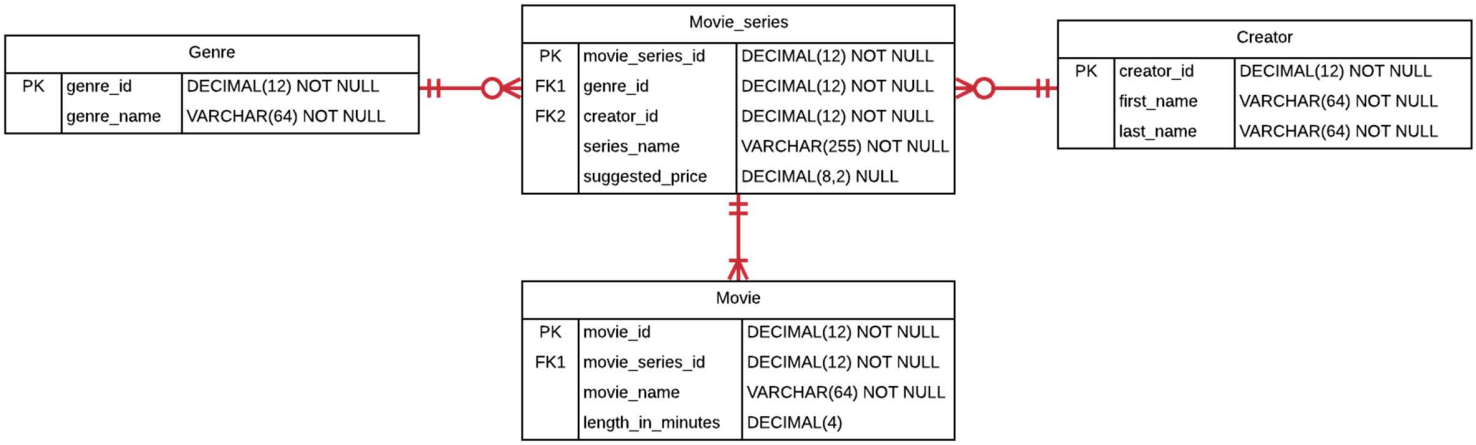
Section One – Aggregating Data

Section Background

To practice aggregating data, you will be working with the following simplified Movie Series schema.



This schema contains basic information about various movie series and the movies that comprise them, such as the Star Wars series with its movies.

In this schema, the Movie_series table represents the overall movie series, and contains a primary key, the name of the series, foreign keys to its genre and creator, and a suggested price for the entire series. The Genre table represents the genre of a movie such as “Fantasy”, “Family Film”, and the like. It contains a primary key and the name of the genre. The Creator table represents who created the series, and contains a primary key and the name of each creator. The Movie table represents movies that comprise each movie series, and contains a primary key, a foreign key to the movie’s series, the name of the movie, and the length of the movie, in minutes.

The schema is intentionally simplified compared to what you might see in a real-world production schema. Many attributes and entities that would exist in a production database are not present. Nevertheless, there is sufficient complexity in the existing relationships and attributes to challenge you to learn various aggregation scenarios you encounter in real-world schemas.

As a reminder, for each step that requires SQL, make sure to capture a screenshot of the command and the results of its execution. *Further, make sure to eliminate unneeded*

columns from the result set, to name your columns something user-friendly and human readable, and to format any prices as currencies.

Section Steps

1. *Creating Table Structure and Data* – Create the tables in the schema, including all of their columns, datatypes, and constraints, and populate the tables with data. Most but not all of the data is given to you in the table below; *you should also insert information for one additional movie series of your choosing.* Although the data is in flattened representation below, you will of course insert the data relationally into the schema with foreign keys referencing the appropriate primary keys.

