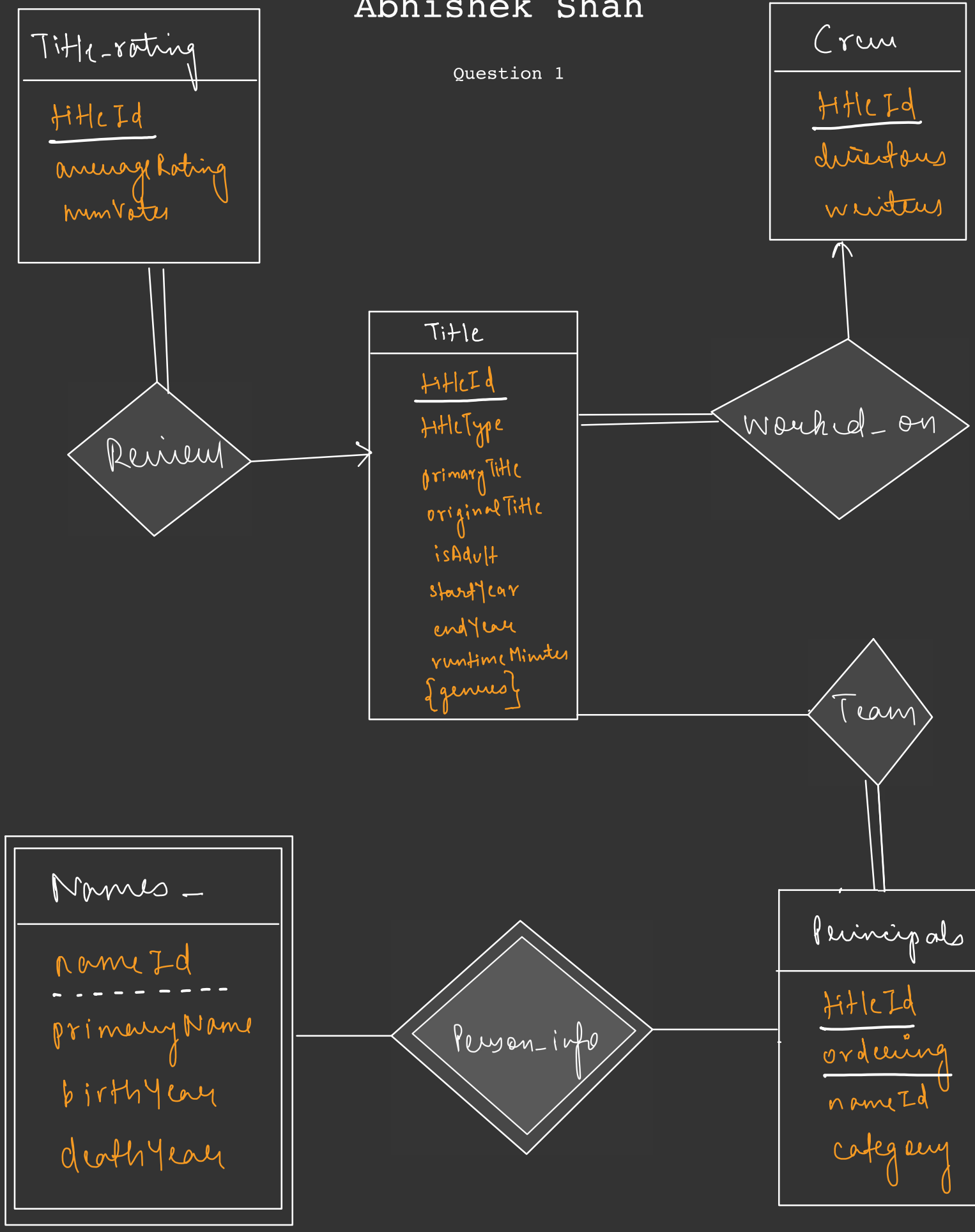
# Abhishek Shah

Question 1



**Question 2**

Based on the ER diagrams, I created a relational model with five tables representing five different entities from my ER diagram which in turn represented the given datasets. The 5 tables created were,

