Section Slides: <http://webdev.slides.com/coltsteele/mysql-99-101#/42>

# Adding A Couple New Books

* For this section we will add a few new books to our table, to further illustrate the additional tools we’ll be learning
* Code summary

INSERT INTO books

(title, author\_fname, author\_lname, released\_year, stock\_quantity, pages)

VALUES ('10% Happier', 'Dan', 'Harris', 2014, 29, 256),

('fake\_book', 'Freida', 'Harris', 2001, 287, 428),

('Lincoln In The Bardo', 'George', 'Saunders', 2017, 1000, 367);

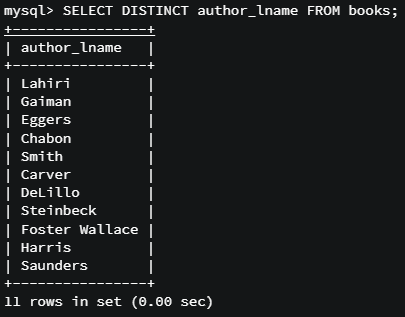
SELECT title FROM books;

# Using DISTINCT

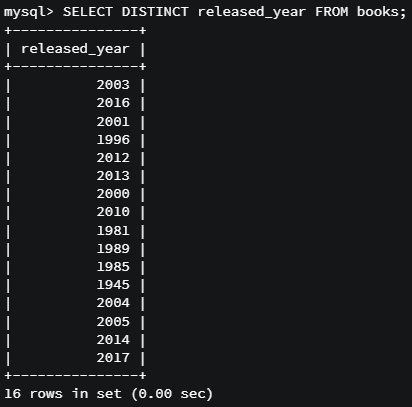
* The **DISTINCT** function is used in conjunction with SELECT to return only distinct (unique) entries in a selection
* In our *books* table, some authors are present multiple times because they have more than one title in our table



* + If we just want unique author last names, we use SELECT **DISTINCT**.

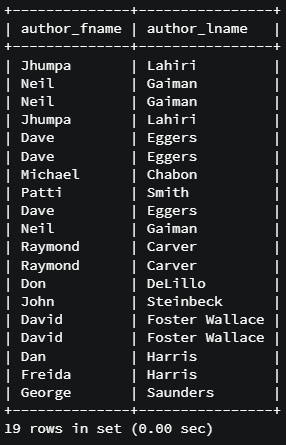


* Another example, where we select the distinct years of publication. This example shows how this function works with integers/numbers in addition to strings



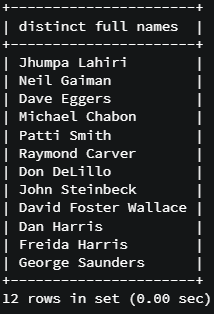
* An advanced topic, there are two authors with the last name “Harris” in our table.





* + If we were to select last names only, we would only get one “Harris” back because they are the same last night
  + But what if we wanted distinct FULL names? Can we combine two columns and have the method resolve distinctions among them? Yes, we can!





* Code summary

SELECT author\_lname FROM books;

SELECT DISTINCT author\_lname FROM books;

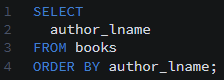
SELECT author\_fname, author\_lname FROM books;

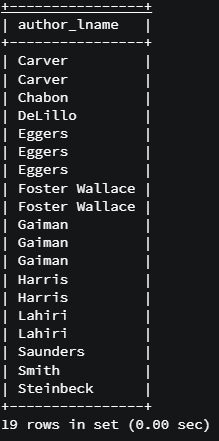
SELECT DISTINCT CONCAT(author\_fname,' ', author\_lname) FROM books;