Genre	Creator	Series	Suggested Price	Movie	Length
Fantasy	George Lucas	Star Wars	$129.99	Episode I: The Phantom Menace	136
Fantasy	George Lucas	Star Wars	$129.99	Episode II: Attack of the Clones	142
Fantasy	George Lucas	Star Wars	$129.99	Episode III: Revenge of the Sith	140
Fantasy	George Lucas	Star Wars	$129.99	Episode IV: A New Hope	121
Family Film	John Lasseter	Toy Story	$22.13	Toy Story	121
Family Film	John Lasseter	Toy Story	$22.13	Toy Story 2	135
Family Film	John Lasseter	Toy Story	$22.13	Toy Story 3	148
Fantasy	John Tolkien	Lord of the Rings		The Lord of the Rings: The Fellowship of the Ring	228
Fantasy	John Tolkien	Lord of the Rings		The Lord of the Rings: The Two Towers	235
Fantasy	John Tolkien	Lord of the Rings		The Lord of the Rings: The Return of the King	200

Note that the suggested price for the Lord of the Rings series is null (has no value).

```sql
CREATE TABLE Genre (
    genre_id DECIMAL(12) NOT NULL PRIMARY KEY,
    genre_name VARCHAR(64) NOT NULL
    );

CREATE TABLE Creator (
    creator_id DECIMAL(12) NOT NULL PRIMARY KEY,
    first_name VARCHAR(64) NOT NULL,
    last_name VARCHAR(64) NOT NULL
    );

CREATE TABLE Movie_series (
    movie_series_id DECIMAL(12) NOT NULL PRIMARY KEY,
    genre_id DECIMAL(12) NOT NULL FOREIGN KEY REFERENCES Genre(genre_id),
    creator_id DECIMAL(12) NOT NULL FOREIGN KEY REFERENCES Creator(creator_id),
    series_name VARCHAR(255) NOT NULL,
    suggested_price DECIMAL(8, 2) NULL,
    );

CREATE TABLE Movie (
    movie_id DECIMAL(12) NOT NULL PRIMARY KEY,
    movie_series_id DECIMAL(12) NOT NULL FOREIGN KEY REFERENCES Movie_series(movie_series_id),
    movie_name VARCHAR(64) NOT NULL,
    length_in_minutes DECIMAL(4)
    );
```

121 %

Messages

```
Commands completed successfully.
```

```sql
-- Genre (1): Fantasy. Creator (1): George Lucas. Series (1): Star Wars.
INSERT INTO Genre (genre_id, genre_name)
VALUES (1, 'Fantasy');
INSERT INTO Creator (creator_id, first_name, last_name)
VALUES (1, 'George', 'Lucas');
INSERT INTO Movie_series (movie_series_id, genre_id, creator_id, series_name, suggested_price)
VALUES (1, 1, 1, 'Star Wars', 129.99);
INSERT INTO Movie (movie_id, movie_series_id, movie_name, length_in_minutes)
VALUES (1, 1, 'Episode I: The Phantom Menace', 136);
INSERT INTO Movie (movie_id, movie_series_id, movie_name, length_in_minutes)
VALUES (2, 1, 'Episode II: Attack of the Clones', 142);
INSERT INTO Movie (movie_id, movie_series_id, movie_name, length_in_minutes)
VALUES (3, 1, 'Episode III: Revenge of the Sith', 140);
INSERT INTO Movie (movie_id, movie_series_id, movie_name, length_in_minutes)
VALUES (4, 1, 'Episode IV: A New Hope', 121);
```

121 %

Messages

```
(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)
```

```sql
-- Genre (2): Family Film. Creator (2): John Lasseter. Series (2): Toy Story.
INSERT INTO Genre (genre_id, genre_name)
VALUES (2, 'Family Film');
INSERT INTO Creator (creator_id, first_name, last_name)
VALUES (2, 'John', 'Lasseter');
INSERT INTO Movie_series (movie_series_id, genre_id, creator_id, series_name, suggested_price)
VALUES (2, 2, 2, 'Toy Story', 22.13);
INSERT INTO Movie (movie_id, movie_series_id, movie_name, length_in_minutes)
VALUES (5, 2, 'Toy Story', 121);
INSERT INTO Movie (movie_id, movie_series_id, movie_name, length_in_minutes)
VALUES (6, 2, 'Toy Story 2', 135);
INSERT INTO Movie (movie_id, movie_series_id, movie_name, length_in_minutes)
VALUES (7, 2, 'Toy Story 3', 148);
```

