deti universidade de aveiro
departamento de electrónica, telecomunicações e informática

MOVIE ADVISOR

Base de Dados – Projeto final

LEI – 28/05/2024 - P4

David Amorim, 112610

Francisca Silva, 112841

CONCEITO

Uma base de dados para uma plataforma de reviews de filmes e séries, que permite aos utilizadores deixar uma crítica, ver críticas ou procurar algo para ver, aplicando filtros com base nos géneros, entre outros. Permite também ver os detalhes de filmes/séries tais como ano de lançamento, orçamento, sinopse, descrição, etc.

REQUISITOS

- Criar filmes, séries, temporadas e episódios
- Registar-se na plataforma
- Ver informações de um filme
- Deixar uma crítica ao filme
- Ver críticas de outros utilizadores
- Dar like em críticas de outros utilizadores
- Procurar filmes com base em filtros
- Criar listas pessoais (watchlists)*
- Ver listas criadas por outros utilizadores**
- Adicionar filme a uma lista pessoal

Notas: * As watchlists podem ser públicas ou privadas, sendo que
** apenas podem ser visualizadas por outros utilizadores as watchlists públicas.

DIAGRAMA DER

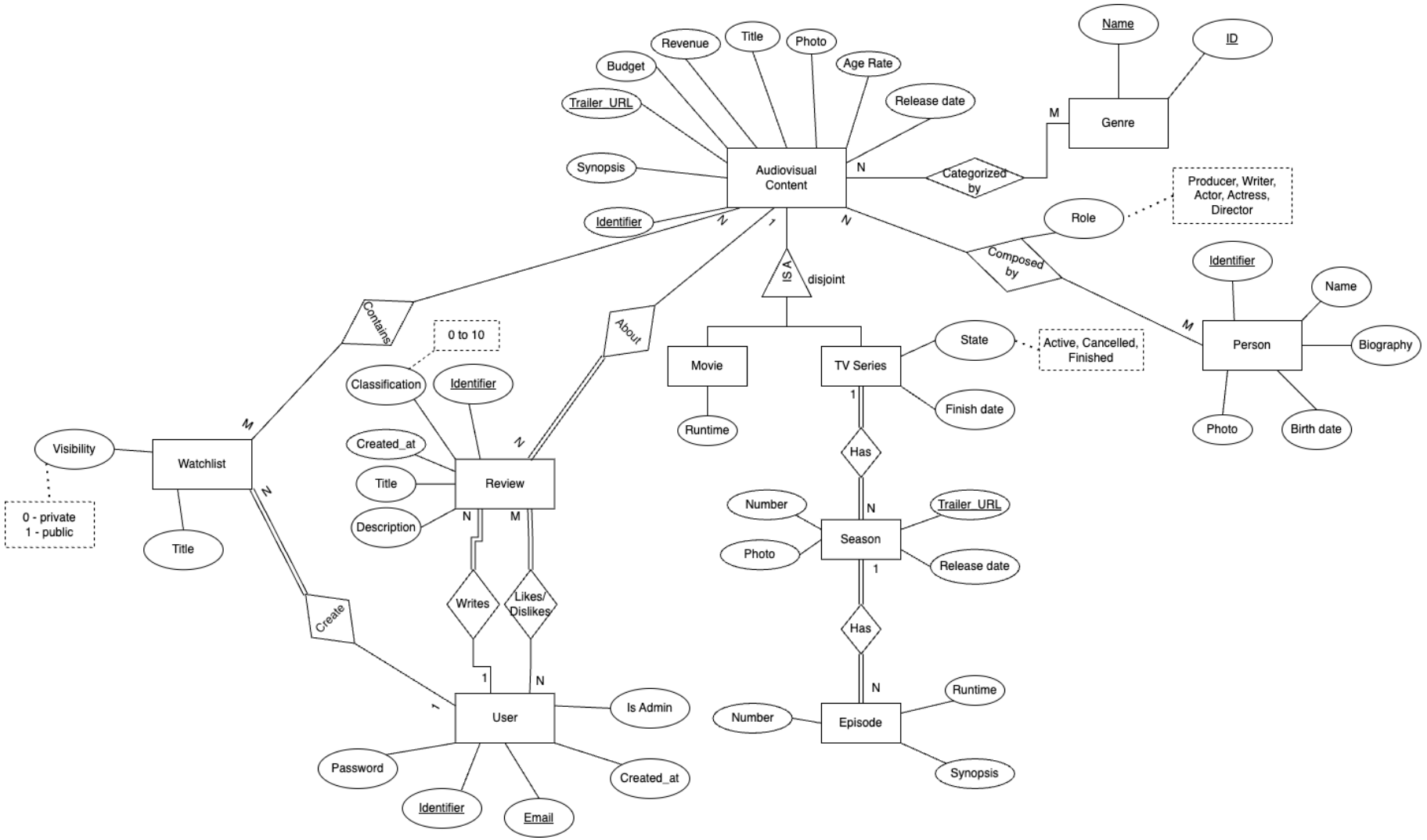
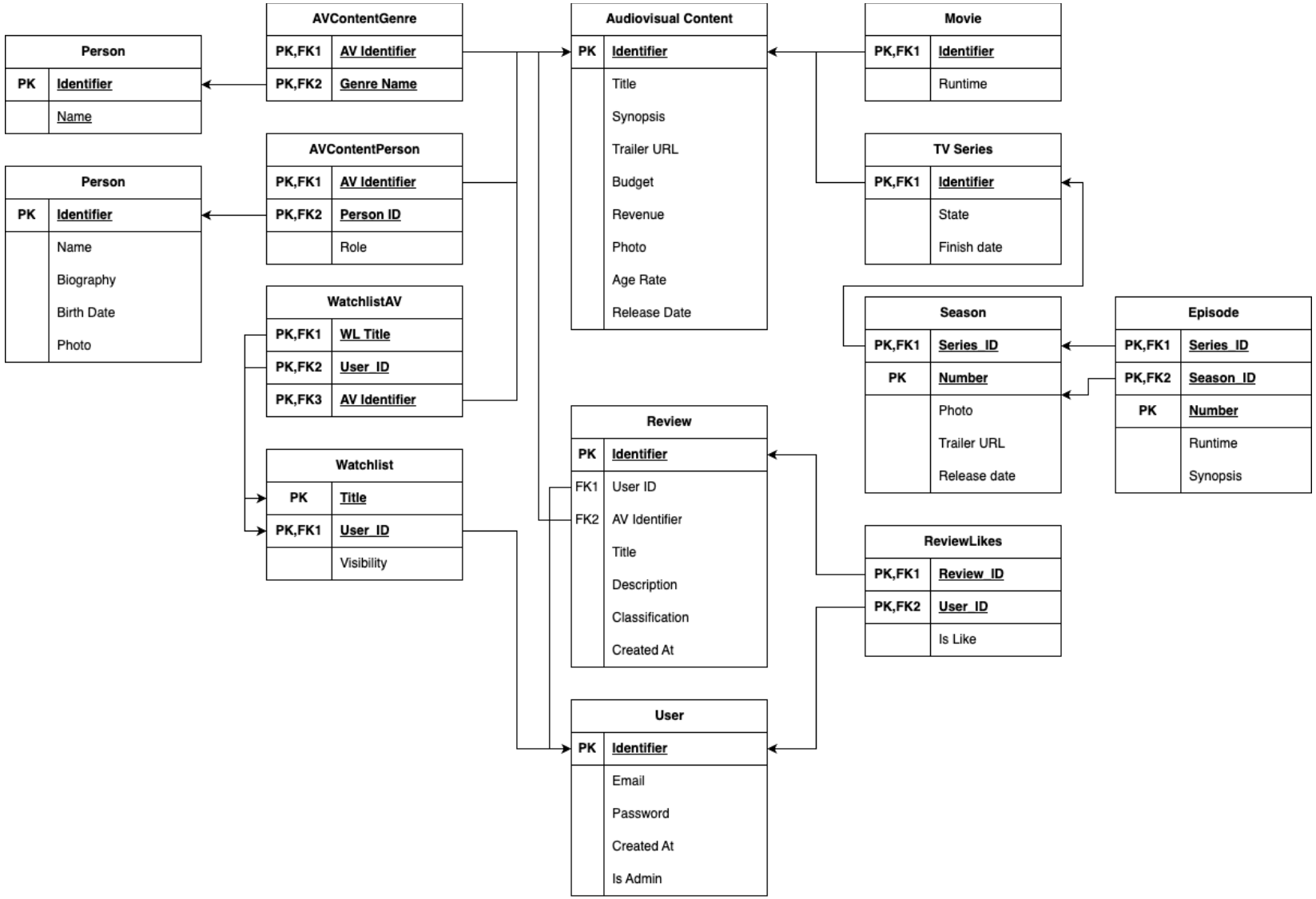


DIAGRAMA ER


Person
PK Identifier
Name
AVContentGenre
PK,FK1 AV Identifier
PK,FK2 Genre Name
Audiovisual Content
PK Identifier
Title
Synopsis
Trailer URL
Budget
Revenue
Photo
Age Rate
Release Date
Movie
PK,FK1 Identifier
Runtime
Person
PK Identifier
Name
Biography
Birth Date
Photo
AVContentPerson
PK,FK1 AV Identifier
PK,FK2 Person ID
Role
TV Series
PK,FK1 Identifier
State
Finish date
WatchlistAV
PK,FK1 WL Title
PK,FK2 User_ID
PK,FK3 AV Identifier
Season
PK,FK1 Series_ID
PK Number
Photo
Trailer URL
Release date
Episode
PK,FK1 Series_ID
PK,FK2 Season_ID
PK Number
Runtime
Synopsis
Review
PK Identifier
FK1 User ID
FK2 AV Identifier
Title
Description
Classification
Created At
Watchlist
PK Title
PK,FK1 User_ID
Visibility
ReviewLikes
PK,FK1 Review_ID
PK,FK2 User_ID
Is Like
User
PK Identifier
Email
Password
Created At
Is Admin