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

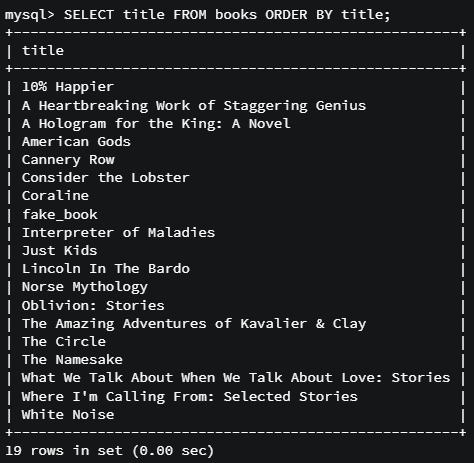
# Sorting Data with ORDER BY

* The **ORDER BY** function allows us to sort our data by specified criteria, perhaps in alphabetical order, or increasing or decreasing value
* Example: This sorts our data in alphabetical order by author last name
  + By default this is sorted in *ascending* order

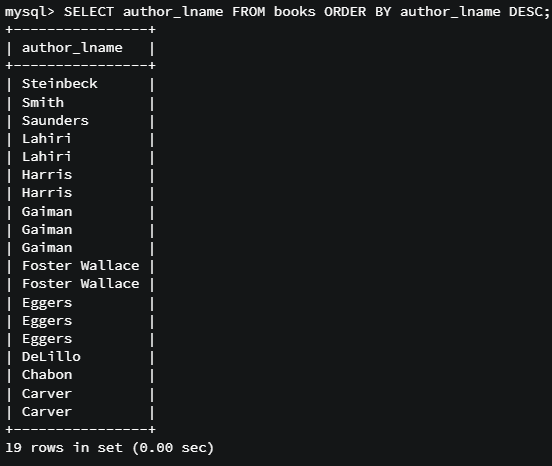




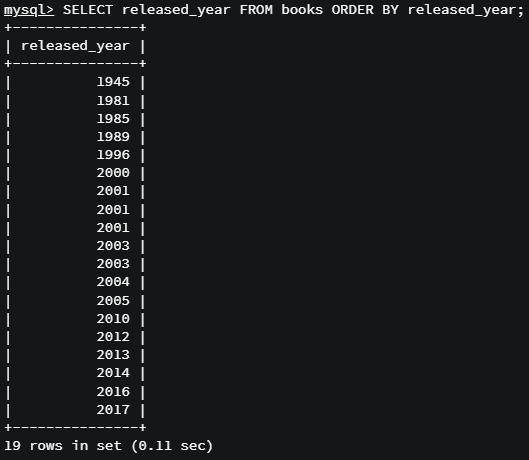
* Another example with titles, where we have both letters and numbers in our titles (though they are all VARCHAR)



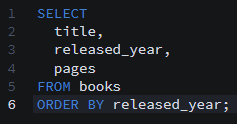
* + Notice how the numbers come before letters in alphabetical order
* We can change the order to *descending* by adding **DESC** to the end of the query