121 %

Messages

```
(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)
```

```sql
-- Genre (1): Fantasy. Creator (3): John Tolkien. Series (3): Lord of the Rings.
INSERT INTO Creator (creator_id, first_name, last_name)
VALUES (3, 'John', 'Tolkien');
INSERT INTO Movie_series (movie_series_id, genre_id, creator_id, series_name, suggested_price)
VALUES (3, 1, 3, 'Lord of the Rings', NULL);
INSERT INTO Movie (movie_id, movie_series_id, movie_name, length_in_minutes)
VALUES (8, 3, 'The Lord of the Rings: The Fellowship of the Ring', 228);
INSERT INTO Movie (movie_id, movie_series_id, movie_name, length_in_minutes)
VALUES (9, 3, 'The Lord of the Rings: The Two Towers', 235);
INSERT INTO Movie (movie_id, movie_series_id, movie_name, length_in_minutes)
VALUES (10, 3, 'The Lord of the Rings: The Return of the King', 200);
```

121 %

Messages

```
(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)
```

```sql
-- Genre (2): Family Film. Creator (4): Sergio Pablos. Series (4): Despicable Me.
INSERT INTO Creator (creator_id, first_name, last_name)
VALUES (4, 'Sergio', 'Pablos');
INSERT INTO Movie_series (movie_series_id, genre_id, creator_id, series_name, suggested_price)
VALUES (4, 2, 4, 'Despicable Me', 19.99);
INSERT INTO Movie (movie_id, movie_series_id, movie_name, length_in_minutes)
VALUES (11, 4, 'Despicable Me', 95);
INSERT INTO Movie (movie_id, movie_series_id, movie_name, length_in_minutes)
VALUES (12, 4, 'Despicable Me 2', 98);
INSERT INTO Movie (movie_id, movie_series_id, movie_name, length_in_minutes)
VALUES (13, 4, 'Despicable Me 3', 96);
```

121 %

Messages

```
(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)

(1 row affected)
```

Command to check if values were inserted correctly:

```sql
SELECT genre_name AS Genre, first_name + ' ' + last_name AS Creator,
    series_name as Series, format(suggested_price, '$.00') as 'Suggested Price',
    movie_name as Movie, length_in_minutes as Length
FROM Genre
INNER JOIN Movie_series ON Movie_series.genre_id = Genre.genre_id
INNER JOIN Creator ON Creator.creator_id = Movie_series.creator_id
INNER JOIN Movie ON Movie.movie_series_id = Movie_series.movie_series_id
ORDER BY first_name ASC;
```

121 %

Results | Messages

	Genre	Creator	Series	Suggested Price	Movie	Length
1	Fantasy	George Lucas	Star Wars	$129.99	Episode I: The Phantom Menace	136
2	Fantasy	George Lucas	Star Wars	$129.99	Episode II: Attack of the Clones	142
3	Fantasy	George Lucas	Star Wars	$129.99	Episode III: Revenge of the Sith	140
4	Fantasy	George Lucas	Star Wars	$129.99	Episode IV: A New Hope	121
5	Family Film	John Lasseter	Toy Story	$22.13	Toy Story	121
6	Family Film	John Lasseter	Toy Story	$22.13	Toy Story 2	135
7	Family Film	John Lasseter	Toy Story	$22.13	Toy Story 3	148
8	Fantasy	John Tolkien	Lord of the Rings	NULL	The Lord of the Rings: The Fellowship of the Ring	228
9	Fantasy	John Tolkien	Lord of the Rings	NULL	The Lord of the Rings: The Two Towers	235
10	Fantasy	John Tolkien	Lord of the Rings	NULL	The Lord of the Rings: The Return of the King	200
11	Family Film	Sergio Pablos	Despicable Me	$19.99	Despicable Me	95
12	Family Film	Sergio Pablos	Despicable Me	$19.99	Despicable Me 2	98
13	Family Film	Sergio Pablos	Despicable Me	$19.99	Despicable Me 3	96