1. **Title** representing attributes from title.basics.tsv
2. **Title_rating** representing attributes from title.ratings.tsv
3. **Principals** representing attributes from title.principals.tsv
4. **Crew** representing attributes from title.crew.tsv
5. **Names_** representing attributes from names.basics.tsv

The code for creating them are given below.

```
CREATE TABLE Title(
        titleId VARCHAR(15) PRIMARY KEY,
        titleType VARCHAR(15),
        primaryTitle TEXT,
        originalTitle TEXT,
        isAdult BOOLEAN,
        startYear INTEGER,
        endYear INTEGER,
        runtimeMinutes INTEGER,
        genres VARCHAR(255)
);
-- Query returned successfully in 132 msec.
```

```
CREATE TABLE Title_rating(
        titleId VARCHAR(15) PRIMARY KEY,
        averageRating FLOAT,
        numVotes INTEGER
);
-- Query returned successfully in 89 msec.
```

```
CREATE TABLE Principals(
        titleId VARCHAR(15),
        ordering_ INTEGER,
        nameId VARCHAR(15),
        category VARCHAR(50),
        job TEXT,
        characters_ TEXT,
        PRIMARY KEY(titleId, ordering_)
);
-- Query returned successfully in 143 msec.
```

---

```
CREATE TABLE Crew(
        titleId VARCHAR(15) PRIMARY KEY,
        directors TEXT,
        writers TEXT
);
-- Query returned successfully in 96 msec.
```

---

```
CREATE TABLE Names(
        nameId VARCHAR(15) PRIMARY KEY,
        primaryName VARCHAR(255) NOT NULL,
        birthYear SMALLINT,
        deathYear SMALLINT,
        primaryProfession TEXT,
        knownForTitles TEXT
);
 -- Query returned successfully in 110 msec.
```

---

**Issues Faced:**

1. For exploring the dataset, I unzipped the files, using Python and explored the dataset using the **pandas module**. After exploring the dataset, the maximum length for the attributes of particular datatype was decided.
2. By exploring the dataset, I also made sure to check my ER diagram and to see if all my entities or decided primary keys and foreign keys sync up together or not.
3. Multiple iterations of correction were needed to finally come up with a ER diagram which eventually led to the creation of these 5 tables.
4. Reserved keywords were handled by using a underscore (_) before the name variable.
5. Original Primary keys (VARCHAR) will be casted to INTEGERS while populating data (Question 4)

---

**Question 3**

---

**title.akas.tsv.gz**

This dataset contains detailed information about all the titles.

It contains the following attributes:

titleId – A string of characters and digits, used to uniquely identify a particular title.

ordering – A number between 1,2,3 to categorize rows for finding a particular title.

title – A string of characters with the name of that title.

region – A string for identifying the Demographics for the title.

language – A string to display the languages the title is in.

types – Array of string values categorizing the types of title they have in the dataset.
It can be anything from a dvd to tv to video, etc.

attributes – If the title has any other title, it will be shown in this column.
This is a multivalued attribute.

isOriginalTitle – Boolean value type. True if the title is not original, false otherwise.

This table is like an alias to the **title.basics.tsv** table. It can be easily said if made a entity out of this table it would be considered a weak entity as this dataset has no relevance of its own.

types and attributes can be considered as a multivalued attribute.

**title.basics.tsv**

This table contains precise information about any title.

It contains the following attributes:

tconst – A string of characters and digits, used to uniquely identify a particular title.

titleType - Array of string values categorizing the types of title they have in the dataset.
It can be anything from a TVEpisode to a short to a movie.

primaryTitle – The best of the best well known title.

originalTitle – The original title in the language that it created.

isAdult - Boolean value type. True if the title is Adult, false otherwise.

startYear – Year when the movie was made

endYear – Year when the particular parts of movies or series came to an end.

runtimeMinutes – Total running time of a title.

genres – Category and type of title. A maximum of three genres are attached to its title.

This table is the main table for defining the entity Title. It contains all the necessary information which is both precise and complete.

genres can be considered as multivalued attribute as it has multiple values to begin with . Almost all the database are going to be linked with this database via the tconst, which is a unique identifier for identifying title.

**title.crew.tsv.gz**

This table contains all the information for all the crew that worked on a particular title.