STORED PROCEDURES

```sql
CREATE PROC Authenticate(
    @Email VARCHAR(64),
    @Password VARCHAR(128),

    @UserID INT OUTPUT
)
AS
    DECLARE @HashedPassword VARBINARY(64) = HASHBYTES('SHA2_512', @Password);
    DECLARE @StoredPassword VARBINARY(64);

    SELECT @UserID=ID, @StoredPassword=Password FROM "User" WHERE Email=@Email;

    IF @@ROWCOUNT = 0
    BEGIN
        -- Create a new user
        INSERT INTO "User" (Email, Password) VALUES (@Email, @HashedPassword);
        SET @UserID = @@IDENTITY;
    END
    ELSE
    BEGIN
        -- Check if passwords match
        IF @HashedPassword != @StoredPassword
        BEGIN
            SET @UserID = NULL;
            RAISERROR('Invalid email or password!', 16, 1);
        END
    END
```

```sql
CREATE PROC UpdateWatchlist (
    @OldTitle VARCHAR(32),
    @Title VARCHAR(32),
    @UserID INT,
    @Visibility BIT,
    @AVList AVList READONLY
)
AS
    BEGIN TRAN;

    UPDATE Watchlist SET Title=@Title, Visibility=@Visibility WHERE Title=@OldTitle AND UserID=@UserID;

    DELETE FROM WatchlistAV WHERE WLTitle=@Title AND UserID=@UserID AND AVIdentifier NOT IN (SELECT ID FROM @AVList);

    INSERT INTO WatchlistAV (WLTitle, UserID, AVIdentifier)
        SELECT @Title, @UserID, * FROM @AVList WHERE ID NOT IN (SELECT AVIdentifier FROM WatchlistAV WHERE WLTitle=@Title AND UserID=@UserID);

    COMMIT TRAN;
```

STORED PROCEDURES

```sql
CREATE PROC DeleteReview(
    @UserID INT,
    @ReviewID INT
)
AS
    IF (@UserID != (SELECT UserID FROM Review WHERE ID=@ReviewID))
    BEGIN
        RAISERROR('You cannot delete other people''s reviews', 16, 1);
        RETURN;
    END

    DELETE FROM Review WHERE ID=@ReviewID;
```

```sql
CREATE PROC CreateSerie(
    @Title VARCHAR(64),
    @Synopsis VARCHAR(255),
    @TrailerURL VARCHAR(128),
    @Budget MONEY,
    @Revenue MONEY,
    @Photo VARCHAR(128),
    @AgeRate SMALLINT,
    @ReleaseDate DATE,

    @State VARCHAR(16),
    @FinishDate DATE,

    @GenreList GenreList READONLY,

    @SeasonNumber INT,
    @SeasonPhoto VARCHAR(128),
    @SeasonTrailerURL VARCHAR(128),
    @SeasonReleaseDate DATE,

    @EpNumber INT,
    @EpRuntime SMALLINT,
    @EpSynopsis VARCHAR(255)
)
AS
    BEGIN TRAN;
    BEGIN TRY
        INSERT INTO AudioVisualContent (Title, Synopsis, TrailerURL, Budget, Revenue, Photo, AgeRate, ReleaseDate) VALUES
            (@Title, @Synopsis, @TrailerURL, @Budget, @Revenue, @Photo, @AgeRate, @ReleaseDate);

        DECLARE @ID INT;

        SET @ID = @@IDENTITY;

        INSERT INTO TVSeries (ID, State, FinishDate) VALUES
            (@ID, @State, @FinishDate);

        INSERT INTO AVContentGenre (AVIdentifier, GenreID)
            SELECT @ID, * FROM @GenreList;

        EXEC CreateSeason @ID, @SeasonNumber, @SeasonPhoto, @SeasonTrailerURL, @SeasonReleaseDate, @EpNumber, @EpRuntime, @EpSynopsis;
    END TRY

    BEGIN CATCH
        RAISERROR('Failed to create a serie', 16, 1);
        ROLLBACK TRAN;
    END CATCH

    COMMIT TRAN;
```