2. *Counting Matches* – A video reseller needs to know how many movies are available that are at least two hours and fifteen minutes long. Write a single query to fulfill this request.

```sql
SELECT COUNT(movie_name) AS 'Count movies duration >= 135 minutes'
FROM Movie
WHERE length_in_minutes >= 135;
```

121 %

Results | Messages

	Count movies duration >= 135 minutes
1	8

3. *Determining Highest and Lowest* – The same video reseller needs to know the price of the most expensive and least expensive series. Write a single query that fulfill this request. Explain how and why the SQL processor treated the suggested price for the Lord of the Rings series differently than the other suggested price values.

```sql
SELECT format(MAX(suggested_price), '$.00') AS 'Price most expensive series',
       format(MIN(suggested_price), '$.00') AS 'Price least expensive series'
FROM Movie_series;
```

121 %

Results | Messages

	Price most expensive series	Price least expensive series
1	$129.99	$19.99

The price for the Lord of the Rings series was not taken into account by the aggregation function because it is NULL. NULL should not be misunderstood as the number 0. NULL is != 0. The lowest price was thus $19.99 for the Despicable Me series and not NULL for the Lord of the Rings series.

4. *Grouping Aggregate Results* – A film production company is considering purchasing the rights to extend a series, and needs to know the name of each movie series, along with the number of movies in each series. Write a single query to fulfill this request.

```sql
SELECT series_name AS Series, COUNT(movie_name) AS 'Count movies per series'
FROM Movie_series
JOIN Movie ON Movie.movie_series_id = Movie_series.movie_series_id
GROUP BY series_name;
```

121 %

Results | Messages

	Series	Count movies per series
1	Despicable Me	3
2	Lord of the Rings	3
3	Star Wars	4
4	Toy Story	3

Note: JOIN is same as INNER JOIN.

5. *Limiting Results by Aggregation* – A same film production company wants to search for genres that have at least 6 associated movies. Write a single query to fulfill this

request, making sure to list only genres that have at least 6 movies, along with the number of movies for the genre.

```
99  SELECT genre_name AS Genre, COUNT(movie_name) AS 'Count movies per genre - if >= 6 movies'
100 FROM Genre
101 JOIN Movie_series ON Movie_series.genre_id = Genre.genre_id
102 JOIN Movie ON Movie.movie_series_id = Movie_series.movie_series_id
103 GROUP BY genre_name
104 HAVING COUNT(movie_name) >= 6;
```

121 %

Results | Messages

	Genre	Count movies per genre - if >= 6 movies
1	Family Film	6
2	Fantasy	7

6. *Adding Up Values* – Boston University wants to offer its students a movie-binge weekend by playing every movie in a series. To make sure the series is as bingeable as possible, BU wants to be sure the series will run for at least 9 hours. Write a single query that gives this information, with useful columns.

```
107 SELECT series_name AS Series, SUM(length_in_minutes) AS 'Series length in minutes - if >= 9 h'
108 FROM Movie_series
109 JOIN Movie ON Movie.movie_series_id = Movie_series.movie_series_id
110 GROUP BY series_name
111 HAVING SUM(length_in_minutes) >= 540;
```

121 %

Results | Messages

	Series	Series length in minutes - if >= 9 h
1	Lord of the Rings	663

7. *Integrating Aggregation with Other Constructs* – A research institution requests the names of all movie series' creators, as well as the number of "Fantasy" movies they have created (even if they created none). The institution wants the list to be ordered from most to least; the creator who created the most fantasy films will be at the top of the list, and the one with the least will be at the bottom. Write a single query that gives this information, with useful columns.

```
128 SELECT first_name + ' ' + last_name AS Creators, COUNT(genre_name) AS 'Count fantasy movies created'
129 FROM Creator
130 JOIN Movie_series ON Movie_series.creator_id = Creator.creator_id
131 LEFT JOIN Genre ON Genre.genre_id = Movie_series.genre_id AND Genre.genre_name = 'Fantasy'
132 JOIN Movie ON Movie.movie_series_id = Movie_series.movie_series_id
133 GROUP BY genre_name, first_name, last_name
134 ORDER BY genre_name DESC;
135
```