This contains the following attributes:

tonst - A string of characters and digits, used to uniquely identify a particular title.

directors – array of nconst(name id) of directors who directed a title or multiple titles.

writers - array of nconst(name id) of writers who wrote a title or multiple titles

The directors and writers attributes of this table can act as a relation between the person and the title, that is, these attributes can help relate two tables, names and title to be connected to each other. We'll just have to make sure to give both writers and directors two columns of their own, namely titleId and nameId  to make themselves a relation in a relational model.

**title.episode.tsv.gz**

This table contains information about tv series and their episodes.

This contains the following attributes:

Tconst - A string of characters and digits, used to uniquely identify a particular title.

parentTConst – A string of characters to uniquely identify a TV Series.

seasonNumber – Stores information about season number for which episode/episodes belong to.

episodeNumber – Stores information about the episode number current title (tconst).

For retrieving any data about the Tv series, this table is the only one that can help.

**title.principals.tsv.gz**

This table contains information about all the cast and crew that worked on a title/titles.

This contains the following attributes:

tconst - A string of characters and digits, used to uniquely identify a particular title.

ordering – A integer used to categorise the ordering based on the title

nconst - A string of characters and digits, used to uniquely identify a particular person.

category – The type of job the person does is stored in this.

Job – The job title that was given to the crew/cast member.

characters – The name of the character the person played. This is dependent on the nconst value.

The category attribute of this table can come in very handy. This attribute alone can eliminate the possibility of removing the title.crew.tsv.gz file as it contains almost everything of what was there in the title.crew.tsv.gz.

We can also derive what role a particular person had by combining all attributes in this particular table.

**title.ratings.tsv.gz**

This table contains all the reviews related to the title/titles.

This contains the following attributes:

tconst - A string of characters and digits, used to uniquely identify a particular title.

averageRating – A float value describing all the ratings for a particular title.

numVotes – This attribute can help figure out all the votes the title received till the time the data was recorded.

**name.basics.tsv.gz**

This table contains all the information about a person's particulars.

nconst - A string of characters and digits, used to uniquely identify a particular person.

primaryName – Name of the person by which everyone calls him/her.

birthYear – The year of the birth for the person.

deathYear – The year of the death for the person.

primaryProfession – Array of string of professions the person has. It has a maximum of 3 professions.

knownForTitles – Array of string for all the famous title the person was played throughout the career.

primaryProfession can be considered as a multivalued attribute.

**Question 4**

Aside from creating the tables (Question 2), this is the code for all the necessary things to be done when loading the dataset.

## Table: Title

**--Loading data**

```
COPY Title
FROM 'C:\Users\abbys\OneDrive\Desktop\title.basics.tsv'
DELIMITER E'\t' NULL '\N' CSV HEADER ENCODING 'UTF8' QUOTE E'\b';
--COPY 8668093 Query returned successfully in 1 min 9 secs.
```

**--Removing unwanted characters**

```
UPDATE Title
SET titleId = REPLACE(titleId, 'tt', '');
--Query returned successfully in 4 min 51 secs.
```

**--Casting to INTEGER**

```
ALTER TABLE Title
ALTER COLUMN titleId TYPE INTEGER
USING(titleId::INTEGER);
-- Query returned successfully in 1 min 31 secs.
```

## Table: Title_rating

**--Loading data**

```
COPY Title_rating
FROM 'C:\Users\abbys\OneDrive\Desktop\title.ratings.tsv'
DELIMITER E'\t' NULL '\N' CSV HEADER ENCODING 'UTF8' QUOTE E'\b';
--COPY 1210745 Query returned successfully in 7 secs 846 msec.
```

**--Removing unwanted characters**

```
UPDATE Title_rating

SET titleId = REPLACE(titleId, 'tt', '');

-- Query returned successfully in 22 secs 236 msec.
```

**--Casting to INTEGER**

```
ALTER TABLE Title_rating

ALTER COLUMN titleId TYPE INTEGER

USING(titleId::INTEGER);

-- Query returned successfully in 7 secs 158 msec.
```

