neuesten Filme an und die Durchschnittsbewertung
```

Erklärung:
SQL Code beginnt mit der Selektion von Titel, Erscheinungsjahr, dem Genre und dem Bewertungsdurchschnitt aus der Film Tabelle.
Ein Inner Joint wird benutzt um jedem Film das passende Genre_ID zu zuweisen.
Der zweite Inner Joint verknüpft die die Genre_IDs mit dem Namen des Genres
Dannach sucht man nach einem bestimmten Genre welches man interessant wird mit dem WHERE.
ORDER BY gibt dann den die Filme nach erscheinungsjahr aus mit den neusten als erster Stelle.
Die Tabelle besteht aus dem Titel, Erscheinungsjahr, Genre und dem Bewertungsdurchschnitt

2.3.3 Als Benutzer möchte ich nach Filmen suchen können, die von einem bestimmten Regisseur geleitet wurden, um deren Werke zu erkunden

Akzeptanzkriterien:

- Alle Filme von einem Regisseur anzeigen lassen
- Mit dem Erscheinungsdatum

SQL:

```sql
SELECT
    p.name  -- Namen der Person anzeigen
FROM
    Person p -- aus Person
JOIN
    regie_in g ON p.Person_ID = g.Person_ID -- verbindet alle Personen welche regie haben mit ihren Namen
GROUP BY
    p.name


-- Mit dem Ausgewählten Regisseur hier vortfahren
SELECT
    p.Name, f.Titel, f.Erscheinungsjahr -- Name von der Person mit relation regie_in zu film, Titel und Erscheinungsdatums aller Filme
FROM
    Person p -- aus Tabelle Person
JOIN
    regie_in ri ON p.Person_ID = ri.Person_ID -- Joint Tabelle Person p mit der Tabelle regie_in ri anhand der Person_ID
JOIN
    Film f ON ri.Film_ID = f.Film_ID -- Joint Tabelle regie_in ri  mit der Tabelle Film f anhand  der Film_ID
WHERE
    p.Name = 'Christopher Nolan' -- Regisseur nach Interesse und welcher Verfügbar ist eingeben
ORDER BY
    f.Erscheinungsjahr DESC, p.Name, f.Titel; -- Gibt Alle Filme dieses Regisseures in Erscheinungsjahr mit neuesten zu erst
```

Erklärung:

1.SQL Code hat die Aufgabe dem User anzuzeigen welche Regisseure es zur Auswahl gibt.
Selektiert aus der Personen Tabelle die Namen, mithilfe dem INNER JOIN wähle ich nur die Personen die eine regie_in Relation haben.
Und gebe sie dann aus,
2.SQL Code fängt an mit der Auswahl von Name der Personen, Titel und Erscheinungsjahr. (Zusätzlich könnte man auch den Bewertungsdurchschnitt der einzelnen Filme mit ausgeben)
Der erste INNER JOIN wählt alle Personen_IDs aus welche eine Relation regie_in haben
Der zweite INNER JOIN verbindet die Filme f mit den Regisseur_ID um so alle Regisseure mit allen ihren Filmen ausgeben zu können.
WHERE beschränkt die Suche, man sucht dann nur nach einen bestimmten Regisseur.
Die Ausgabe ist dann der Regisseur, Titel der Filme und deren Erscheingungsjahr DESC.

2.3.4 Als Benutzer möchte ich wissen wie streng jeder Kritiker bewertet, um so zu entscheiden, ob seine Meinung ernst zu nehmen ist

Akzeptanzkriterien:

- Durchschnittsbewertung der Kritiker
- Sortiert nach Durschnittsbewertungs DESC.

SQL:

```sql
SELECT
    k.Name AS Kritiker, AVG(b.Bewertungszahl) AS BewertungsDurchschnitt
FROM
    Bewertung b -- aus der Tabelle Bewertung b
JOIN
    Kritiker k ON b.Kritiker_ID = k.Kritiker_ID -- joint die Name der Kritiker mit ihrer zugehörigen avg bewertungsanzahl basierend auf der KritikerID
GROUP BY
    k.Name
ORDER BY
    BewertungsDurchschnitt DESC; -- Sortiert mit den Kritikern mit der höchsten Bewertungsdurchschnitt als erstes
```

Erklärung:
SQL Code selektiert die Namen k als Kritiker, den AVG der Bewertungszahl als BewertungsDurchschnitt
Der INNER JOIN verbindet alle Kritiker mit ihres zugehörigen BewertungsDurchschnittes. Gibt alle Kritiker mit ihren zugehörigen Durchschnitts DESC.

2.3.5
Als Kritiker möchte ich, wenn ich ein neues Lieblingsgenre habe, diese automatisch geändert wird anhand meines best-bewertensten Genres

Akzeptanzkriterien:

- Lieblingsgenre ist entsprechend der Bewertungsdurchschnitt angepasst
- Die Möglichkeit, mehrere Lieblingsgenres zu haben.

SQL:

```sql
START TRANSACTION;

-- Temporäre Tabelle mit den Durchschnittsbewertungen der Genres für den Kritiker erstellen
CREATE TEMPORARY TABLE Temp_Durchschnittsbewertung AS
SELECT g.Genre_ID, AVG(b.Bewertungszahl) AS Durchschnittsbewertung
FROM Bewertung b
JOIN Film f ON b.Film_ID = f.Film_ID
JOIN ist_Genre ig ON f.Film_ID = ig.Film_ID
JOIN Genre g ON ig.Genre_ID = g.Genre_ID
JOIN Kritiker k ON b.Kritiker_ID = k.Kritiker_ID
WHERE k.Kritiker_ID = 5  -- Hier die ID des Kritikers einfügen (1-5 als Kritiker_ID)
GROUP BY g.Genre_ID
ORDER BY Durchschnittsbewertung DESC;

-- Temporäre Tabelle mit den bestbewerteten Genres erstellen
CREATE TEMPORARY TABLE BestbewerteteGenres AS
SELECT Genre_ID
FROM Temp_Durchschnittsbewertung
WHERE Durchschnittsbewertung = (
    SELECT MAX(Durchschnittsbewertung)
    FROM Temp_Durchschnittsbewertung
);

-- Aktuelle Lieblingsgenres des Kritikers löschen
DELETE FROM LieblingsGenre
WHERE Kritiker_ID = 5;

-- Neue Lieblingsgenres in die Tabelle LieblingsGenre einfügen
INSERT INTO LieblingsGenre (Kritiker_ID, Genre_ID)
SELECT 5, Genre_ID
FROM BestbewerteteGenres;

COMMIT;
```

Erklärung:

Start der Transaktion, Transaktionen sind verübergehend, werden erst durch COMMIT dauerhaft.

Temporäre Tabelle: Temp_Durchschnittsbewertung. Gefühlt mit der Genre_ID, Bewertungszahl aus der Bewertungstabelle mit der Aggregation AVG als Durchschnittsbewertung Die 4 INNER JOINTs verbinden die Filme, Genre, Kritiker, Kritker_ID, ist_Genre und die Genre_ID. WHERE sucht nur nach der passenden Kritiker_ID so dass eine Tabelle vorliegt mit den Lieblingsgenres und ihren Durchschnittsbewertungen

Aus dieser Temporären Tabelle wird eine neue Temporäre Tabelle erstellt welche nur die höchst bewerteten Genres enthält mit Hilfe der Aggregation MAX zur Auswahl der Genres.

DELETE FROM löscht alle Einträge in LieblingsGenre für diese Kritiker_ID

INSERT INTO Wählt aus der BesbewertetGenre Temp_Tabelle die neuen LieblingsGenres aus und fügt sie ein

COMMIT; beendet die Transaktion.

zur Überprüfung ob die Transaktion erfolgreich war.
Neuer lg für Kritiker 5 sollte nun der höchstbewertete aus der 2. Temp_Tabelle sein.