121 %

Results | Messages

	Creators	Count fantasy movies created
1	George Lucas	4
2	John Tolkien	3
3	John Lasseter	0
4	Sergio Pablos	0

Explanation:
In contrast to the previous questions, this question is special: all creators should be included in the results set, even if they have not created a movie of the 'Fantasy' genre. To accomplish this, AND Genre.genre_name = 'Fantasy' (line 131) filters out the non-fantasy movies, which, with an (INNER) JOIN in line 131, would result in a results set

only containing the values that have matching values in the joined tables, which would remove the rows that would have been NULL in the genre_name column from the results set:

```
114 -- Not part of final answer, just a helpful intermediate step for understanding.
115 SELECT first_name, last_name, genre_name, movie_name
116 FROM Creator
117 JOIN Movie_series ON Movie_series.creator_id = Creator.creator_id
118 JOIN Genre ON Genre.genre_id = Movie_series.genre_id AND Genre.genre_name = 'Fantasy'
119 JOIN Movie ON Movie.movie_series_id = Movie_series.movie_series_id;
```

121 %

Results | Messages

	first_name	last_name	genre_name	movie_name
1	George	Lucas	Fantasy	Episode I: The Phantom Menace
2	George	Lucas	Fantasy	Episode II: Attack of the Clones
3	George	Lucas	Fantasy	Episode III: Revenge of the Sith
4	George	Lucas	Fantasy	Episode IV: A New Hope
5	John	Tolkien	Fantasy	The Lord of the Rings: The Fellowship of the Ring
6	John	Tolkien	Fantasy	The Lord of the Rings: The Two Towers
7	John	Tolkien	Fantasy	The Lord of the Rings: The Return of the King

To prevent this, the LEFT JOIN is used in line 131, which here keeps every Creator in the results set with every row that contains the value NULL in the genre_name column:

```
114 -- Not part of final answer, just a helpful intermediate step for understanding.
115 SELECT first_name, last_name, genre_name, movie_name
116 FROM Creator
117 JOIN Movie_series ON Movie_series.creator_id = Creator.creator_id
118 LEFT JOIN Genre ON Genre.genre_id = Movie_series.genre_id AND Genre.genre_name = 'Fantasy'
119 JOIN Movie ON Movie.movie_series_id = Movie_series.movie_series_id;
```

121 %

Results | Messages

	first_name	last_name	genre_name	movie_name
1	George	Lucas	Fantasy	Episode I: The Phantom Menace
2	George	Lucas	Fantasy	Episode II: Attack of the Clones
3	George	Lucas	Fantasy	Episode III: Revenge of the Sith
4	George	Lucas	Fantasy	Episode IV: A New Hope
5	John	Lasseter	NULL	Toy Story
6	John	Lasseter	NULL	Toy Story 2
7	John	Lasseter	NULL	Toy Story 3
8	John	Tolkien	Fantasy	The Lord of the Rings: The Fellowship of the Ring
9	John	Tolkien	Fantasy	The Lord of the Rings: The Two Towers
10	John	Tolkien	Fantasy	The Lord of the Rings: The Return of the King
11	Sergio	Pablos	NULL	Despicable Me
12	Sergio	Pablos	NULL	Despicable Me 2
13	Sergio	Pablos	NULL	Despicable Me 3

Then when COUNT is applied to the genre_name column with GROUP BY as indicated in the above answer, the NULL rows will be counted as 0 and the expected and correct result will be returned.

Section Two –Data Visualization

Section Background

Data visualization is presenting information in visual form, commonly with charts and graphs. People are adept at recognizing patterns, trends, and differences visually. Visual data stories are understood accurately and quickly; recognition comes much more slowly with pages and pages of text and tables.

