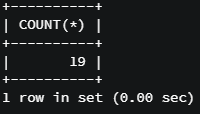
Section slides: <http://webdev.slides.com/coltsteele/mysql-99-101#/80>

* This section will focus on aggregate functions, such as counting, averaging, summing, etc., all of which involves grouping our data in some way
* So rather than working with individual rows, we can group items and perform more interesting calculations on them
  + For example, we can find the average number of sales per year, where we group items by year
  + Or we can group books by author and determine the average number of pages in books written by each author

The COUNT Function

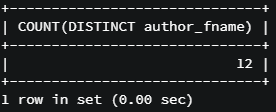
* This function counts whatever you want it to count
* <https://www.w3schools.com/Sql/func_mysql_count.asp>
* We can count the number of books in our database as follows:
  + Within the syntax, the asterisk \* simply means you want to count the number of items, whether it be the number of rows in the table, or the number of “super rows” that are formed after performing GROUP BY (more on that later)





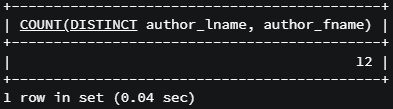
* We can also use it to ask questions such as, how many *unique* author\_fnames are in our database?
  + To solve this, we can combine COUNT and DISTINCT



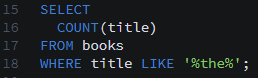


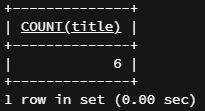
* An edge case shows itself when we try to find the number of *unique* authors in our books table, where we select author\_lname, author\_fname using DISTINCT.
  + This shows that we have 12 unique selections of author\_lname and author\_fname





* What if we want to find how many titles contain the word “the”? To accomplish this, we can use a combination of COUNT with WHERE and LIKE





* Code summary for the COUNT function

SELECT COUNT(\*) FROM books;

SELECT COUNT(author\_fname) FROM books;

SELECT COUNT(DISTINCT author\_fname) FROM books;

SELECT COUNT(DISTINCT author\_lname) FROM books;

SELECT COUNT(DISTINCT author\_lname, author\_fname) FROM books;

SELECT title FROM books WHERE title LIKE '%the%';

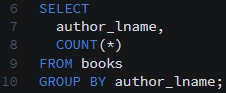
SELECT COUNT(\*) FROM books WHERE title LIKE '%the%';

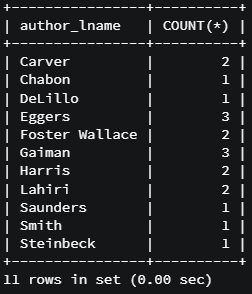
The GROUP BY Function

* GROUP BY is an aggregate function that works a bit differently than the others that we’ve seen.
  + It does not provide an immediate result. You must use it alongside an aggregate function
  + GROUP BY summarizes or aggregates identical data into single rows
  + It allows us to group things by particular properties and values, and we can use these groupings to do even more analyses
* Check out the syntax below, where we grouping by author\_lname

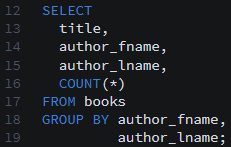


* + If you try this, you will just see a list of unique author\_lnames, alongside just ONE of their book titles
    - In newer versions of MySQL, you’ll get an error since GROUP BY is not really intended to be used this way
    - This is because only\_full\_group\_by is enabled
  + But in the background, the method is actually grouping all of the titles that share the same author into a “super row” underneath one author name. They are grouped together, we just can’t see them
* Suppose we want to count how many books are attributed to each unique author\_lname. We can do the following:
  + This means “give me two columns, the first having author\_lname, and the second having the number of rows under that author\_lname



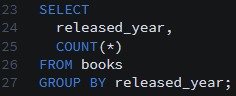


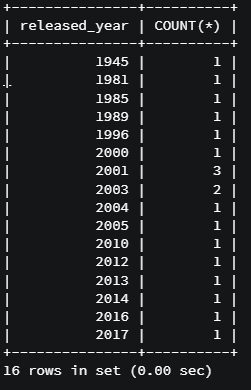
* + We see an issue here, which is that the query is telling us that the author whose last name is Harris has two books. In reality, there are two authors with the last name Harris (Dan and Freida), and they each have one book. We can solve this using the following selection