```sql
SELECT g.Name AS LieblingsGenre
FROM LieblingsGenre lg
JOIN Genre g ON lg.Genre_ID = g.Genre_ID
WHERE lg.Kritiker_ID = 5;
```

2.4 Marius Ureche

2.4.1 Als Kritiker möchte ich eine Visualisierung der Karrieren meiner Lieblingsschauspieler sehen, damit ich verfolgen kann, in welchen Filmen sie im Laufe der Zeit mitgespielt haben und welche Genres sie bevorzugen.

Akzeptanzkriterien:

- Die Liste muss den Jahren der Filme zeigen, in denen die Schauspieler mitgespielt haben.
- Es sollte erkennbar sein, zu welchem Genre jeder Film gehört.

SQL:

```sql
SELECT
    f.Titel AS film_title,
    f.Erscheinungsjahr AS release_year,
    GROUP_CONCAT(g.Name SEPARATOR ', ') AS preferred_genres
FROM
    Person p
    INNER JOIN spielt_in si ON p.Person_ID = si.Person_ID
    INNER JOIN Film f ON si.Film_ID = f.Film_ID
    INNER JOIN ist_Genre ig ON f.Film_ID = ig.Film_ID
    INNER JOIN Genre g ON ig.Genre_ID = g.Genre_ID
WHERE
    p.Name = 'Schauspieler Name'
GROUP BY
    f.Film_ID, f.Titel, f.Erscheinungsjahr
ORDER BY
    f.Erscheinungsjahr;
```

Erklärung:
Die Abfrage in 2.4.1 zeigt eine Liste von Filmen an, in denen ein bestimmter Schauspieler mitgespielt hat, zusammen mit den Genres dieser Filme. Der Code verwendet eine Kombination von Tabellen und Joins, um die Daten zu verknüpfen und abzurufen.

1. SELECT-Klausel: Die Abfrage wählt den Titel und das Erscheinungsjahr jedes Films aus, in dem der Schauspieler mitgespielt hat. Zusätzlich wird eine Liste der Genres erstellt, die mit diesen Filmen verbunden sind.
2. GROUP_CONCAT-Funktion: Diese Funktion wird verwendet, um die Genres, denen der Film zugeordnet ist, als kommagetrennte Zeichenkette zusammenzufassen.
3. FROM-Klausel: Die Abfrage beginnt mit der Person-Tabelle, die Informationen über Personen enthält, einschließlich Schauspieler.
4. INNER JOIN spielt_in: Verbindet die Person-Tabelle mit der spielt_in-Tabelle, um die Filme zu finden, in denen die Personen mitgespielt haben.
5. INNER JOIN Film: Verbindet die spielt_in-Tabelle mit der Film-Tabelle, um Details zu den Filmen zu erhalten.
6. INNER JOIN ist_Genre: Verbindet die Film-Tabelle mit der ist_Genre-Tabelle, um die Genres der Filme zu identifizieren.
7. INNER JOIN Genre: Verbindet die ist_Genre-Tabelle mit der Genre-Tabelle, um die Namen der Genres zu erhalten.
8. WHERE-Klausel: Filtert die Ergebnisse nach einem bestimmten Schauspieler, dessen Name hier Schauspieler Name ist.
9. GROUP BY-Klausel: Gruppiert die Ergebnisse nach Film-ID, Titel und Erscheinungsjahr, um sicherzustellen, dass jeder Film nur einmal in der Liste erscheint.
10. ORDER BY-Klausel: Sortiert die Ergebnisse nach dem Erscheinungsjahr der Filme in aufsteigender Reihenfolge.

2.4.2 Als Kritiker möchte ich eine Übersicht über alle Filme in meinen Lieblingsgenres sehen, damit ich leicht entscheiden kann, welche Filme ich als Nächstes sehen möchte.

Akzeptanzkriterien:

- Die Sicht sollte Filme in meinen Lieblingsgenres anzeigen.
- Die Sicht sollte Titel, Durchschnittsbewertung und Bewertungsanzahl der Filme enthalten.

SQL:

```sql
CREATE VIEW FavoriteGenreFilms AS
SELECT
    f.Film_ID,
    f.Titel AS film_title,
    f.Bewertungsdurchschnitt AS average_rating,
    f.Bewertungsanzahl AS total_reviews,
    g.Name AS genre_name
FROM
    Film f
    INNER JOIN ist_Genre ig ON f.Film_ID = ig.Film_ID
    INNER JOIN Genre g ON ig.Genre_ID = g.Genre_ID
    INNER JOIN LieblingsGenre lg ON ig.Genre_ID = lg.Genre_ID
WHERE
    lg.Kritiker_ID = 3;
```

Erklärung:
Die Abfrage in 2.4.2 erstellt eine Sicht (View), die Filme in den Lieblingsgenres eines bestimmten Kritikers anzeigt, und zeigt dann den Inhalt dieser Sicht an. Am Ende wird die Sicht wieder gelöscht.

1. CREATE VIEW: Diese Anweisung erstellt eine Sicht (View) namens FavoriteGenreFilms, die wie eine gespeicherte Abfrage fungiert und die gleichen Filme in den Lieblingsgenres eines bestimmten Kritikers anzeigt.
2. SELECT-Klausel: Die Abfrage wählt die Film-ID, den Titel, die durchschnittliche Bewertung, die Anzahl der Bewertungen und das Genre jedes Films aus.
3. INNER JOIN ist_Genre: Verbindet die Film-Tabelle mit der ist_Genre-Tabelle, um die Genres der Filme zu identifizieren.
4. INNER JOIN Genre: Verbindet die ist_Genre-Tabelle mit der Genre-Tabelle, um die Namen der Genres zu erhalten.
5. INNER JOIN LieblingsGenre: Verbindet die ist_Genre-Tabelle mit der LieblingsGenre-Tabelle, um die Genres der Lieblingsgenres des Kritikers zu finden.
6. WHERE-Klausel: Filtert die Ergebnisse nach der Kritiker-ID, die hier auf 3 gesetzt ist, um die Lieblingsgenres dieses Kritikers zu identifizieren.
7. SELECT * FROM FavoriteGenreFilms;: Ruft alle Daten aus der erstellten Sicht FavoriteGenreFilms ab.
8. DROP VIEW FavoriteGenreFilms;: Löscht die Sicht nach der Verwendung, um Ressourcen freizugeben.

2.4.3. Als Kritiker möchte ich eine Liste der Regisseure sehen, deren Filme ich am höchsten bewertet habe, damit ich meine Präferenzen für bestimmte Regisseure besser verstehen kann.

Akzeptanzkriterien:

- Die Liste sollte die Regisseure enthalten, die in den Filmen, die der Kritiker bewertet hat, Regie geführt haben
- Die Durchschnittsbewertung der Filme jedes Regisseurs durch den Kritiker sollte angezeigt werden.
- Die Regisseure sollten nach der Durchschnittsbewertung sortiert sein, von der höchsten zur niedrigsten.

SQL:

```sql
SELECT
    p.Name AS regisseur_name,
    AVG(b.Bewertungszahl) AS avg_bewertung
FROM
    Bewertung b
    INNER JOIN Film f ON b.Film_ID = f.Film_ID
    INNER JOIN regie_in r ON f.Film_ID = r.Film_ID
    INNER JOIN Person p ON r.Person_ID = p.Person_ID
WHERE
    b.Kritiker_ID = 2
GROUP BY
    p.Person_ID, p.Name
ORDER BY
    avg_bewertung DESC;
```

Erklärung:

1. SELECT-Klausel: Die Abfrage wählt den Namen des Regisseurs und die durchschnittliche Bewertung der Filme, die dieser Regisseur gedreht hat.
2. AVG-Funktion: Berechnet die durchschnittliche Bewertung (avg_bewertung) der Filme, die von jedem Regisseur gedreht wurden.
3. FROM-Klausel: Die Abfrage beginnt mit der Bewertung-Tabelle, die die Bewertungen der Filme enthält.
4. INNER JOIN Film: Verbindet die Bewertung-Tabelle mit der Film-Tabelle, um die Film-Details zu erhalten.
5. INNER JOIN regie_in: Verbindet die Film-Tabelle mit der regie_in-Tabelle, um die Regisseure der Filme zu identifizieren.
6. INNER JOIN Person: Verbindet die regie_in-Tabelle mit der Person-Tabelle, um die Namen der Regisseure zu erhalten.
7. WHERE-Klausel: Filtert die Ergebnisse nach der Kritiker-ID, die hier auf 2 gesetzt ist, um die Bewertungen dieses Kritikers zu analysieren.
8. GROUP BY-Klausel: Gruppiert die Ergebnisse nach der Person-ID und dem Namen des Regisseurs, um sicherzustellen, dass jeder Regisseur nur einmal in der Liste erscheint.
9. ORDER BY-Klausel: Sortiert die Liste der Regisseure nach der durchschnittlichen Bewertung ihrer Filme in absteigender Reihenfolge.

2.4.4 Als Kritiker möchte ich eine Liste der Filme sehen, die von bestimmten Regisseuren geleitet und in meinen Lieblingsgenres sind, damit ich die besten Filme dieser Regisseure in meinen bevorzugten Genres entdecken kann.

Akzeptanzkriterien:

- Zeige eine Liste der Filme mit Titel, Regisseur und Genre.
- Filtern nach meinen Lieblingsgenres und bestimmten Regisseuren.

SQL:

```sql
SELECT
    f.Titel AS film_title,
    p.Name AS director_name,
    g.Name AS genre_name
FROM
    Film f
    INNER JOIN regie_in ri ON f.Film_ID = ri.Film_ID
    INNER JOIN Person p ON ri.Person_ID = p.Person_ID
    INNER JOIN ist_Genre ig ON f.Film_ID = ig.Film_ID
    INNER JOIN Genre g ON ig.Genre_ID = g.Genre_ID
    INNER JOIN LieblingsGenre lg ON g.Genre_ID = lg.Genre_ID
WHERE
    lg.Kritiker_ID = 3
    AND p.Name = 'Regisseur Name'
ORDER BY
    f.Bewertungsdurchschnitt;
```

Erklärung:

1. SELECT-Klausel: Die Abfrage wählt den Titel des Films, den Namen des Regisseurs und das Genre jedes Films aus.
2. FROM-Klausel: Die Abfrage beginnt mit der Film-Tabelle, die Details zu den Filmen enthält.
3. INNER JOIN regie_in: Verbindet die Film-Tabelle mit der regie_in-Tabelle, um die Regisseure der Filme zu identifizieren.
4. INNER JOIN Person: Verbindet die regie_in-Tabelle mit der Person-Tabelle, um die Namen der Regisseure zu erhalten.
5. INNER JOIN ist_Genre: Verbindet die Film-Tabelle mit der ist_Genre-Tabelle, um die Genres der Filme zu identifizieren.
6. INNER JOIN Genre: Verbindet die ist_Genre-Tabelle mit der Genre-Tabelle, um die Namen der Genres zu erhalten.
7. INNER JOIN LieblingsGenre: Verbindet die Genre-Tabelle mit der LieblingsGenre-Tabelle, um die Genres der Lieblingsgenres des Kritikers zu finden.
8. WHERE-Klausel: Filtert die Ergebnisse nach der Kritiker-ID, die hier auf 3 gesetzt ist, und nach dem Namen des Regisseurs, der hier Christopher Nolan ist.
9. ORDER BY-Klausel: Sortiert die Ergebnisse nach der durchschnittlichen Bewertung der Filme.

2.4.5 Als Systemadministrator möchte ich automatisch die Durchschnittsbewertung und die Anzahl der Bewertungen eines Films

aktualisieren, wenn eine neue Bewertung hinzugefügt oder eine bestehende Bewertung geändert wird, um die Filmdaten stets aktuell zu halten.

Akzeptanzkriterien:

Beim Einfügen oder Ändern einer Bewertung sollten die Durchschnittsbewertung und die Anzahl der Bewertungen des Films aktualisiert werden.

SQL:

```
DELIMITER //

CREATE TRIGGER update_film_rating
AFTER INSERT ON Bewertung
FOR EACH ROW
BEGIN
    DECLARE avg_rating DECIMAL(3, 1);
    DECLARE total_reviews INT;

    SELECT AVG(Bewertungszahl), COUNT(*)
    INTO avg_rating, total_reviews
    FROM Bewertung
    WHERE Film_ID = NEW.Film_ID;

    UPDATE Film
    SET Bewertungsdurchschnitt = avg_rating, Bewertungsanzahl =
total_reviews
    WHERE Film_ID = NEW.Film_ID;
END//