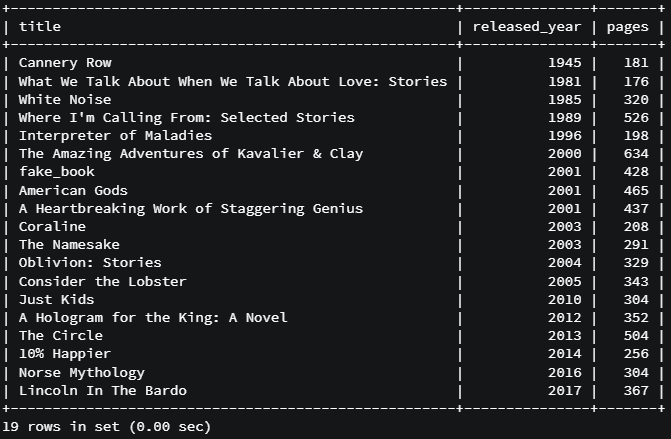


* Sorting, obviously, works numerically as well

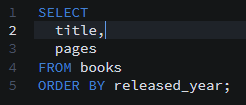


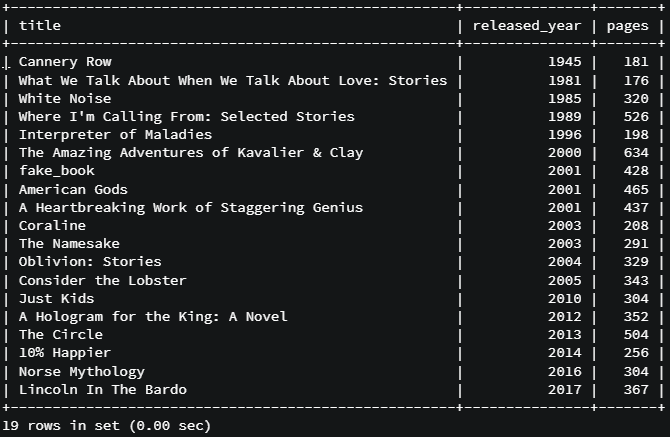
* We don’t have to order by the property that we’re selecting. We can select multiple properties (columns) for our data that are ordered by one of those properties (column), or even by the property that you’re not selecting
  + For example, in our books table we can select the title, released year, and pages, ordered by released year



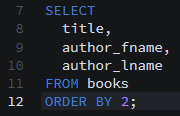


* + We can even leave the selection for released\_year out entirely, but still order by it



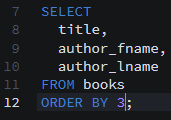


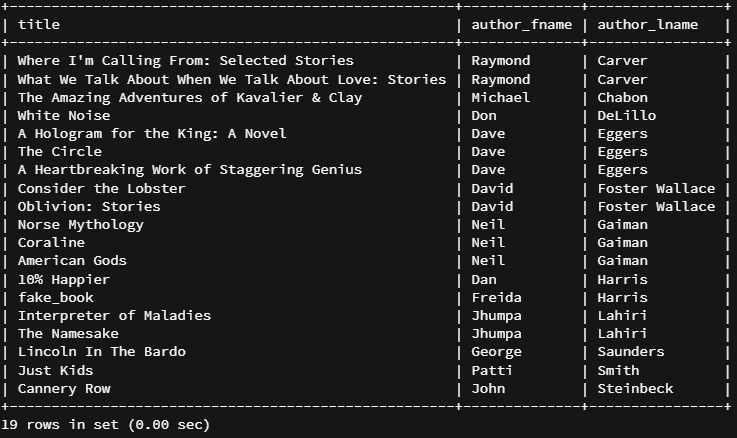
* A quirk to consider – take a look at this query. Here, **ORDER BY 2** is simply a shortcut indicating that we want to order by the second property that we’re selecting by.



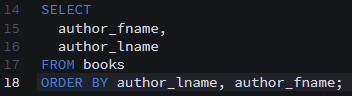


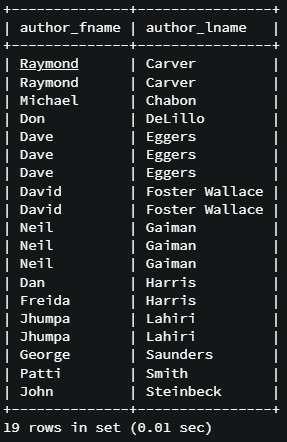
* + We can easily change this to a different property by just changing the number





* + Order matters here – the order by number is agnostic to the actual property that it is sorting by. If you declare the selections in a different order, then it will select the property in that position
* What about **ordering by multiple columns?** This is something that is done, albeit relatively infrequently. In this example, the names will be sorted first by author last name, then by author first name
  + In this particular example, notice how Dan Harris and Freida Harris share the same last name. However, with this dual sorting, Dan will be listed before Freida since “D” comes before “F” in the alphabet





* Code summary: ORDER BY

SELECT author\_lname FROM books;

SELECT author\_lname FROM books ORDER BY author\_lname;

SELECT title FROM books;

SELECT title FROM books ORDER BY title;

SELECT author\_lname FROM books ORDER BY author\_lname DESC;

SELECT released\_year FROM books;

SELECT released\_year FROM books ORDER BY released\_year;