**--Setting foreign key**

```
DELETE FROM Title_rating WHERE NOT exists ( SELECT NULL FROM Title WHERE Title_rating.titleId = Title.TitleId );

ALTER TABLE Title_rating ADD CONSTRAINT fk_title_rating_tileId FOREIGN KEY(titleId) REFERENCES Title(titleId);

-- Query returned successfully in 8 secs 97 msec.
```

---

## Table: Names_

**--Loading data**

```
COPY Names_

FROM 'C:\Users\abbys\OneDrive\Desktop\name.basics.tsv'

DELIMITER E'\t' NULL '\N' CSV HEADER ENCODING 'UTF8' QUOTE E'\b';

-- COPY 11376160 Query returned successfully in 1 min 21 secs.
```

**--Dropping unwanted columns**

```
ALTER TABLE Names_
DROP COLUMN primaryProfession;

ALTER TABLE Names_
DROP COLUMN knownForTitles;

-- Query returned successfully in 58 msec.
```

**--Removing unwanted characters**

```
UPDATE Names_

SET NameId = REPLACE(NameId, 'nm', '');

--Query returned successfully in 3 min 37 secs.
```

**--Casting to INTEGER**

```
ALTER TABLE Names_

ALTER COLUMN NameId TYPE INTEGER

USING(NameId::INTEGER);

--Query returned successfully in 1 min 25 secs.
```

---

## Table: Principals

**--Loading data**

```
COPY Principals

FROM 'C:\Users\abbys\OneDrive\Desktop\title.principals.tsv'

DELIMITER E'\t' NULL '\N' CSV HEADER ENCODING 'UTF8' QUOTE E'\b';

--COPY 48654706 Query returned successfully in 4 min 25 secs.
```

**--Dropping unwanted columns**

```
ALTER TABLE Principals
DROP COLUMN job;

ALTER TABLE Principals
DROP COLUMN characters_;

--Query returned successfully in 55 msec.
```

**--Removing unwanted characters**

```
UPDATE Principals

SET titleId = REPLACE(titleId, 'tt', ''), nameId = REPLACE(nameId, 'nm', '');

--Query returned successfully in 15 min 4 secs.
```

**--Casting to INTEGER**

```
ALTER TABLE Principals

ALTER COLUMN TitleId TYPE INTEGER

USING(TitleId::INTEGER);

--Query returned successfully in 5 min 4 secs.
```

```
ALTER TABLE Principals

ALTER COLUMN nameId TYPE INTEGER

USING(nameId::INTEGER);

--Query returned successfully in 4 min 29 secs.
```

**--Setting foreign key**

```
DELETE FROM Principals WHERE NOT exists ( SELECT NULL FROM Title WHERE Principals.titleId = Title.TitleId );

ALTER TABLE Principals ADD CONSTRAINT fk_Principals_tileId FOREIGN KEY(titleId) REFERENCES Title(titleId);

--Query returned successfully in 2 min 8 secs.
```

```
DELETE FROM Principals WHERE NOT exists ( SELECT NULL FROM Names_ WHERE Principals.nameId = Names_.nameId);

ALTER TABLE Principals ADD CONSTRAINT fk_Principals_nameId FOREIGN KEY(nameId) REFERENCES Names_(nameId);

--Query returned successfully in 1 min 49 secs.
```

---

## Table: Crew

**--Loading data**

```
COPY Crew

FROM 'C:\Users\abbys\OneDrive\Desktop\title.crew.tsv'

DELIMITER E'\t' NULL '\N' CSV HEADER ENCODING 'UTF8' QUOTE E'\b';

-- COPY 8658181 Query returned successfully in 49 secs 574 msec.
```

**--Removing unwanted characters**

```
UPDATE Crew
SET titleId = REPLACE(titleId, 'tt', '');
--Query returned successfully in 2 min 38 secs.

UPDATE Crew
SET directors = REPLACE(directors, 'nm', ''), writers = REPLACE(writers, 'nm', '');
--Query returned successfully in 2 min 39 secs.
```

**--Casting to INTEGER**

```
ALTER TABLE Crew
ALTER COLUMN titleId TYPE INTEGER
USING(titleId::INTEGER);
--Query returned successfully in 52 secs 62 msec.
```

