Data Engineering Projekt 1

Ilef Kalboussi (81774)
Nina Scheiffele (87488)
Kevin Garrison (85826)

17 November 2024

1 Modelldefinition

Dieses Datenbankmodell unterstützt eine Streaming-Anwendung (Netflix) für Medien, die es den Nutzern ermöglicht, Watchlists zu erstellen und Medieninhalte zu bewerten, mit einer strukturierten Verwaltung von Abonnements und Medientypen. Hier eine Übersicht über die Hauptentitäten und ihre Beziehungen:

1. **User:** User werden in Main User und Other User unterteilt. Hauptnutzer sind die primären Kontoinhaber und können Abonnements abschließen, während Weitere Nutzer sekundäre Nutzer desselben Kontos mit eingeschränkter Funktionalität sein können.
2. **Subscription:** Eine Subscription ist an einen Hauptnutzer gebunden und enthält Attribute wie IBAN, Preis, Abonnementtyp (monatlich, jährlich, Test) sowie Start- und Enddatum.
3. **Watchlist:** Jeder Nutzer hat eine Watchlist, die Medien (Movies oder Series) enthält, an denen er interessiert ist. Die Zuordnungstabelle watchlist_media unterstützt die viele-zu-viele-Beziehung zwischen Watchlist und Media.
4. **Media:** Dies umfasst sowohl Movies als auch Series, die jeweils spezifische Attribute haben. Movies besitzen eine Dauer, während Series eine Anzahl der Staffeln und mehrere Episodes haben. Medias können von Nutzern durch Reviews bewertet werden.
5. **Cast:** Dies umfasst die Rollen Actor und Director, die entweder einem Movie oder einer Series zugeordnet ist. Die CheckConstraint auf Cast stellt sicher, dass ein Cast im gesamten nur zu einem Medientyp gehört.
6. **Reviews:** Nutzer können Reviews für jedes Media Objekt abgeben und dadurch eine Bewertung für dieses Medium hinterlassen.

2 Beschreibung des Use Cases

Die Anwendung ermöglicht es Hauptnutzern, ein Konto zu erstellen und ein Abonnement abzuschließen. Nutzer können Medien zu ihrer Watchlist hinzufügen, diese bewerten und Details zur Besetzung einsehen. Verschiedene Medien (Filme und Serien) haben unterschiedliche Strukturen, und zur Besetzung gehören sowohl Schauspieler als auch Regisseure, die spezifischen Medien zugeordnet sind. Dieses Modell unterstützt somit eine umfassende Medien-Streaming-Dienststruktur, mit Flexibilität in der Verwaltung von Medien, Nutzern, Abonnements und deren Beziehungen untereinander.

3 Beschreibung der Queries

Hier ist eine detaillierte Beschreibung der Queries in der Klasse Queries:

1. **Count Media per User**:
   - **Query:**

```
def count_media_per_user(self, session):
    """Counts the number of media items in each user's watchlist."""
    result = session.query(
        User.username,
        func.count(Media.id).label('media_count')
    ).join(Watchlist, User.id == Watchlist.user_id) \
    .join(watchlist_media, Watchlist.id == watchlist_media.c.watchlists_id) \
    .join(Media, Media.id == watchlist_media.c.medias_id) \
    .group_by(User.id) \
    .all()

    for username, media_count in result:
        print(f"Username: {username}, Media Count: {media_count}")
```

   - **Beschreibung:** Diese Methode zählt die Anzahl von Medienelementen (Filme, Serien usw.), die sich in der Watchlist jedes Benutzers befinden.
   - **Funktionsweise:** Die Query verbindet die Tabellen User, Watchlist, *watchlist_media* und Media, um die Beziehungen zwischen Benutzern und den Medienelementen in ihren Watchlists herzustellen. Mithilfe von *group_by* wird die Zählung der Medienelemente pro Benutzer erstellt. Die Ergebnisse geben für jeden Benutzer den Namen und die Anzahl der Medienelemente zurück.
   - **Beispielausgabe:**

```
Username: john_doe,
Media Count: 5
John Doe hat fünf Medienelemente in seine Watchlist.
```