DELIMITER ;
```

Erklärung:

1. DELIMITER //: Ändert den Standardbefehlstrenner von ; auf //, damit der Trigger-Code, der selbst ; enthält, korrekt definiert werden kann.
2. CREATE TRIGGER update_film_rating: Erstellt einen Trigger mit dem Namen update_film_rating.
3. AFTER INSERT ON Bewertung: Der Trigger wird nach dem Einfügen eines neuen Datensatzes in die Bewertung-Tabelle ausgeführt.
4. FOR EACH ROW: Der Trigger wird für jede neue Zeile, die in die Bewertung-Tabelle eingefügt wird, ausgeführt.
5. BEGIN ... END: Umfasst den gesamten Codeblock, der vom Trigger ausgeführt wird.
6. DECLARE avg_rating DECIMAL(3, 1);: Deklariert eine Variable avg_rating zur Speicherung der durchschnittlichen Bewertung mit einer Genauigkeit von einer Dezimalstelle.
7. DECLARE total_reviews INT;: Deklariert eine Variable total_reviews zur Speicherung der Anzahl der Bewertungen als Ganzzahl.
8. SELECT AVG(Bewertungszahl), COUNT(*) INTO avg_rating, total_reviews FROM Bewertung WHERE Film_ID = NEW.Film_ID;: Berechnet die neue durchschnittliche Bewertung und die Gesamtanzahl der Bewertungen für den Film, der gerade bewertet wurde (NEW.Film_ID). Die Ergebnisse werden in den Variablen avg_rating und total_reviews gespeichert.
9. UPDATE Film SET Bewertungsdurchschnitt = avg_rating, Bewertungsanzahl = total_reviews WHERE Film_ID = NEW.Film_ID;: Aktualisiert die Film-Tabelle, um die Bewertungsdurchschnitt- und Bewertungsanzahl-Spalten des Films mit den neu berechneten Werten zu aktualisieren.
10. DELIMITER ;: Setzt den Standardbefehlstrenner wieder auf ; zurück.

3. relationale Algebra

3.1 Maximilian Sgodin

zu 2.1.1

Durchschnittliche Bewertung für einen bestimmten Film anzeigen:

γTitel,AVG(Bewertungszahl),COUNT(Bewertungszahl)(πTitel,Bewertungszahl(σf.Film_ID=[Film_ID](Film⋈Bewertung)))

Alle Bewertungen für einen bestimmten Film anzeigen:

τ−Verfassungsdatum(πKritikerName,Bewertungszahl,Kommentar,Verfassungsdatum(σb.Film_ID=[Film_ID](Bewertung)⋈b.Kritiker_ID=k.Kritiker_IDKritiker))

Kritikerprofil-Anzeigen:

τ−Verfassungsdatum(πTitel,Bewertungszahl,Kommentar,Verfassungsdatum(σb.Kritiker_ID=[Kritiker_ID](Bewertung)⋈b.Film_ID=f.Film_IDFilm))

zu 2.1.5

τ−Verfassungsdatum(πTitel,Bewertungszahl,Kommentar,Verfassungsdatum(σb.Kritiker_ID=[Kritiker_ID](Bewertung)⋈b.Film_ID=f.Film_IDFilm))

3.2 Denis Francesco Volpe

zu 2.2.4
Join-Operationen (darstellbar mit ⋈ oder σ…(Kreuzprodukt))

- R1 = σFilm.FilmID=Bewertung.FilmID(Film×Bewertung)
- R2 = σR1.FilmID=ist_Genre.FilmID(R1×ist_Genre)
- R3 = σR2.GenreID=Genre.GenreID(R2×Genre)

Selektion: filtern nach “WHERE Genre.Name =’Action’”

- R4=σGenre.Name=′Action′(R3)

Projektion: SELECT Film.Film_ID, Film.Titel, Genre.Name

- R5=πFilm.FilmID,Film.Titel,Genre.Name,Bewertung.Bewertungszahl(R4)

  Gruppierung und Aggregation : GROUP BY
  Film.FilmID,Film.Titel,Genre.Name **AVG(**Bewertung.Bewertungszahl) AS Durchschnittsbewertung, **COUNT(**Bewertung.Bewertungszahl) AS AnzahlBewertungen

- R6=γFilm.FilmID,Film.Titel,Genre.Name,AVG(Bewertung.Bewertungszahl)→Durchschnittsbewertung,COUNT(Bewertung.Bewertungszahl)→AnzahlBewertungen(R5)

  Selection nach HAVING:

- R7=σAnzahlBewertungen>0(R6)

  zu 2.2.1
  Aggregation : AVG(Bewertungszahl) AS Durchschnittsbewertung

- RKBD=γKritikerID,AVG(Bewertungszahl)→Durchschnittsbewertung(Bewertung)

  Join der Kritiker-Tabelle mit den aggregierten Daten :
  JOIN Kritiker_Bewertungsdurchschnitt kbd ON k.Kritiker_ID = kbd.Kritiker_ID

- RUpdate=Kritiker⋈Kritiker.KritikerID=RKBD.KritikerID RKBD

  Update und Setzen des Bewertungsstils :
  SET k.Bewertungsstil = CASE
  WHEN kbd.Durchschnittsbewertung <= 4 THEN 'Bad'
  WHEN kbd.Durchschnittsbewertung BETWEEN 5 AND 7 THEN 'Neutral'

- πKritikerID,Bewertungsstil→'Bad'(σDurchschnittsbewertung≤4(RUpdate))
- πKritikerID,Bewertungsstil→'Neutral'(σ5≤Durchschnittsbewertung≤7(RUpdate))
- πKritikerID,Bewertungsstil→'Good'(σDurchschnittsbewertung>7(RUpdate))

3.3 Stephane Sandevski

2.3.1 Gibt die Gage der Schauspielern aller Filmen aus

Join-Operation: als Kreuzprodukt ⋈

J = σp.Person_ID=g.Person_ID(Person ⋈ spielt_in)

Projektion: π

P = πp.name,g.Gage(J)

Aggregation:γ

A = γp.name,∑(g.Gage)→Gesamtgage(P)

Sortierung:

τGesamtgage DESC(A)

2.3.2: Alle Filme eines bestimmten Regisseurs anzeigen lassen

erste Abfrage:
Join-Operation:

J = σp.Person_ID=g.Person_ID(Person ⋈ regie_in)

Projektion:

P = πp.name(J)

Gruppierung:

γp.name(P)

zweite Abfrage:
Join-Operation:

J1 = σp.Person_ID=ri.Person_ID(Person ⋈ regie_in)
J2 = ri.Film_ID=f.Film_ID(J1 ⋈ Film)

Selektion:

S = σp.Name='ChristopherNolan'(J2)

Projektion:
P = πp.Name,f.Titel,f.Erscheinungsjahr(S)
Sortierung:
τf.ErscheinungsjahrDESC,p.Name,f.Titel(P)

3.4 Marius Ureche

2.4.1 Liste der Karrieren meiner Lieblingsschauspieler

1. Projektion (π):

πfilm_title,release_year,preferred_genres

Wählt die Spalten film_title, release_year und preferred_genres aus.

2. Aggregation und Gruppierung (γ):

γf.Film_ID,f.Titel,f.Erscheinungsjahr,preferred_genres

Gruppiert nach Film_ID, Titel und Erscheinungsjahr und aggregiert preferred_genres.

3. Selektion (σ):

σp.Name=’Schauspieler Name’

Filtert die Zeilen, in denen der Name der Person 'Schauspieler Name' ist.

4. Joins (⋈):

Person⋈p.Person_ID=si.Person_ID(spielt_in⋈si.Film_ID=f.Film_ID(Film⋈f.Film_ID=ig.Film_ID(ist_Genre⋈ ig.Genre_ID=g.Genre_IDGenre)))

Führt die erforderlichen Joins zwischen den Tabellen durch.

