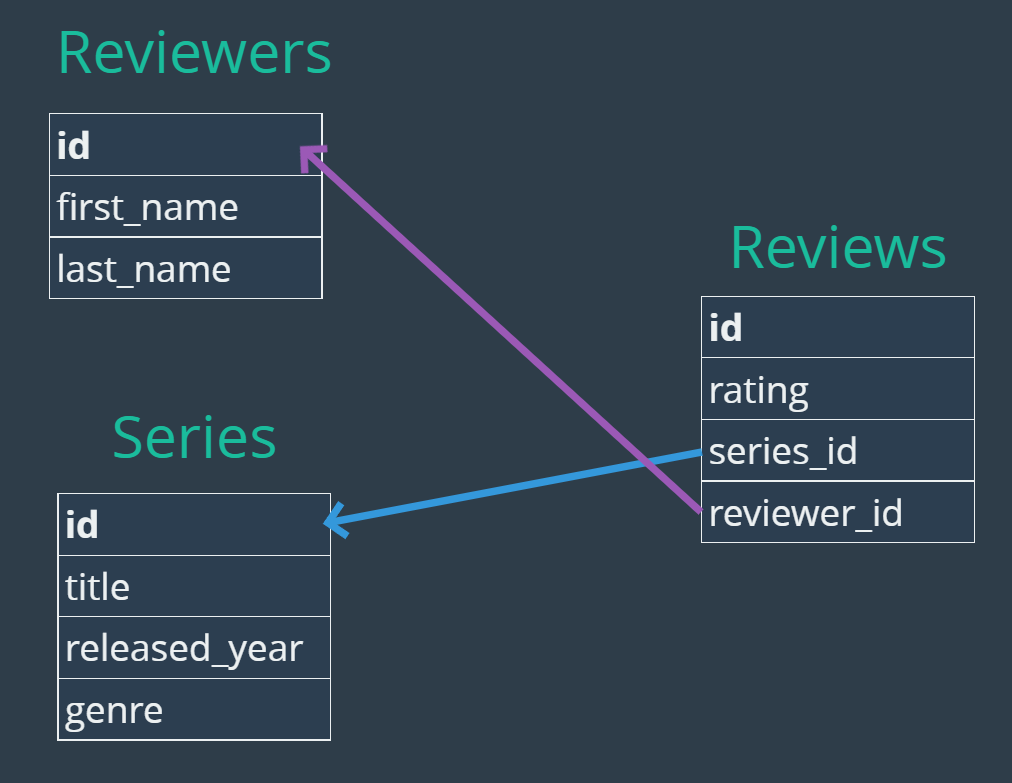
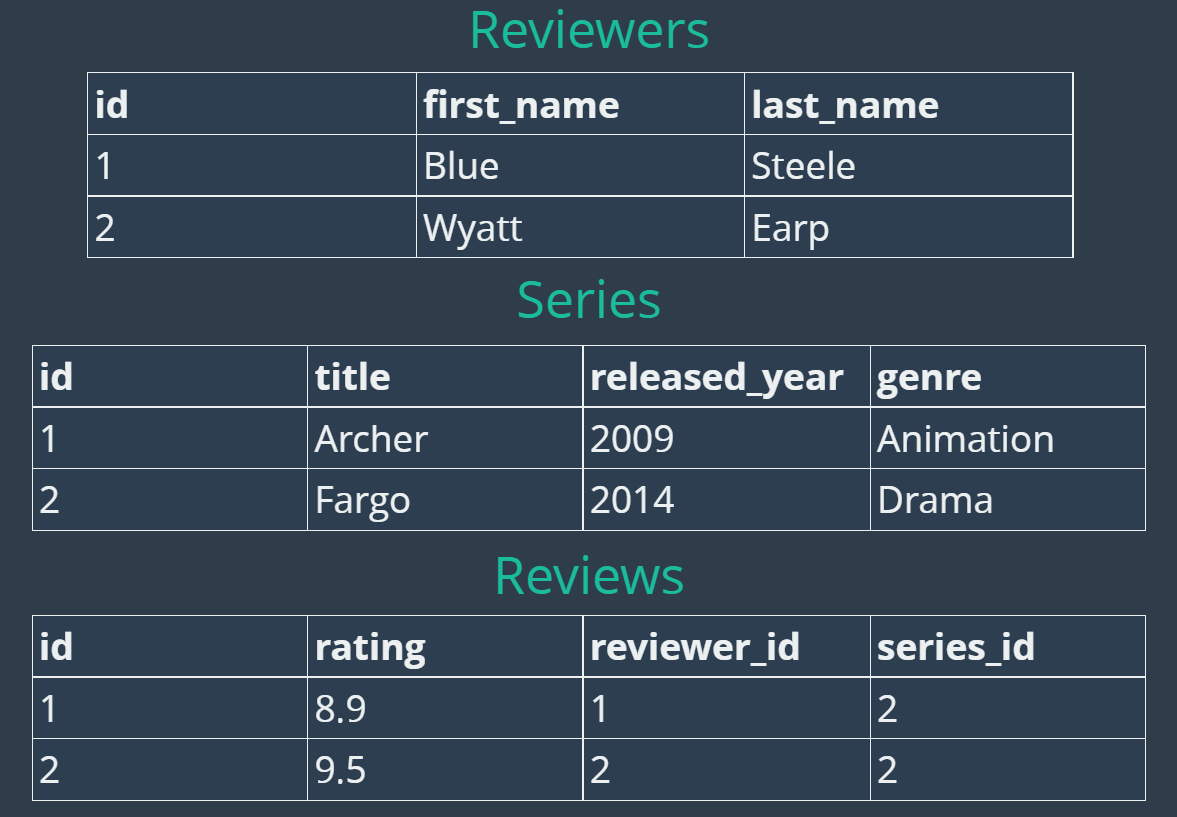
Section slides: http://webdev.slides.com/coltsteele/mysql-99-104#/36