**--Setting foreign key**

```
DELETE FROM Crew WHERE NOT exists ( SELECT NULL FROM Title WHERE Crew.titleId = Title.TitleId );
ALTER TABLE Crew ADD CONSTRAINT fk_Crew_tileId FOREIGN KEY(titleId) REFERENCES Title(titleId);
--Query returned successfully in 21 secs 30 msec.
```

---

**Runtime Report for Loading the Entire Dataset:**

| **Table** | **Time to load and run everything (mm:ss)** |
|---|---|
| Title | 07:54 |
| Title_rating | 00:44 |
| Names_ | 07:35 |
| Principals | 33:09 |
| Crew | 07:31 |

---

**Total time taken = ~55 minutes**

**Issues faced:**

1. As all my primary keys needed to be INTEGERS and not STRINGS(VARCHAR).

   My primary keys (titleId, nameId) were a combination of two characters(tt, nm) followed by digits. So, I needed to get rid of those two additional characters before casting my primary key as INTEGER.

   I solved this issue by using UPDATE command to first update that column and followed by the REPLACE command to replace the additional characters("tt, "nm") by nothing("").

   After this step, I casted my primary keys to INTEGER using the ALTER command.

2. During the creation of foreign keys, I was not able to assign the foreign keys directly as I was getting a type incompatibility error.
   eg: The primary key in my **Title** table which was **tableId** was casted to integer after its creation. In the **Title_rating** table where I was mapping the **titleId** of the **Title_rating** table as the **foreign key** of that table.

   To counter that I needed to cast the **titleId** of the **Title_rating** table to **INTEGER** first and then I added the **foreign key constraint** for the same.

   The reason behind using foreign key constraint was that, I was not able to assign the foreign key directly as some rows were absent in the data for the same column in the other table.

   I solved this, by first deleting the rows that contained those missing values and then creating and assigning the foreign key constraint and the foreign key.

**Question 5**

```
"""
file: transactions.py
description: CSCI 620, Assignment 1 - Question 5
language: Python
author: Abhishek Shah, as5553
"""

import psycopg2


def transactions():
    """
    This method will try to insert rows in the database
connected.
    """
    conn = None
    updated_rows = 0
    try:
        # establishing the connection
        conn = psycopg2.connect(
            host='localhost',
            database='imdb620',
            user='postgres',
            password='Abhishek@123',
            port=5432)

        cur = conn.cursor()

        # inserting row 1
        row_1 = 'INSERT INTO Title(titleId, TitleType,
primaryTitle, OriginalTitle, isAdult,\
        startYear, endYear, runtimeMinutes, genres) VALUES
(%s, %s, %s, %s, %s, %s, %s, %s, %s)'
        row_1_values = (10144553, 'movie', 'Big Data', 'CSCI',
'false', 2021, 2023, 10, 'News')
        cur.execute(row_1, row_1_values)

        # inserting row 2
        row_2 = 'INSERT INTO Title(titleId, TitleType,
primaryTitle, OriginalTitle, isAdult,\
        startYear, endYear, runtimeMinutes, genres) VALUES
(%s, %s, %s, %s, %s, %s, %s, %s, %s)'
        row_2_values = (10144550, 'tvEpisode', 'Algorithms',
'CSCI', 'true', 2021, 2023, 10, 'Comedy')
        cur.execute(row_2, row_2_values)

        # inserting row 3
```

```
        row_3 = 'INSERT INTO Title(titleId, TitleType,
primaryTitle, OriginalTitle, isAdult,\
        startYear, endYear, runtimeMinutes, genres) VALUES
(%s, %s, %s, %s, %s, %s, %s, %s, %s)'
        row_3_values = (10144558, 'short', 'Artificial',
'CSCI', 'false', 2021, 2023, 10, 'News')
        cur.execute(row_3, row_3_values)

        # keeping count of updated rows if any
        updated_rows = cur.rowcount

        # committing the changes
        conn.commit()
        # closing the cursor connection
        cur.close()
    except (Exception, psycopg2.DatabaseError) as error:
        print("Transaction Failed. Reverting all operations.",
error)
        conn.rollback()
    finally:
        if conn is not None:
            conn.close()
            print("Connection closed")

    print("Updated_rows = ", updated_rows)


if __name__ == '__main__':
    transactions()
```