SELECT released\_year FROM books ORDER BY released\_year DESC;

SELECT released\_year FROM books ORDER BY released\_year ASC;

SELECT title, released\_year, pages FROM books ORDER BY released\_year;

SELECT title, pages FROM books ORDER BY released\_year;

SELECT title, author\_fname, author\_lname

FROM books ORDER BY 2;

SELECT title, author\_fname, author\_lname

FROM books ORDER BY 3;

SELECT title, author\_fname, author\_lname

FROM books ORDER BY 1;

SELECT title, author\_fname, author\_lname

FROM books ORDER BY 1 DESC;

SELECT author\_lname, title

FROM books ORDER BY 2;

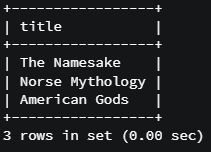
SELECT author\_fname, author\_lname FROM books

ORDER BY author\_lname, author\_fname;

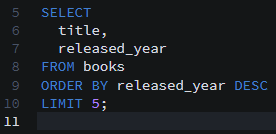
# Using LIMIT

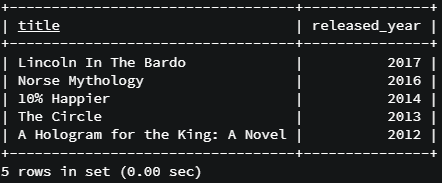
* The **LIMIT** modifier allows us to specify the number of results that we want to select. By default, a selection will include all rows that meet the selection criteria
* Example from our *books* table, where we select only the first 3 entries
  + The LIMIT modifier will display the “first 3” whether they are sorted or not





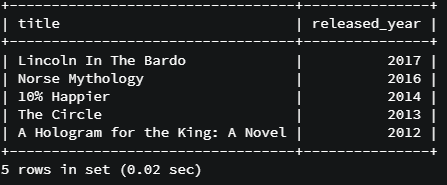
* LIMIT becomes much more useful when used alongside sorting via ORDER BY
* In this example, we want the 5 most recently-released books, in descending order. We can do the following





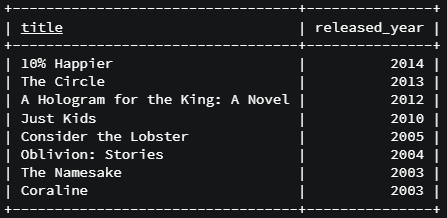
* Another use case is to specify the starting point of your selection limit, followed by how many items you want to list starting from there. In the code below, we want to start from the first (zeroth) row and list five rows of data. It functionally does the same thing as our previous example above.
  + Notice how the indexing of rows is different from the indexing of strings. In MySQL, row indexes start at 0, while string indexes start at 1.





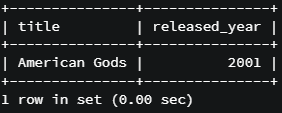
* + Another example using different limits:





* + An uncommon use case is to identify an exact row by using a range of 1. The example below will simply give us the 11th row (indexed at position 10) as follows:





* There is not elegant way to specify that you want to select all the way to the end of the table. Instead, you have to use a large number for the second parameter:
  + Basic syntax: **SELECT \* FROM tbl LIMIT 95,18446744073709551615;**
  + Example:





* Code summary, LIMIT

SELECT title FROM books LIMIT 3;

SELECT title FROM books LIMIT 1;

SELECT title FROM books LIMIT 10;

SELECT \* FROM books LIMIT 1;

SELECT title, released\_year FROM books

ORDER BY released\_year DESC LIMIT 5;

SELECT title, released\_year FROM books

ORDER BY released\_year DESC LIMIT 1;

SELECT title, released\_year FROM books

ORDER BY released\_year DESC LIMIT 14;

SELECT title, released\_year FROM books

ORDER BY released\_year DESC LIMIT 0,5;

SELECT title, released\_year FROM books

ORDER BY released\_year DESC LIMIT 0,3;

SELECT title, released\_year FROM books

ORDER BY released\_year DESC LIMIT 1,3;