# Many to Many Basics

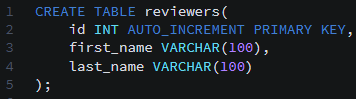
* Many to many is a trickier relationship compared to one to many
* Many to many describes a relationship in which multiple entries in one table can correspond to multiple entries in another table
  + A table of books and a table of authors. Authors can be many books, and books can be written by more than one author
  + A table of blog posts with content and a table of tags (#) for that post. Any given post can have multiple tags, and a tag can be associated with multiple posts
  + A table of students and a table of classes. Each student can be taking multiple classes, and each class can have multiple students enrolled
* For this section, we’re going to imagine building a TV show reviewing application, such as Rotten Tomatoes
  + You will have **Series** table and a **Reviewers** table
    - Each of these tables will exist on their own, and will not be connected to each other directly. The Series Data table will consist of data for each series (title, released year, genre, etc.), while the Reviewers Data table will consist of data for each reviewer (last name, first name, etc.)
  + However, we will connect these table using a **Reviews** table. This will contain columns that point to both the series and the reviewer, and will contain the rating given on each review
* Here is our schema, where the Reviews table connects the Reviewers and Series tables by pointing to the Reviewers id and Series id as foreign keys. We will need to use joins extensively in order to get data that is human-readable:



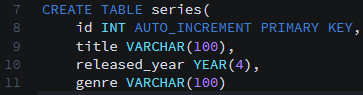


# Creating Our Tables

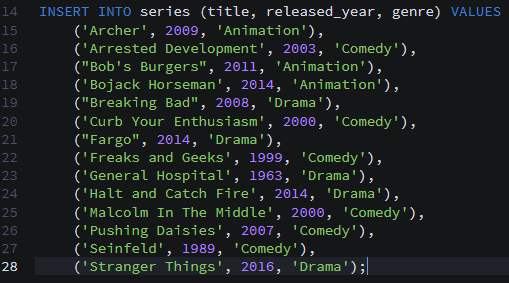
* Let’s start with creating our *reviewers* and *series* table*s*
  + Reviewers table

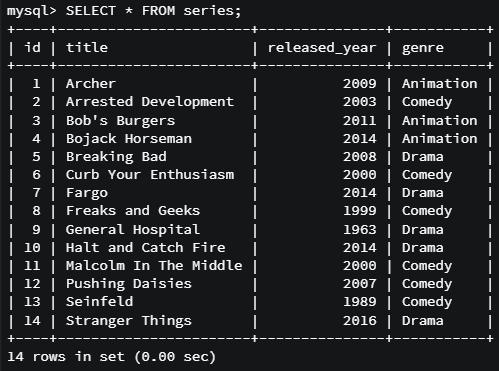


* + Series table
    - Here we will use the **YEAR** datatype for the released\_year, so that we have all the nifty functionality of years without the burden of having days and months, or the pitfalls of using INT.