2.4.4 Liste der Filme sehen, die von bestimmten Regisseuren geleitet sind und in meinen Lieblingsgenres der Kritiker sind

πfilm_title,director_name,genre_name(σlg.Kritiker_ID=3∧p.Name=’Regiseour name’(((((Film⋈f.Film_ID=ri.Film_IDregie_in)⋈ri.Person_ID=p.Person_IDPerson)⋈f.Film_ID=ig.Film_IDist_Genre)⋈ig.Genre_ID=g.Genre_IDGenre)⋈g.Genre_ID=lg.Genre_IDLieblingsGenre))

Erläuterung:

1. Projektion (π):

πfilm_title,director_name,genre_name

Wählt die Spalten Titel (als film_title umbenannt), Name der Person (als director_name umbenannt) und Name des Genres (als genre_name umbenannt) aus.

2. Selektion (σ):

σlg.Kritiker_ID=3∧p.Name='Regiseour name'

Filtert die Zeilen, in denen die Kritiker_ID in der Tabelle LieblingsGenre gleich 3 ist und der Name der Person Regiseour name ist.

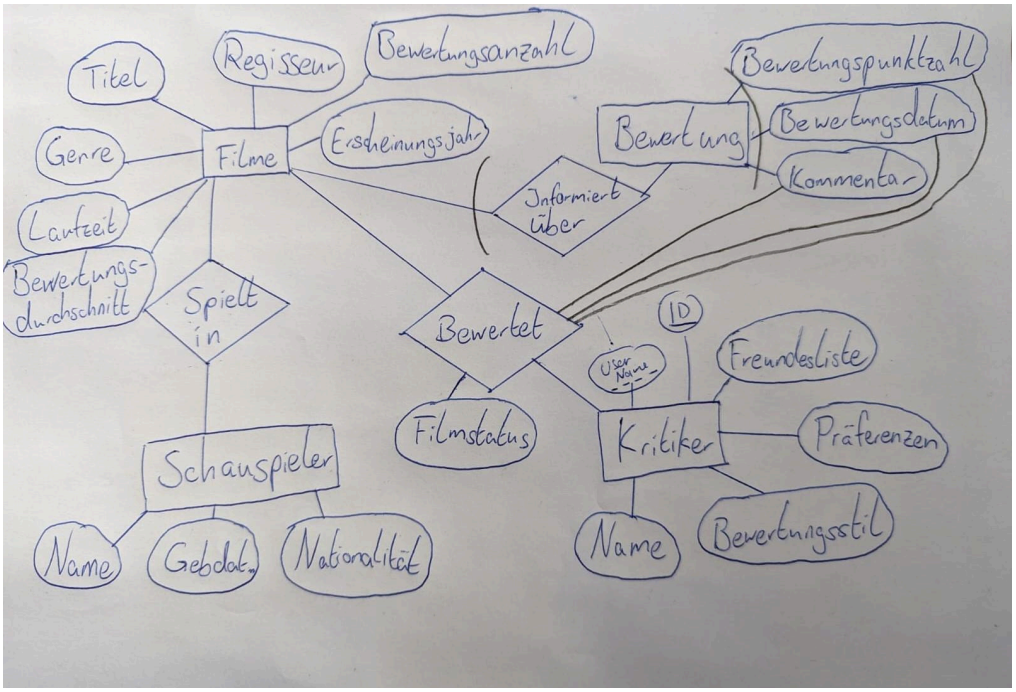
3. Joins (⋈):

((((Film⋈f.Film_ID=ri.Film_IDregie_in)⋈ri.Person_ID=p.Person_IDPerson)⋈ f.Film_ID=ig.Film_IDist_Genre)⋈ig.Genre_ID=g.Genre_IDGenre)⋈g.Genre_ID=lg.Genre_IDLieblingsGenre

Führt die erforderlichen Verknüpfungen (Joins) zwischen den Tabellen durch.

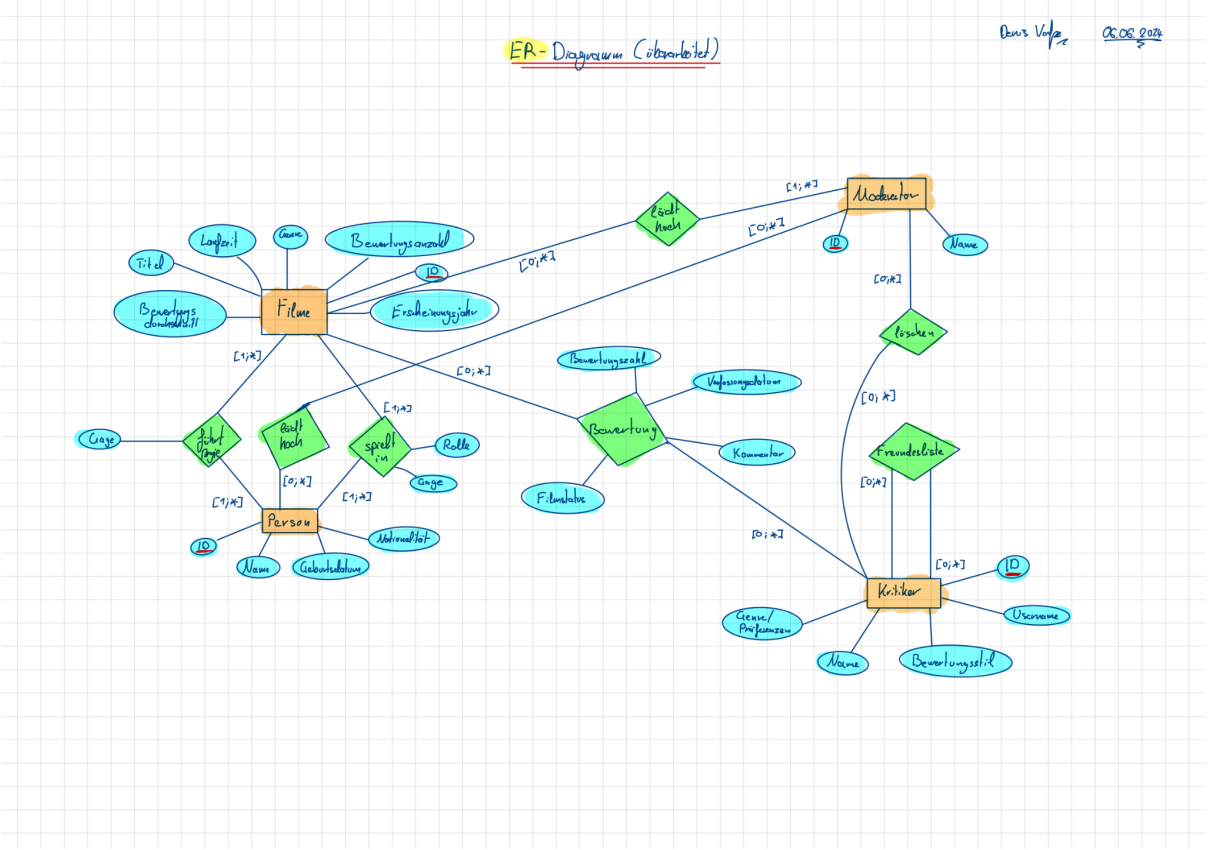
4. ER-Diagramm mit Iterationen

4.1 ER-Diagramm der ersten Iteration



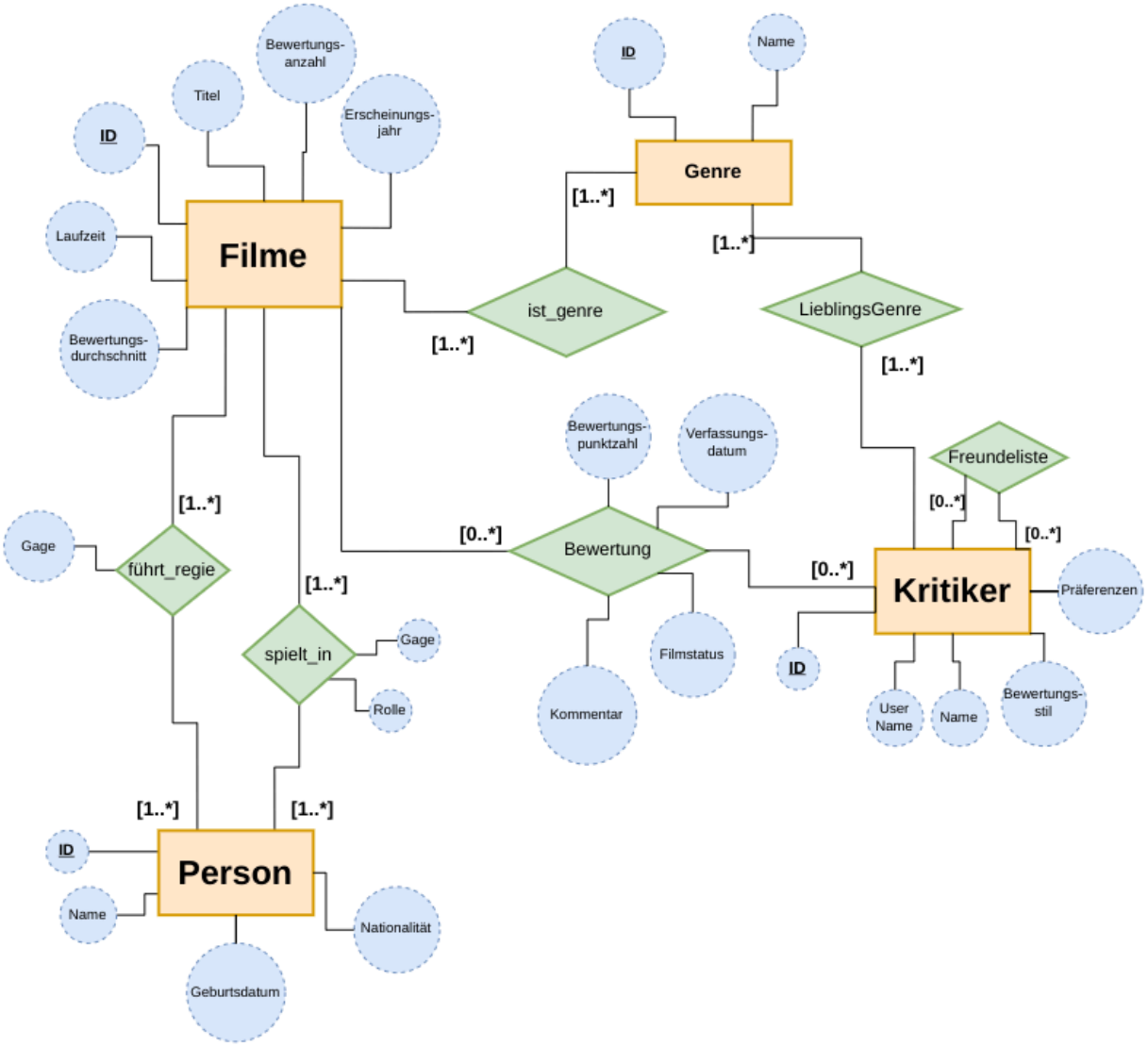
In der ersten Iteration haben wir die Hauptkomponenten identifiziert und ihre grundlegenden Beziehungen definiert. Mit dem ersten Diagramm haben wir aber nicht allzu lang gearbeitet und haben es stetig weiterentwickelt.

4.2 ER-Diagramm der zweiten Iteration



Bei der zweiten Iteration kam neben den neuen Relationen zwischen Film und Person noch der Moderator dazu. Bei dem Moderator haben wir uns auch wieder sehr schnell dagegen entschieden, da er quasi die Rolle des Administrators/Nutzers der Datenbank darstellt.

4.3 ER-Diagramm der dritten Iteration



Durch erhaltenes Feedback sind wir nach der Überarbeitung zu diesem Diagramm der dritten Iteration gekommen, bei dem die Entität Genre noch dazukam. Das Diagramm zeigt die verschiedenen Entitäten und Beziehungen, die aus den Anforderungen der User Stories abgeleitet wurden. Die Hauptentitäten sind "Film", "Person", "Kritiker", "Genre" und "Bewertung".

Die Entität "Film" umfasst Attribute wie Film_ID, Titel, Laufzeit, Bewertungsdurchschnitt, Bewertungsanzahl und Erscheinungsjahr, was auf die Anforderungen hinweist, detaillierte Informationen über Filme und deren Bewertungen zu speichern (User Story 2.1.2). "Person" umfasst Person_ID, Name, Geburtsdatum und Nationalität, um Schauspieler und Regisseure zu identifizieren (User Story 2.1.2).

Die Entität "Kritiker" enthält Kritiker_ID, UserName, Name und Bewertungsstil. Dies ermöglicht die Verwaltung der Kritikerprofile und deren Bewertungen (User Story 2.1.1 und 2.2.1). "Genre" enthält Genre_ID und Name, um die verschiedenen Filmgenres zu

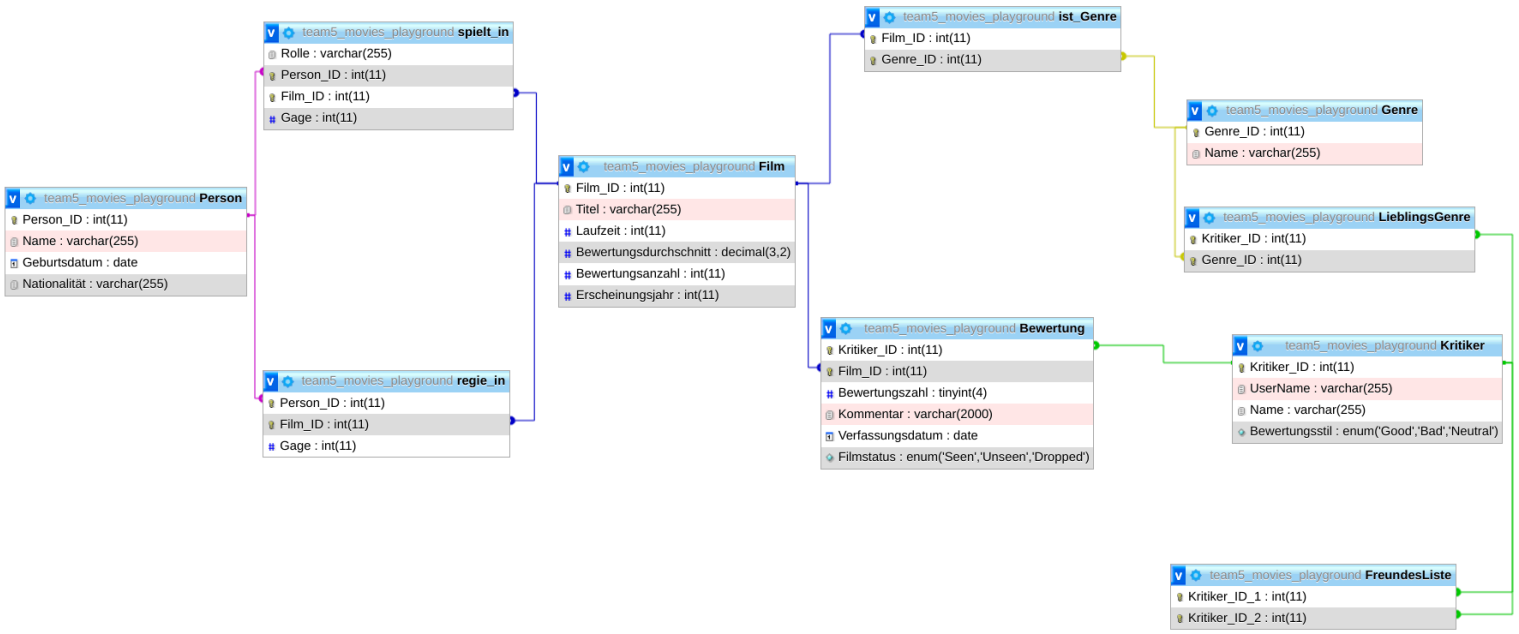
kategorisieren (User Story 2.1.3). "Bewertung" umfasst Kritiker_ID, Film_ID, Bewertungszahl, Kommentar, Verfassungsdatum und Filmstatus, was es ermöglicht, detaillierte Bewertungen und Kommentare zu speichern (User Story 2.1.1 und 2.2.2).

Die Beziehungen im Diagramm zeigen, wie diese Entitäten miteinander verbunden sind. Die Beziehung "führt_regie" zwischen "Film" und "Person" ermöglicht die Verknüpfung von Filmen mit ihren Regisseuren (User Story 2.1.2), während "spielt_in" die Schauspieler in den Filmen verknüpft (User Story 2.1.2). Die Beziehung "ist_genre" verbindet Filme mit ihren Genres (User Story 2.1.3). Die Beziehung "Bewertung" verknüpft Kritiker und Filme, um Bewertungen zu speichern (User Story 2.1.1). "LieblingsGenre" verbindet Kritiker mit ihren bevorzugten Genres (User Story 2.4.1), und die selbstreferenzielle Beziehung "FreundesListe" ermöglicht es, Freundschaften zwischen Kritikern darzustellen (User Story 2.4.1).

Die Kardinalitäten wurden so gewählt, dass sie die Realität der Filmindustrie und die Anforderungen der Benutzer widerspiegeln. Beispielsweise kann ein Film mehrere Schauspieler und Regisseure haben, und ein Kritiker kann mehrere Filme bewerten. Diese Design-Entscheidungen stellen sicher, dass die Datenbank effizient und flexibel ist, um die Anforderungen der Filmkritik-Website zu erfüllen.

Die Entitäten und Beziehungen wurden so gewählt, dass sie die Anforderungen der User Stories vollständig abdecken und die Datenbank flexibel und effizient gestalten. Beispielsweise ermöglicht die Beziehung `Bewertung` die Speicherung detaillierter Kritiken und Kommentare zu Filmen, während die selbstreferentielle Beziehung `FreundesListe` die sozialen Interaktionen zwischen den Kritikern unterstützt. Die Kardinalitäten wurden sorgfältig festgelegt, um die tatsächlichen Szenarien der Filmindustrie und die Anforderungen der Benutzer und Kritiker zu reflektieren.

5. Relationales Schema



Aus dem ER-Diagramm haben wir die Entitäten "Film", "Person", "Kritiker", "Genre" und "Bewertung" in relationale Tabellen überführt. Dabei haben wir die Attribute der Entitäten als Spalten in den entsprechenden Tabellen definiert.

Abweichungen und Verfeinerungen

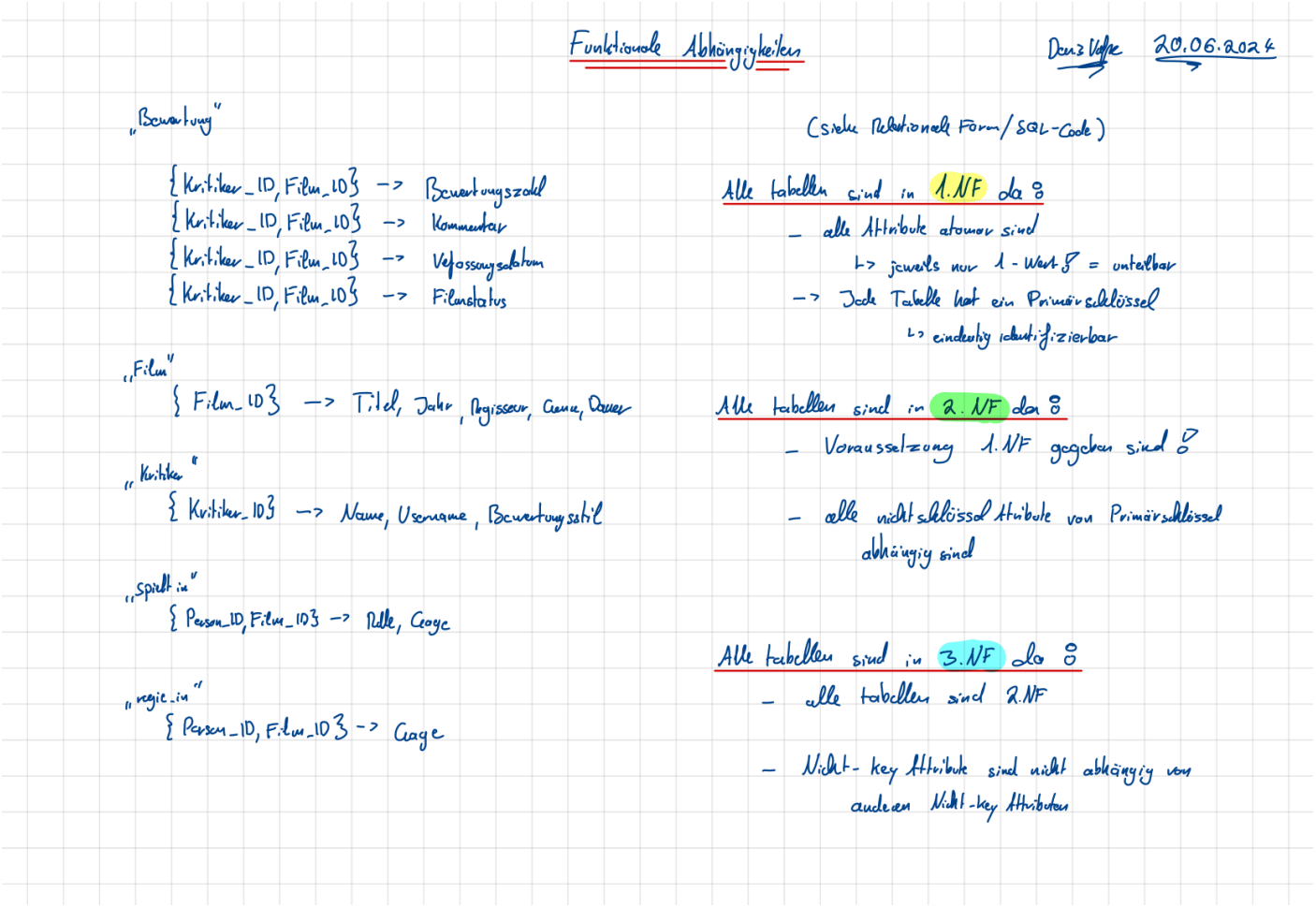
1. **Kardinalitäten und Fremdschlüssel**:
    - Jede Beziehung im ER-Diagramm wurde durch Fremdschlüssel in den entsprechenden Tabellen realisiert. Zum Beispiel wurde die Beziehung "führt_regie" zwischen "Person" und "Film" durch die Fremdschlüssel in der Tabelle `regie_in` umgesetzt.