In the modern age of data driven decision-making, data stories are important for any field – sales, finance, human resources, engineering, information technology, just to name a few. Conveying those data stories effectively is just as important. If you can design and implement effective databases, and also build visualizations from your database to tell data stories, you will have a skillset desired by organizations worldwide. In this section, you have a chance to visualize data by writing queries to obtain results, and using those results to create commonly used charts.

Section Steps

8. *Visualizing Data with One or Two Measures* – SQL results were obtained in #4, in particular the name each movies series along with total number of movies in each series. To address this step, you will need to expand this SQL to also include the suggested price of each series. The SQL will retrieve the name, suggested price, and the total number of movies in each move series. Use these results to address the following.

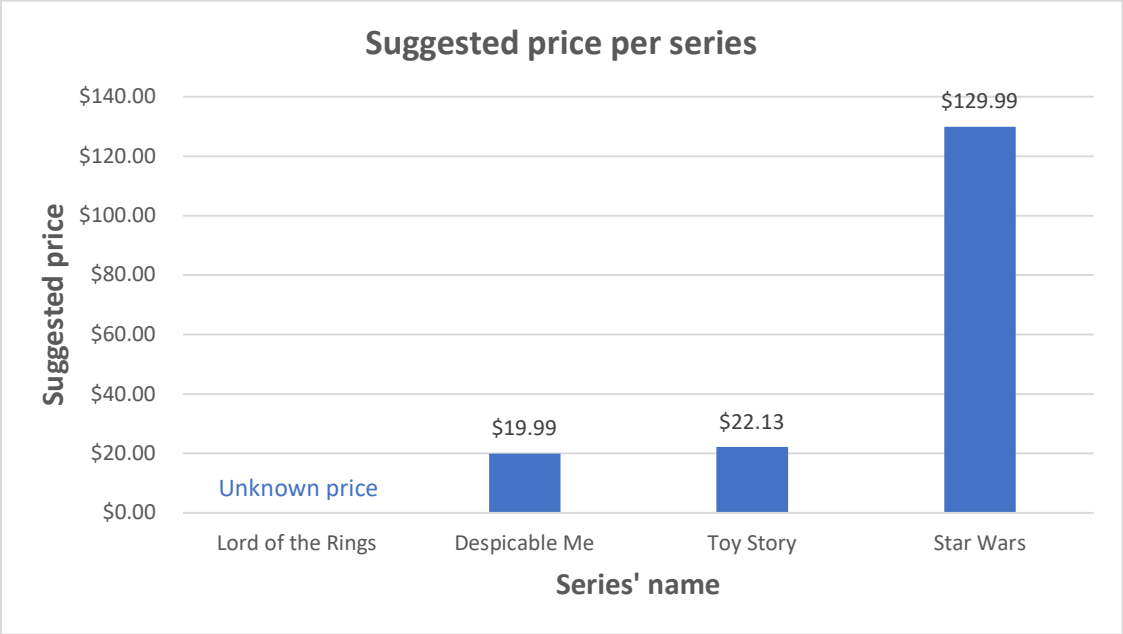
```sql
SELECT series_name AS 'Series name', format(suggested_price, '$.00') AS 'Suggested price',
    COUNT(series_name) AS 'Total movies per series'
FROM Movie_series
JOIN Movie ON Movie.movie_series_id = Movie_series.movie_series_id
GROUP BY series_name, suggested_price;
```