SELECT title, released\_year FROM books

ORDER BY released\_year DESC LIMIT 10,1;

SELECT \* FROM tbl LIMIT 95,18446744073709551615;

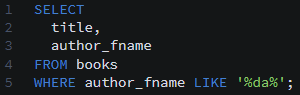
SELECT title FROM books LIMIT 5;

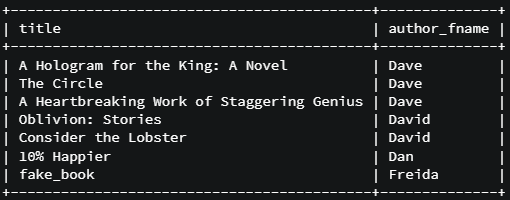
SELECT title FROM books LIMIT 5, 123219476457;

SELECT title FROM books LIMIT 5, 50;

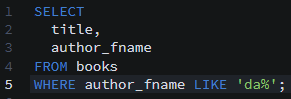
# Using LIKE for Better Searching

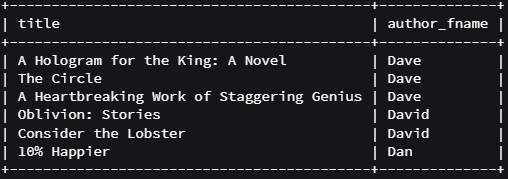
* The **LIKE** method is very useful and allows us to make more flexible searches
  + This contrasts with WHERE, in which we could only find exact matches
* This example pseudo query will find author last names that contain the letters “da”. The percent symbol (“%”) is a **wildcard**, meaning it can be any character or set of characters.
  + WHERE author\_fname LIKE '%da%'
  + Using actual code:



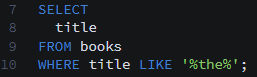


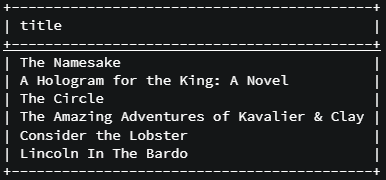
* What if we wanted only names that *start with “Da”*? We just modify our query slightly and remove the first wildcard.
  + Notice that “Freida” is now gone as it no longer matches the pattern
  + The search is case-insensitive





* Another example: selecting titles with the word “the” in it





* Code summary:

SELECT title, author\_fname FROM books WHERE author\_fname LIKE '%da%';

SELECT title, author\_fname FROM books WHERE author\_fname LIKE 'da%';

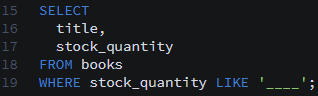
SELECT title FROM books WHERE title LIKE 'the';

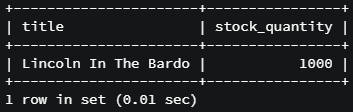
SELECT title FROM books WHERE title LIKE '%the';

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

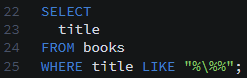
# The Underscore Wildcard

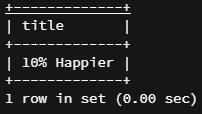
* The underscore character is a wildcard that specifies *exactly one character*. It can be any character, but only one of them
* Example: Suppose we want to select books where our stock quantities are four digits long. We can do the following:





* What happens if you’re looking for something with an actual underscore or an actual % sign in it? In this case, we *escape* the wildcard by using a backslash (“\”)
  + The backslash means you are literally looking for the symbol that follows, in MySQL it is usually a “%” or “\_”





* Code summary:

SELECT title, stock\_quantity FROM books;

SELECT title, stock\_quantity FROM books WHERE stock\_quantity LIKE '\_\_\_\_';

SELECT title, stock\_quantity FROM books WHERE stock\_quantity LIKE '\_\_';

(235)234-0987 LIKE '(\_\_\_)\_\_\_-\_\_\_\_'

SELECT title FROM books;

SELECT title FROM books WHERE title LIKE '%\%%'

SELECT title FROM books WHERE title LIKE '%\\_%'