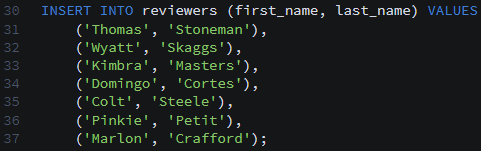


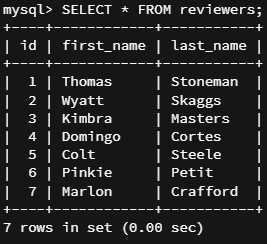
* Inserting some data
  + Series



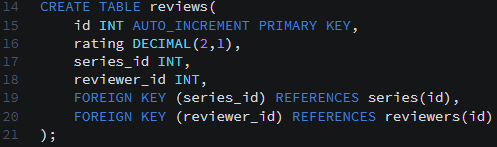


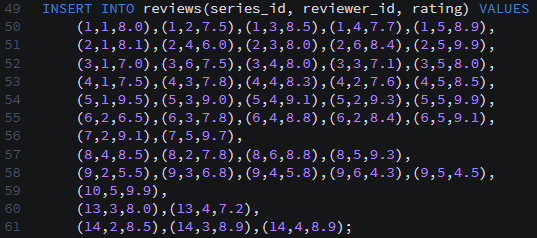
* + Reviewers

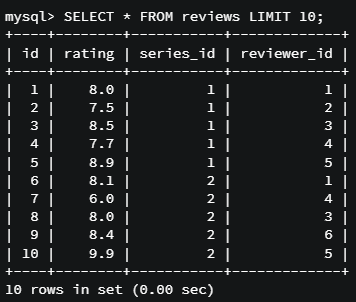




* Creating and populating the reviews table, which depends on the first two tables (as foreign keys)







* Code summary

**-- CREATING THE REVIEWERS TABLE**

CREATE TABLE reviewers (

id INT AUTO\_INCREMENT PRIMARY KEY,

first\_name VARCHAR(100),

last\_name VARCHAR(100)

);

**CREATING THE SERIES TABLE**

CREATE TABLE series(

id INT AUTO\_INCREMENT PRIMARY KEY,

title VARCHAR(100),

released\_year YEAR(4),

genre VARCHAR(100)

);

**CREATING THE REVIEWS TABLE**

CREATE TABLE reviews (

id INT AUTO\_INCREMENT PRIMARY KEY,

rating DECIMAL(2,1),

series\_id INT,

reviewer\_id INT,

FOREIGN KEY(series\_id) REFERENCES series(id),

FOREIGN KEY(reviewer\_id) REFERENCES reviewers(id)

);

**INSERTING A BUNCH OF DATA**

INSERT INTO series (title, released\_year, genre) VALUES

('Archer', 2009, 'Animation'),

('Arrested Development', 2003, 'Comedy'),

("Bob's Burgers", 2011, 'Animation'),

('Bojack Horseman', 2014, 'Animation'),

("Breaking Bad", 2008, 'Drama'),

('Curb Your Enthusiasm', 2000, 'Comedy'),

("Fargo", 2014, 'Drama'),

('Freaks and Geeks', 1999, 'Comedy'),

('General Hospital', 1963, 'Drama'),

('Halt and Catch Fire', 2014, 'Drama'),

('Malcolm In The Middle', 2000, 'Comedy'),

('Pushing Daisies', 2007, 'Comedy'),

('Seinfeld', 1989, 'Comedy'),

('Stranger Things', 2016, 'Drama');

INSERT INTO reviewers (first\_name, last\_name) VALUES

('Thomas', 'Stoneman'),

('Wyatt', 'Skaggs'),

('Kimbra', 'Masters'),

('Domingo', 'Cortes'),

('Colt', 'Steele'),

('Pinkie', 'Petit'),

('Marlon', 'Crafford');