UDFS / VIEWS

```sql
CREATE FUNCTION overallScoreByID(@ID INT) RETURNS DECIMAL(4, 2)
AS
BEGIN
    DECLARE @OverallScore DECIMAL(4, 2);
    DECLARE @ScoreSum INT;
    DECLARE @ReviewCount INT;

    SET @ScoreSum = 0;
    SET @ReviewCount = 0;

    DECLARE ReviewCursor CURSOR FAST_FORWARD
    FOR SELECT ID, Classification FROM Review WHERE AVIdentifier = @ID;

    OPEN ReviewCursor;

    DECLARE @ReviewID INT;
    DECLARE @Classification SMALLINT;

    FETCH ReviewCursor INTO @ReviewID, @Classification;

    DECLARE @LikeCount INT;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        SELECT @LikeCount=COUNT(IsLike) FROM ReviewLikes WHERE ReviewID = @ReviewID;

        SET @ReviewCount = @ReviewCount + @LikeCount + 1;
        SET @ScoreSum = @ScoreSum + (@LikeCount + 1) * @Classification;

        FETCH ReviewCursor INTO @ReviewID, @Classification;
    END

    CLOSE ReviewCursor;
    DEALLOCATE ReviewCursor;

    IF @ReviewCount = 0
    BEGIN
        RETURN -1;
    END

    SET @OverallScore = CAST(@ScoreSum AS FLOAT) / @ReviewCount;

    RETURN @OverallScore;
END
```

```sql
CREATE VIEW AllMovies
AS SELECT AudioVisualContent.*, Runtime FROM Movie
JOIN AudioVisualContent ON Movie.ID = AudioVisualContent.ID;

GO
CREATE VIEW AllSeries
AS SELECT AudioVisualContent.*, FinishDate, State FROM TVSeries
JOIN AudioVisualContent ON TVSeries.ID = AudioVisualContent.ID;

GO
CREATE VIEW GetPublicWatchlists
AS SELECT * FROM Watchlist WHERE Visibility = 1;
```

TRIGGERS

```
CREATE TRIGGER SeriesFinishDate ON TVSeries
AFTER INSERT, UPDATE
AS
    DECLARE @FinishDate DATE;
    DECLARE @Id INT;
    DECLARE @State VARCHAR(16);

    SELECT @Id=ID, @FinishDate=FinishDate, @State=State FROM inserted;

    IF (@FinishDate IS NOT NULL AND @State = 'Active')
    BEGIN
        RAISERROR('Cannot set a finish date on an unfinished serie. The serie state must be finished or cancelled', 16, 1);
        ROLLBACK TRAN;
        RETURN;
    END

    IF @FinishDate IS NOT NULL
    BEGIN
        IF (SELECT ReleaseDate FROM AudioVisualContent WHERE ID=@Id) > @FinishDate
        BEGIN
            RAISERROR('Cannot set a finish date before serie release date', 16, 1);
            ROLLBACK TRAN;
            RETURN;
        END
    END

    IF (EXISTS(SELECT * FROM Season WHERE ID=@Id AND ReleaseDate > @FinishDate))
    BEGIN
        RAISERROR('Cannot set a finish date before season release dates', 16, 1);
        ROLLBACK TRAN;
    END
```

```
CREATE TRIGGER DeleteEp ON Episode
AFTER DELETE
AS
    DECLARE @SeasonID INT;
    DECLARE @SerieID INT;
    DECLARE @Number INT;

    SELECT @Number=Number, @SerieID=Series_ID, @SeasonID=Season_ID FROM deleted;

    IF (EXISTS(SELECT * FROM Season WHERE ID=@SerieID AND Number=@SeasonID)
            AND NOT EXISTS(SELECT * FROM Episode WHERE Series_ID=@SerieID AND Season_ID=@SeasonID))
    BEGIN
        RAISERROR('Cannot delete episode because a season must have at least one episode, delete the season instead', 16, 1);
        ROLLBACK;
    END

    IF (EXISTS(SELECT * FROM Episode WHERE Series_ID=@SerieID AND Season_ID=@SeasonID AND Number > @Number))
    BEGIN
        RAISERROR('Cannot delete episode non-sequentially', 16, 1);
        ROLLBACK;
    END
```

ÍNDICES

Índice: non-clustered no AudiovisualContent title

Query: SELECT * FROM AudioVisualContent WHERE Title LIKE 'Forr%';

# Rows	Sem índice (ms)	Com índice (ms)
60	3	3
100000	13	0
200000	20	3
300000	33	7
400000	40	3
500000	53	3
600000	67	7
700000	76	4
800000	107	4

ÍNDICES

Índice: non-clustered User(Email)

Query: EXEC Authenticate
SELECT @UserID=ID, @StoredPassword=Password FROM "User" WHERE Email=@Email;

# Rows	Sem índice (ms)	Com índice (ms)
24	10	7
100000	87	6
200000	87	7
300000	100	7
400000	184	7
500000	107	7
600000	134	4
700000	107	6
800000	114	3

OBRIGADO!

David Amorim e Francisca Silva