2. **Average Rating by Genre:**
   - **Query:**

```
def average_rating_by_genre(self, session):
    """Calculates the average rating of movies and series by genre."""
    result = session.query(
        Media.genre,
        func.avg(Media.rating).label('average_rating')
    ).group_by(Media.genre) \
    .all()

    for genre, avg_rating in result:
        print(f"Genre: {genre}, Average Rating: {avg_rating:.2f}")
```

   - **Beschreibung:** Diese Methode berechnet die durchschnittliche Bewertung von Medien nach Genre.
   - **Funktionsweise:** Sie gruppiert die Medien (Media) nach Genre und berechnet für jedes Genre den Durchschnittswert der Bewertungen (rating). Nur Medienelemente mit einer Bewertung werden berücksichtigt.
   - **Beispielausgabe:**

```
Genre: Action,
Average Rating: 8.23
Die durchschnittliche Bewertung für Action-Medien beträgt 8,23.
```

3. **Count Reviews per Media:**

- **Query:**

```
def count_reviews_per_media(self, session):
    """Counts the number of reviews per media item."""
    result = session.query(
        Media.title,
        func.count(Review.id).label('review_count')
    ).join(Review, Media.id == Review.media_id) \
    .group_by(Media.id) \
    .all()

    for title, review_count in result:
        print(f"Media Title: {title}, Review Count: {review_count}")
```

- **Beschreibung:** Diese Methode zählt die Anzahl der Bewertungen (Review) für jedes Medienelement.
- **Funktionsweise:** Die Query verbindet die Tabellen Media und Review, um die Anzahl der Bewertungen pro Medienelement zu bestimmen. Mithilfe von *group_by* wird die Zählung für jedes Medienelement erstellt.
- **Beispielausgabe:**

```
Media Title: Inception,
Review Count: 150
Der Film "Inception" hat 150 Bewertungen.
```

4. **Total revenue by Subscription Type:**

- **Query:**

```
def total_revenue_by_subscription_type(self, session):
    """Calculates total revenue by subscription type."""
    result = session.query(
        Subscription.subscription_type,
        func.sum(Subscription.price).label('total_revenue')
    ).group_by(Subscription.subscription_type) \
    .all()

    for sub_type, total_revenue in result:
        print(f"Subscription Type: {sub_type}, Total Revenue:
--------${total_revenue:.2f}")
```

- **Beschreibung:** Diese Methode berechnet die Gesamteinnahmen für jeden Abonnementtyp.
- **Funktionsweise:** Sie gruppiert die Abonnements (Subscription) nach dem Abonnementtyp und berechnet die Summe der Preise für jedes Abonnement. Dies gibt den Gesamtumsatz pro Abonnementtyp an.
- **Beispielausgabe:**

```
Subscription Type: Premium,
Total Revenue: $1200.00
Der Premium-Abonnementtyp hat Gesamteinnahmen von $1200,00 erzielt.
```

5. **Count episodes per Series:**

   - **Query:**

```
def count_episodes_per_series(self, session):
    """Counts the number of episodes per series."""
    result = session.query(
        Series.title,
        func.count(Episode.id).label('episode_count')
    ).join(Episode, Series.id == Episode.series_id) \
    .group_by(Series.id) \
    .all()

    for title, episode_count in result:
        print(f"Series Title: {title}, Episode Count: {episode_count}")
```

   - **Beschreibung:** Diese Methode zählt die Anzahl der Episoden für jede Serie.
   - **Funktionsweise:** Die Query verbindet die Tabellen Series und Episode, um die Anzahl der Episoden pro Serie zu berechnen. Mithilfe von *group_by* wird die Zählung der Episoden pro Serie erstellt.
   - **Beispielausgabe:**

```
Series Title: Breaking Bad,
Episode Count: 62
Die Serie "Breaking Bad" hat insgesamt 62 Episoden.
```