    - Die Kardinalitäten (1
      , n:1) wurden durch die Definition der Fremdschlüssel und Primary Keys sichergestellt.

2. **Verfeinerungen**:
    - Die Beziehung "Bewertung" enthält zusätzliche Attribute wie `Filmstatus`, um den Status der Filme für die Kritiker zu verfolgen (User Story 2.2.2).
    - In der Tabelle `spielt_in` haben wir das Attribut `Rolle` hinzugefügt, um die spezifischen Rollen der Schauspieler in den Filmen zu erfassen (User Story 2.1.2).

3. **Referentielle Integrität**:
    - Jede Fremdschlüsseldefinition stellt sicher, dass referentielle Integrität gewahrt bleibt. Zum Beispiel, in der Tabelle `Bewertung` stellt der Fremdschlüssel `Film_ID` sicher, dass jede Bewertung einem existierenden Film zugeordnet ist.

4. **Semantische Bedingungen**:
    - Einschränkungen wie `NOT NULL` und `DEFAULT` wurden verwendet, um sicherzustellen, dass notwendige Daten vorhanden sind und Standardwerte gesetzt werden, um Datenkonsistenz zu gewährleisten. Beispielsweise hat das Attribut `Bewertungsstil` in der Tabelle `Kritiker` einen Standardwert `Neutral` (User Story 2.2.1).

6. Normalisierung



Bei der Erstellung unseres relationalen Schemas haben wir darauf geachtet, dass alle Tabellen mindestens der dritten Normalform (3NF) entsprechen. Der Prozess der Normalisierung ist entscheidend, um Redundanzen zu vermeiden und die Integrität der Daten zu gewährleisten. Im Folgenden beschreiben wir, wie wir die Tabellen normalisiert und welche funktionalen Abhängigkeiten wir dabei berücksichtigt haben.

Erste Normalform (1NF)

Die erste Normalform stellt sicher, dass alle Attributwerte atomar sind, also keine Listen oder Mengen in einer Zelle existieren. Alle Tabellen in unserem Schema erfüllen diese Bedingung, da jedes Attribut nur einen einzelnen Wert pro Datensatz enthält.

Beispiel:

- Die Tabelle `Film` enthält Attribute wie `Titel`, `Laufzeit`, und `Erscheinungsjahr`, die jeweils nur einen einzelnen Wert pro Datensatz speichern.

Zweite Normalform (2NF)

Die zweite Normalform verlangt, dass alle Nicht-Schlüsselattribute voll funktional abhängig vom Primärschlüssel sind. Dies bedeutet, dass jedes Nicht-Schlüsselattribut von dem gesamten Primärschlüssel abhängen muss und nicht nur von einem Teil davon.

Beispiel:

- In der Tabelle `Bewertung` ist der Primärschlüssel die Kombination aus `Kritiker_ID` und `Film_ID`. Alle anderen Attribute (`Bewertungszahl`, `Kommentar`, `Verfassungsdatum`, `Filmstatus`) hängen vollständig von dieser Kombination ab.

Dritte Normalform (3NF)

Die dritte Normalform stellt sicher, dass keine transitive Abhängigkeit zwischen den Attributen existiert. Das bedeutet, dass jedes Nicht-Schlüsselattribut direkt vom Primärschlüssel abhängt und nicht von einem anderen Nicht-Schlüsselattribut.

Beispiel:

- In der Tabelle `Person` hängt `Name`, `Geburtsdatum` und `Nationalität` direkt von `Person_ID` ab.
- In der Tabelle `Kritiker` hängen `UserName`, `Name` und `Bewertungsstil` direkt von `Kritiker_ID` ab.

Überprüfung der Funktionalen Abhängigkeiten

1. **Film**:
    - Funktionale Abhängigkeiten: `Film_ID -> Titel, Laufzeit, Bewertungsdurchschnitt, Bewertungsanzahl, Erscheinungsjahr`
    - Jeder Film wird eindeutig durch seine `Film_ID` identifiziert, und alle anderen Attribute hängen direkt davon ab.

2. **Person**:
    - Funktionale Abhängigkeiten: `Person_ID -> Name, Geburtsdatum, Nationalität`
    - Die `Person_ID` identifiziert eindeutig jede Person und deren Attribute.

3. **Kritiker**:
    - Funktionale Abhängigkeiten: `Kritiker_ID -> UserName, Name, Bewertungsstil`
    - Die `Kritiker_ID` identifiziert eindeutig jeden Kritiker und deren Attribute.

4. **Genre**:
    - Funktionale Abhängigkeiten: `Genre_ID -> Name`
    - Das `Genre_ID` identifiziert eindeutig jedes Genre.

5. **Bewertung**:
    - Funktionale Abhängigkeiten: `(Kritiker_ID, Film_ID) -> Bewertungszahl, Kommentar, Verfassungsdatum, Filmstatus`
    - Die Kombination aus `Kritiker_ID` und `Film_ID` identifiziert eindeutig jede Bewertung und deren Attribute.

6. **FreundesListe**:
    - Funktionale Abhängigkeiten: `(Kritiker_ID_1, Kritiker_ID_2) -> {}`
    - Die Kombination aus `Kritiker_ID_1` und `Kritiker_ID_2` identifiziert eindeutig jede Freundschaftsbeziehung.

7. **LieblingsGenre**:
    - Funktionale Abhängigkeiten: `(Kritiker_ID, Genre_ID) -> {}`
    - Die Kombination aus `Kritiker_ID` und `Genre_ID` identifiziert eindeutig jede Beziehung zwischen Kritiker und ihrem Lieblingsgenre.

8. **ist_Genre**:
    - Funktionale Abhängigkeiten: `(Film_ID, Genre_ID) -> {}`
    - Die Kombination aus `Film_ID` und `Genre_ID` identifiziert eindeutig die Zuordnung von Filmen zu Genres.

9. **regie_in**:
    - Funktionale Abhängigkeiten: `(Person_ID, Film_ID) -> Gage`
    - Die Kombination aus `Person_ID` und `Film_ID` identifiziert eindeutig jede Regiebeziehung und die entsprechende Gage.

10. **spielt_in**:
    - Funktionale Abhängigkeiten: `(Person_ID, Film_ID) -> Rolle, Gage`
    - Die Kombination aus `Person_ID` und `Film_ID` identifiziert eindeutig jede Schauspielbeziehung, die Rolle und die Gage.

7. Implementierung

Die Implementierung des relationalen Schemas erfolgt durch die Erstellung der notwendigen Tabellen unter Berücksichtigung der Datentypen, Primärschlüssel, Fremdschlüssel und anderen Bedingungen.

SQL CODE:

"Create FILM table ":