121 %

Results | Messages

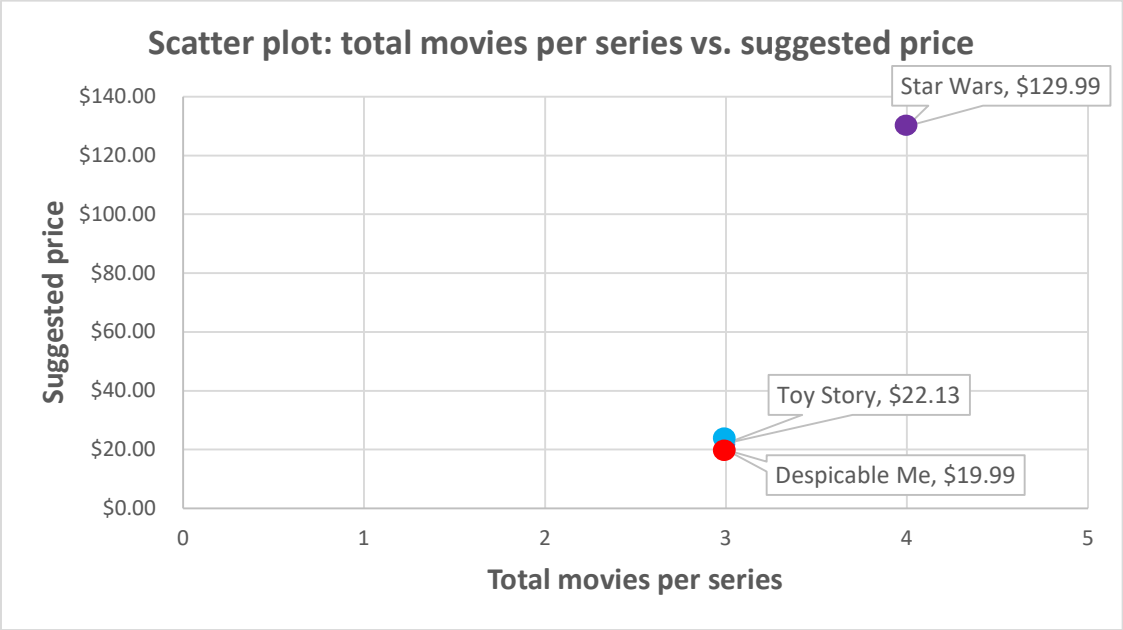
	Series name	Suggested price	Total movies per series
1	Lord of the Rings	NULL	3
2	Despicable Me	$19.99	3
3	Toy Story	$22.13	3
4	Star Wars	$129.99	4

   a. Create a bar chart with the series' name as one axis, and the series' price as another axis. Explain the story this visualization describes.



The bar chart indicates in the caption that the "suggested price per series" data is plotted. So, the bar chart does not inform about actual prices that have to be paid when a series is bought, but about suggested prices. The y-axis indicates "suggested price" and the x-axis indicates "series' name". The plot and thus the data contains a series named Lord of the Rings for which no suggested price information is available, therefore Lord of the Rings was listed with the information "unknown price". Whether or not the Lord of the Rings series should be part of the visualization is entirely dependent on the use case. If the plot should inform about all series and associated prices, Lord of the Rings must be part of the plot. If, however, only the series with known prices is asked for, then Lord of the Rings does not have to be part of the plot. Here it was decided that all data points should be part of the plot.
Despicable Me has a suggested price of $19.99, which is only slightly lower than Toy Story, which has a suggested price of $22.13. While the Star Wars series has by far the highest suggested price of $129.99. If it is looked at the $20 increments on the y-axis, it can be seen that Star Wars has more than six times the suggested price of Despicable Me; and almost six times the suggested price of Toy Story. However, what is not clear from the bar chart is why this is so. To know this, additional information is needed.