INSERT INTO reviews(series\_id, reviewer\_id, rating) VALUES

(1,1,8.0),(1,2,7.5),(1,3,8.5),(1,4,7.7),(1,5,8.9),

(2,1,8.1),(2,4,6.0),(2,3,8.0),(2,6,8.4),(2,5,9.9),

(3,1,7.0),(3,6,7.5),(3,4,8.0),(3,3,7.1),(3,5,8.0),

(4,1,7.5),(4,3,7.8),(4,4,8.3),(4,2,7.6),(4,5,8.5),

(5,1,9.5),(5,3,9.0),(5,4,9.1),(5,2,9.3),(5,5,9.9),

(6,2,6.5),(6,3,7.8),(6,4,8.8),(6,2,8.4),(6,5,9.1),

(7,2,9.1),(7,5,9.7),

(8,4,8.5),(8,2,7.8),(8,6,8.8),(8,5,9.3),

(9,2,5.5),(9,3,6.8),(9,4,5.8),(9,6,4.3),(9,5,4.5),

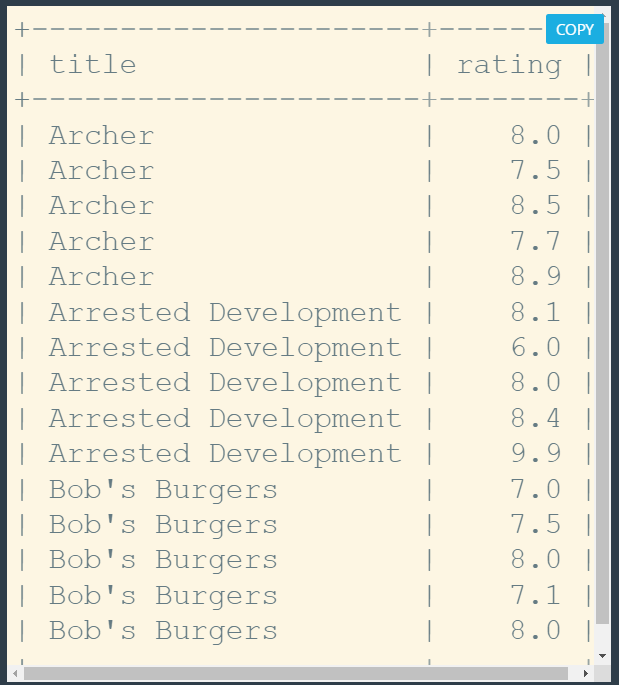
(10,5,9.9),

(13,3,8.0),(13,4,7.2),

(14,2,8.5),(14,3,8.9),(14,4,8.9);

# TV Joins Challenge 1

* Generate the following result, which shows the individual ratings of each series. Each rating is displayed, with shows with multiple reviews appearing multiple times



* Solution

SELECT

title,

rating

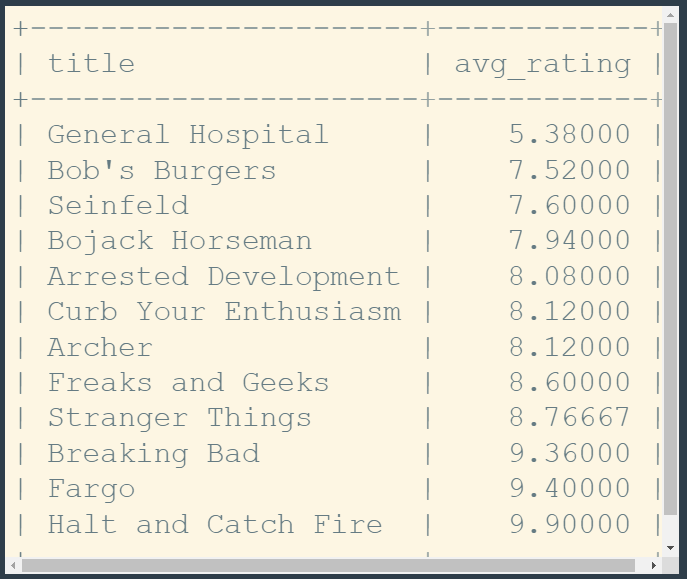
FROM series

JOIN reviews

ON series.id = reviews.series\_id;

# TV Joins Challenge 2

* Generate the following result, which shows the average rating of each series. This will require performing a GROUP BY and calculating an average