```
CREATE TABLE Film (
    Film_ID INT AUTO_INCREMENT PRIMARY KEY,
    Titel VARCHAR(255) NOT NULL,
    Laufzeit INT,
    Bewertungsdurchschnitt DECIMAL(3, 2),
    Bewertungsanzahl INT,
    Erscheinungsjahr INT,
)
```

"Create FreundesListe table " :

```
CREATE TABLE FreundesListe (
    Kritiker_ID_1 INT,
    Kritiker_ID_2 INT,
    PRIMARY KEY (Kritiker_ID_1, Kritiker_ID_2),
    FOREIGN KEY (Kritiker_ID_1) REFERENCES Kritiker(Kritiker_ID),
    FOREIGN KEY (Kritiker_ID_2) REFERENCES Kritiker(Kritiker_ID)
)
```

```
   "Create Genre table" :
   CREATE TABLE Genre (
        Genre_ID INT AUTO_INCREMENT PRIMARY KEY,
        Name VARCHAR(255) NOT NULL
)



   "Create Kritiker table " :
   CREATE TABLE Kritiker (
        Kritiker_ID INT AUTO_INCREMENT PRIMARY KEY,
        UserName VARCHAR(255) NOT NULL,
        Name VARCHAR(255) NOT NULL,,
        Bewertungsstil ENUM('Good', 'Bad', 'Neutral') DEFAULT 'Neutral'
)