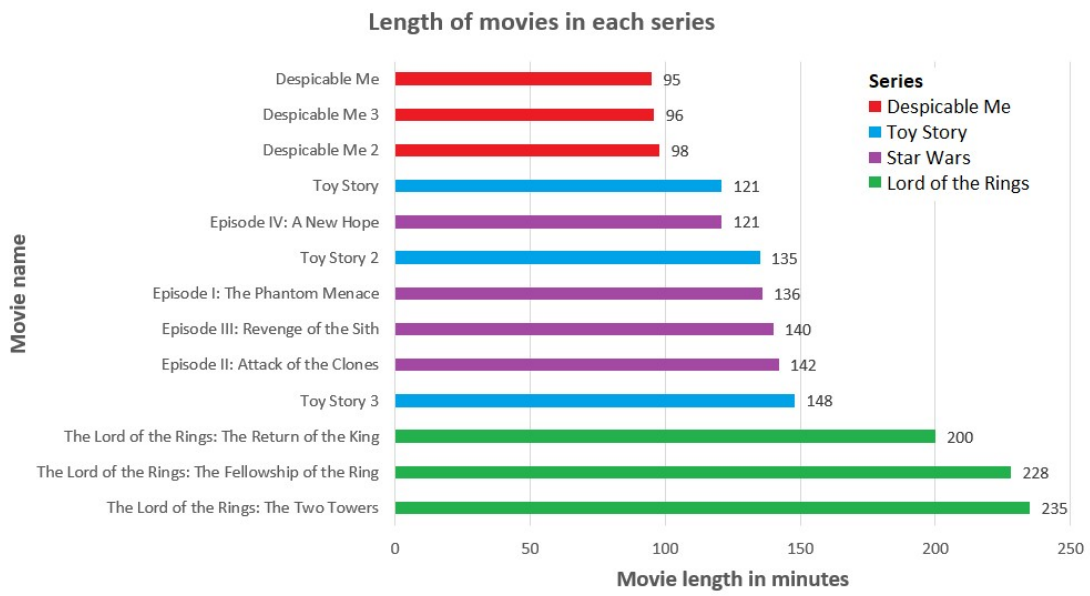
b. Create a scatterplot with the series' price as on axis, and the number of movies in the series as another axis. Ensure that each series is labeled with its name, either directly or with a legend. Explain the story this visualization describes.



Note: The suggested price for the Lord of the Rings series, which consists of 3 movies, is unknown. Therefore, the Lord of the Rings series was not plotted.

The scatter plot indicates in the caption that the "total movies per series" are plotted against (versus) the "suggested price" data. The scatter plot does not inform about actual prices, but suggested prices. The y-axis indicates "suggested price" and the x-axis indicates "total movies per series". The scatter plot plots three differently colored points. A purple point that has the label Star Wars and suggested priced $129.99 (4 movies per series). A blue point representing Toy Story with a suggested price of $22.13 (3 movies per series) and the red point representing Despicable Me with a suggested price of $19.99 (3 movies per series). However, Star Wars has a far higher suggested price than Toy Story and Despicable Me, which is visually evident in the scatter plot by the distance of the points – of around $100 on the y-axis. The scatter plot also shows that Toy Story and Despicable Me are very close in suggested price, so that even the points overlap; the series Toy Story and Despicable Me only have a price difference of $2.14. The scatter plot also contains a (foot)note that informs that there is another series called Lord of the Rings that is not included in the scatter plot because the suggested price for Lord of the Rings is unknown.

9. *Another Data Visualization* – Create a visualization of your choosing for data in the Movie schema. The visualization should tell a useful story. If you find that you need more movies in the schema to tell the story well, feel free to add them. Make sure to explain the data story, and to explain why you chose that particular chart or visualization.



The horizontal bar chart indicates in the caption that the “length of movies in each series” data is plotted. The y-axis indicates “movie name”, the x-axis indicates “movie length in minutes”, and the colors of the bars indicate the “series” name. It can be seen that the Despicable Me series consists of three movies (bars colored red) that have the shortest running times of 95, 96, and 98 minutes. The Toy Story series (bars colored blue) movie Toy Story (1) and a Star Wars series (bars colored violet) movie Episode IV: A New Hope have exactly the same running time of 121 minutes. The movies Toy Story 2 with a running time of 135 minutes and Toy Story 3 with a running time of 148 minutes enclose three Star Wars movies with runtimes of 136 to 142 minutes. The three movies of the Lord of the Rings series (bars colored green) are unchallenged at the top of the runtimes with runtimes of 200, 228, and 235 minutes.
The horizontal bar chart with the y-axis "movie length in minutes", x-axis "movie name", and bars colored according to series was used to plot the sorted running times of all movies per series to see at a glance how long each movie runs in relation to the other movies in the same series and different series. Anyone interested in movies can now select individual movies or series that correspond to their personal preferred runtime. Since the tables in the database contain only a few movies, each movie was plotted with its running time. If the database contained more movies and series, the running times would have had to be aggregated. For example, each series could then have been plotted with its associated average movie running time and number of movies per series, but without movie names.