* Solution

SELECT

title,

AVG(rating) as avg\_rating

FROM series

JOIN reviews

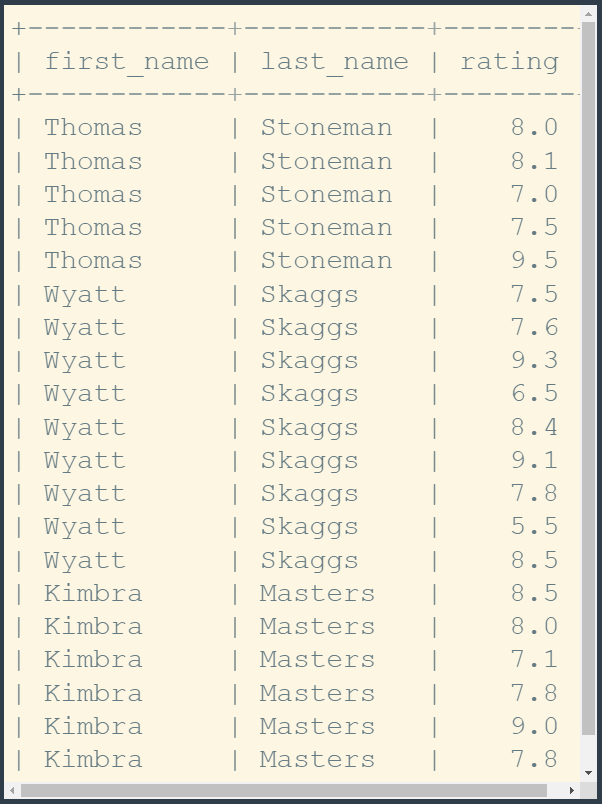
ON series.id = reviews.series\_id

GROUP BY series.id

ORDER BY avg\_rating;

# TV Joins Challenge 3

* Create the following output, with the first and last names of each reviewer and each rating they have given



* Solution

SELECT

first\_name,

last\_name,

rating

FROM reviewers

INNER JOIN reviews

ON reviewers.id = reviews.reviewer\_id;

SELECT

first\_name,

last\_name,

rating

FROM reviews

INNER JOIN reviewers

ON reviewers.id = reviews.reviewer\_id;

# TV Joins Challenge 4

* List all series that have no reviews. A couple series in our data have no reviews. We’ll need to perform a join on the *series* and *reviews* table, but ensure to include series that have no ratings. Thus, we can’t use an INNER JOIN – we’ll need a LEFT or RIGHT JOIN, depending on how you list your tables out. Then using that very property, pare down the to only those series that have no ratings.



* Solution

SELECT title AS unreviewed\_series

FROM series

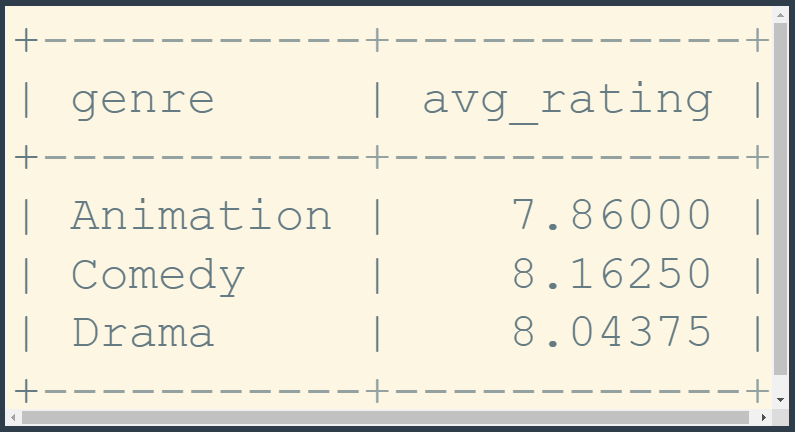
LEFT JOIN reviews

ON series.id = reviews.series\_id

WHERE rating IS NULL;