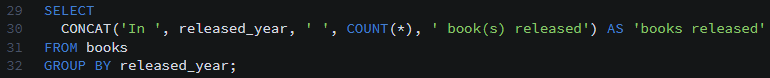


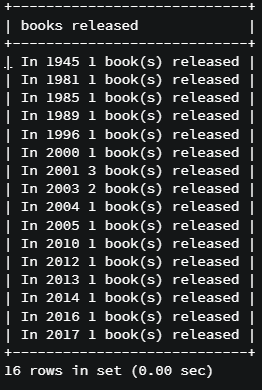
* We can also group by release year and figure out how many books were released each year! Here’s how:





* + We see that 2003 had the most released books out of all years in our table
  + Let’s make it more fancy by throwing in some CONCAT





* Code summary, GROUP BY

SELECT title, author\_lname FROM books;

SELECT title, author\_lname FROM books

GROUP BY author\_lname;

SELECT author\_lname, COUNT(\*)

FROM books GROUP BY author\_lname;

SELECT title, author\_fname, author\_lname FROM books;

SELECT title, author\_fname, author\_lname FROM books GROUP BY author\_lname;

SELECT author\_fname, author\_lname, COUNT(\*) FROM books GROUP BY author\_lname;

SELECT author\_fname, author\_lname, COUNT(\*) FROM books GROUP BY author\_lname, author\_fname;

SELECT released\_year FROM books;

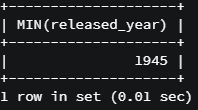
SELECT released\_year, COUNT(\*) FROM books GROUP BY released\_year;

SELECT CONCAT('In ', released\_year, ' ', COUNT(\*), ' book(s) released') AS year FROM books GROUP BY released\_year;

The Basics of MIN and MAX

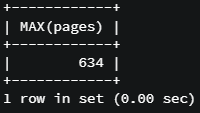
* As you might imagine, the MIN and MAX functions help you identify the minimum or maximum values in a table.
  + Can be used on their own, or can be combined with GROUP BY
* On their own, we can do things like finding the minimum released year of book release





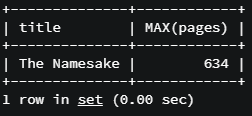
* + Thus, 1945 was the earliest year that a book in our database was released
* Using MAX, we can find the book with the highest number of pages in our table





* Suppose we wanted the title of book with the largest number of pages. To do this, we could try the following:



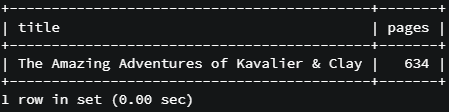


* + This is not entirely correct. While the longest book does have 634 pages, that book is NOT The Namesake.



* + There are multiple ways around this. One way is to disregard MAX altogether and use ORDER BY and LIMIT





* + We’ll see other ways in the coming lectures
* Code summary: MIN and MAX basics

SELECT MIN(released\_year)

FROM books;

SELECT MIN(released\_year) FROM books;

SELECT MIN(pages) FROM books;

SELECT MAX(pages)

FROM books;

SELECT MAX(released\_year)

FROM books;

SELECT MAX(pages), title

FROM books;

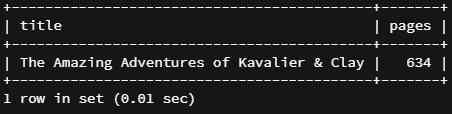
SUBQUERIES – A Problem with Min and Max

* In the previous lecture we saw that the follow code does not return what we want. It returns the *first* title in the table, and then returns the max number of pages. The two are independent from each other



* We also saw a workaround above, in which we sorted the books by pages and selected (via LIMIT) out the book with the largest number of pages
* But what if we want a solution that utilizes the MAX function? We can do that as well, as the following **subquery** shows
  + A subquery is simply one query nested within another. We’ll see more of this later
  + The “lowest” level query will execute first, followed by the next level up.
  + In this case, the query will first find the largest number of pages, then enter that value into “WHERE pages =”. Then the higher level query will run, selecting the specific title and pages associated with that WHERE condition (the largest number of pages)





* One issue with subqueries is that it can be slow, since the system is essentially running two queries. For that reason, the ORDER BY and ASC LIMIT approach is actually faster and gives the same result
* Code summary

SELECT \* FROM books

WHERE pages = (SELECT Min(pages)

FROM books);

SELECT title, pages FROM books

WHERE pages = (SELECT Max(pages)

FROM books);

SELECT title, pages FROM books

WHERE pages = (SELECT Min(pages)

FROM books);

SELECT \* FROM books

ORDER BY pages ASC LIMIT 1;

SELECT title, pages FROM books

ORDER BY pages ASC LIMIT 1;

SELECT \* FROM books

ORDER BY pages DESC LIMIT 1;