   "Create Person table " :
   CREATE TABLE Person (
   Person_ID INT AUTO_INCREMENT PRIMARY KEY,
   Name VARCHAR(255) NOT NULL,
   Geburtsdatum DATE,
   Nationalität VARCHAR(255) NOT NULL,
)



   "Create regie_in table " :
   CREATE TABLE regie_in (
        Person_ID INT,
        Film_ID INT,
        Gage INT,
        PRIMARY KEY (Person_ID, Film_ID),
        FOREIGN KEY (Person_ID) REFERENCES Person(Person_ID),
        FOREIGN KEY (Film_ID) REFERENCES Film(Film_ID)
   UNIQUE KEY (Person_ID, Film_ID)
)



   "Create spielt_in table " :
   CREATE TABLE PersonFilm (
        Person_ID INT,
        Film_ID INT,
        Gage INT
```

```
    PRIMARY KEY(`Person_ID`, `Film_ID`);
        FOREIGN KEY (Person_ID) REFERENCES Person(Person_ID),
        FOREIGN KEY (Film_ID) REFERENCES Film(Film_ID)
)



    "Create Bewertung table " :
    CREATE TABLE Bewertung (
        Kritiker_ID INT,
        Film_ID INT,
        Bewertungszahl TINYINT,
        Kommentar VARCHAR(2000),
        Verfassungsdatum DATE,
    Filmstatus ENUM('Seen', 'Unseen', 'Dropped') DEFAULT 'Unseen',
    PRIMARY KEY(`Kritiker_ID`, `Film_ID`);
        FOREIGN KEY (Kritiker_ID) REFERENCES Kritiker(Kritiker_ID),
        FOREIGN KEY (Film_ID) REFERENCES Film(Film_ID),
        UNIQUE KEY (Kritiker_ID, Film_ID)
)


    "Create LieblingsGenre table" :
    CREATE TABLE LieblingsGenre (
        Kritiker_ID INT,
        Genre_ID INT,
        PRIMARY KEY (Genre_ID, Kritiker_ID),
        FOREIGN KEY (Genre_ID) REFERENCES Genre(Genre_ID),
        FOREIGN KEY (Kritiker_ID) REFERENCES Kritiker(Kritiker_ID),
    UNIQUE KEY (Genre_ID, Kritiker_ID)
)


    "Create ist_Genre table" :
    CREATE TABLE ist_Genre(
        Film_ID INT,
        Genre_ID INT,
        PRIMARY KEY (Genre_ID, Film_ID),
        FOREIGN KEY (Genre_ID) REFERENCES Genre(Genre_ID),
        FOREIGN KEY (Film_ID) REFERENCES Film(Film_ID)
    UNIQUE KEY (Film_ID, Genre_ID)
)
```

8. Reflexion

Arbeitsverteilung

Maximilian Sgodin:
- Erstellung des ER-Diagramms der ersten Iteration
- Erstellung eines Teilbereichs des relationalen Datenbankmodells (Kritiker-Film-Relation) der ersten Iteration
- Befüllung der Tabellen der ersten Iteration
- Verfassung des Berichts

Denis Francesco Volpe:
- Erstellung der ER-Diagramme der zweiten und dritten Iteration
- Erstellung eines Teilbereichs des relationalen Datenbankmodells (Regisseur-Film-Relation) der ersten Iteration
- Erstellung des relationalen Datenbankmodells der dritten Iteration und Normalisierung
- Implementierung der dritten Iteration
- Befüllung der Tabellen der dritten Iteration

Stephane Sandevski:
- Befüllung der Tabellen der ersten Iteration
- Erstellung eines relationalen Datenbankmodells in der ersten Iteration
- Teil-Implementierung der ersten Iteration

Marius Ureche:
- Digitalisierung des ER-Diagramms in 1 und 3 Iteration
- Erstellung des Relationalen Schemas
- Erstellung eines Teilbereichs des relationalen Datenbankmodells (Schauspieler-Film-Relation) der ersten Iteration
- Implementierung der ersten Iteration

Design-Entscheidungen und Alternativen

In unserem Projekt haben wir mehrere entscheidende Design-Entscheidungen getroffen, um die Anforderungen der User Stories zu erfüllen und eine effiziente Datenbankstruktur zu schaffen. Eine der wichtigsten Entscheidungen betraf die Wahl der Entitäten und deren Attribute. Wir haben uns entschieden, separate Entitäten für "Film", "Person", "Kritiker", "Genre" und "Bewertung" zu erstellen, um eine klare Trennung der verschiedenen Datenarten zu gewährleisten. Alternativ hätten wir eine vereinfachte Struktur mit weniger Tabellen wählen können, was jedoch zu Redundanzen und Komplexität bei den Anfragen geführt hätte.

Eine weitere bedeutende Entscheidung war die Implementierung von Fremdschlüsseln zur Gewährleistung der referentiellen Integrität. Dadurch konnten wir sicherstellen, dass Verbindungen zwischen den Tabellen korrekt sind und keine verwaisten Datensätze entstehen. Eine Alternative wäre die Verwendung von Triggern zur Datenintegrität gewesen, was jedoch die Komplexität und die Wartung der Datenbank erhöht hätte.

Rückblick und zukünftige Entscheidungen

Rückblickend würden wir viele unserer Design-Entscheidungen wieder so treffen. Die gewählte Struktur hat sich als effektiv und robust erwiesen, insbesondere im Hinblick auf die Erfüllung der Anforderungen aus den User Stories. Allerdings haben wir gelernt, dass eine sorgfältige Planung und Prüfung der Anforderungen im Vorfeld entscheidend ist, um spätere Anpassungen und Änderungen zu minimieren. Ein Beispiel hierfür ist die Beziehung zwischen "Person" und "Film" für Schauspieler und Regisseure, die klar definiert und einfach zu erweitern sind.

Anpassbarkeit des Schemas

Unser Schema hat sich als gut anpassbar erwiesen. Beispielsweise könnten neue Attribute oder Tabellen problemlos hinzugefügt werden, um zusätzliche Informationen zu speichern, wie etwa Produktionsfirmen oder Drehorte. Auch neue Anfragen, wie die Ermittlung von Filmen basierend auf spezifischen Kritikerbewertungen oder die Erweiterung der Freundesnetzwerke der Kritiker, können ohne größere Anpassungen implementiert werden. Dies zeigt die Flexibilität und Erweiterbarkeit unseres Schemas.

Geschickte und ungeschickte Aspekte

Besonders geschickt hat sich unser Schema im Umgang mit Bewertungen gezeigt. Die Trennung der Bewertungstabellen und die klare Definition der Beziehungen ermöglichen es, detaillierte und flexible Anfragen zu erstellen, ohne die Performance der Datenbank zu beeinträchtigen. Ein ungeschickter Aspekt war jedoch die initiale Implementierung der Freundesliste. Die selbstreferentielle Beziehung zwischen Kritikern erwies sich als komplexer in der Handhabung und erforderte zusätzliche Überlegungen, um eine effiziente Abfrage zu gewährleisten.

Komplexität und Einfachheit

Überraschenderweise war die Implementierung der Beziehung "spielt_in" zwischen "Film" und "Person" einfacher als erwartet. Dies liegt daran, dass die Verknüpfung von Schauspielern und Filmen eine klare und einfache Struktur hat. Im Gegensatz dazu erwies sich die Verwaltung der Lieblingsgenres der Kritiker als komplexer, da dies dynamische Daten sind, die sich basierend auf den Bewertungen ändern können.

Haupterkenntnisse

Unsere Haupterkenntnisse aus diesem Projekt umfassen die Bedeutung einer klaren Strukturierung der Datenbank und die sorgfältige Planung der Beziehungen zwischen den Entitäten. Die Verwendung von Fremdschlüsseln und die Berücksichtigung von Normalisierungsprinzipien haben sich als essentiell erwiesen, um eine effiziente und

wartbare Datenbank zu schaffen. Zudem haben wir gelernt, dass Flexibilität und Erweiterbarkeit entscheidend sind, um zukünftige Anforderungen und Anpassungen problemlos umsetzen zu können.

Diese Reflexion bietet einen umfassenden Überblick über die Design-Entscheidungen, Herausforderungen und Erkenntnisse, die wir während der Modellierung und Implementierung unserer Filmkritik-Datenbank gewonnen haben.