# TV Joins Challenge 5

* Generate a table showing the average rating by genre



* Solution

SELECT genre,

Round(Avg(rating), 2) AS avg\_rating

FROM series

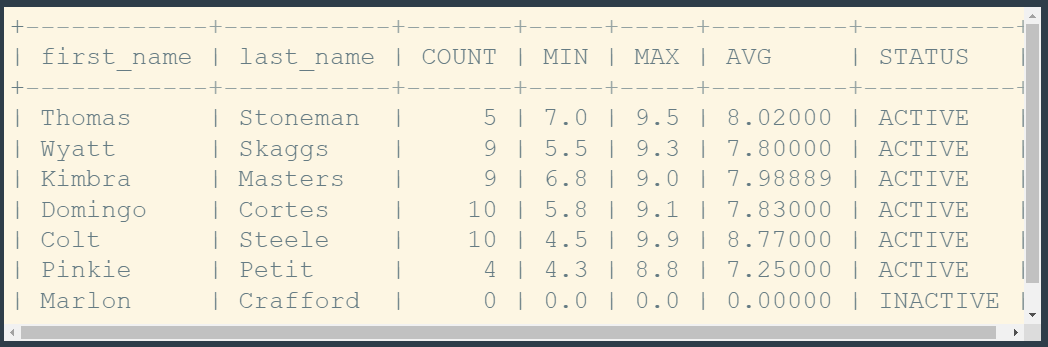
INNER JOIN reviews

ON series.id = reviews.series\_id

GROUP BY genre;

# TV Joins Challenge 6

* Generate a table that lists the first and last names of each reviewer, the total number of reviews they have written, the minimum and maximum ratings, the average ratings, and the status (Active = has written at least one review, Inactive = written no reviews)



* Solutions (different from my code)

***-- CHALLENGE 6 - Reviewer Stats***

SELECT first\_name,

last\_name,

Count(rating) AS COUNT,

Ifnull(Min(rating), 0) AS MIN,

Ifnull(Max(rating), 0) AS MAX,

Round(Ifnull(Avg(rating), 0), 2) AS AVG,

IF(Count(rating) > 0, 'ACTIVE', 'INACTIVE') AS STATUS

FROM reviewers

LEFT JOIN reviews

ON reviewers.id = reviews.reviewer\_id

GROUP BY reviewers.id;

***-- CHALLENGE 6 - Reviewer Stats With POWER USERS***

SELECT first\_name,

last\_name,

Count(rating) AS COUNT,

Ifnull(Min(rating), 0) AS MIN,

Ifnull(Max(rating), 0) AS MAX,

Round(Ifnull(Avg(rating), 0), 2) AS AVG,

CASE

WHEN Count(rating) >= 10 THEN 'POWER USER'

WHEN Count(rating) > 0 THEN 'ACTIVE'

ELSE 'INACTIVE'

end AS STATUS

FROM reviewers

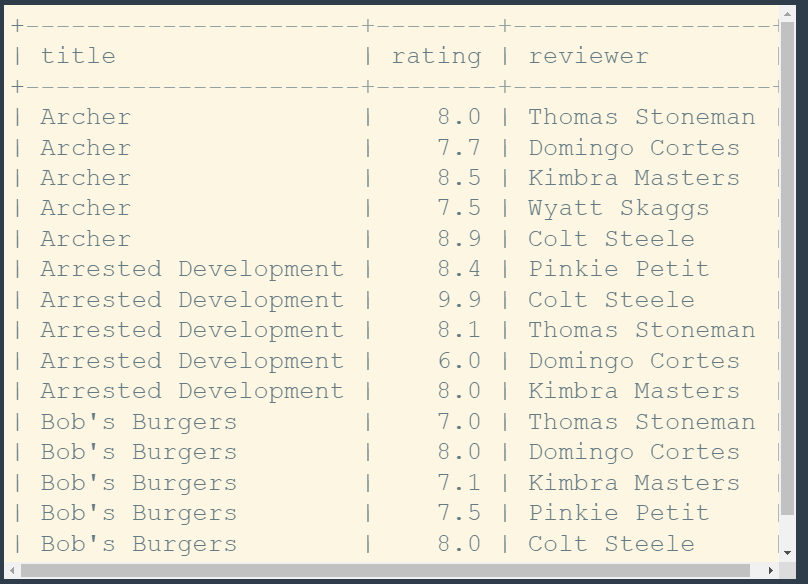
LEFT JOIN reviews

ON reviewers.id = reviews.reviewer\_id

GROUP BY reviewers.id;

# TV Joins Challenge 7

* By joining all three tables, create a table with the titles, ratings, and reviewers



* Solution (slightly different from my solution)

SELECT

title,

rating,

CONCAT(first\_name,' ', last\_name) AS reviewer

FROM reviewers

INNER JOIN reviews

ON reviewers.id = reviews.reviewer\_id

INNER JOIN series

ON series.id = reviews.series\_id

ORDER BY title;