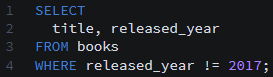
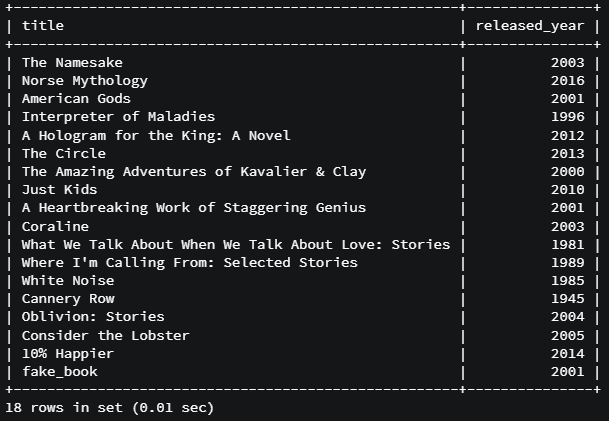
Section slides: <http://webdev.slides.com/coltsteele/mysql-99-103>

* Operators allow us to further refine our selections by narrowing down the selection criteria in specific ways
* We’ll do things like select birthdays between two defined years, or selecting all items in stock and priced below a certain price

# NOT EQUAL

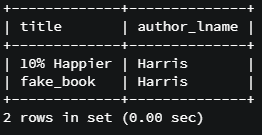
* The **not equal** operator, **“!=”**, is used to *exclude* the defined values
  + A fairly univers
* Documetation: <https://www.w3resource.com/mysql/comparision-functions-and-operators/not-equal-operator.php>
* It’s very similar to using the “=” operator, but it does the opposite
* Let’s seen an example where we *select the titles of all books that were NOT released in 2017*





* Another example where we exclude particular authors
  + To motivate the idea, we first select all books by authors with the last name “Harris”





* + Now we simply flip the switch and use the NOT EQUAL operator to select titles if books by all authors except those with the last name “Harris”





* Code summary

SELECT title FROM books WHERE released\_year = 2017;

SELECT title FROM books WHERE released\_year != 2017;

SELECT title, author\_lname FROM books;

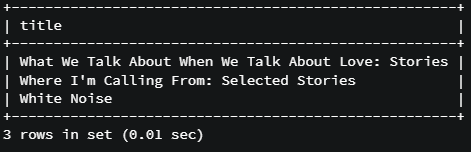
SELECT title, author\_lname FROM books WHERE author\_lname = 'Harris';

SELECT title, author\_lname FROM books WHERE author\_lname != 'Harris';

# NOT LIKE

* The NOT LIKE function is the opposite of LIKE. Recall that “LIKE” allows us to match patterns, usually in strings
* As a refresher, let’s do a LIKE selection for all books that start with “W”





* Now we’ll try a NOT LIKE selection. Here, we’ll select all titles that DO NOT start with the letter “W”
  + Notice how this does not use the exclamation point operator. We simply say “NOT LIKE”





* Code summary

SELECT title FROM books WHERE title LIKE 'W';

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

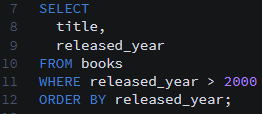
SELECT title FROM books WHERE title LIKE '%W%';

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

SELECT title FROM books WHERE title NOT LIKE 'W%';

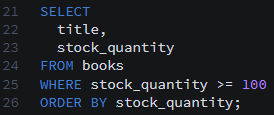
# GREATER THAN and GREATER THAN OR EQUAL TO

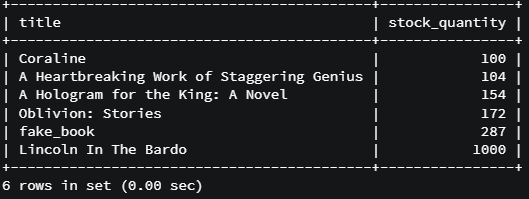
* The greater than operator is the “>” symbol. It is also fairly universal across programming languages
* With this operator, we can do things like *select all books released after the year 2000*





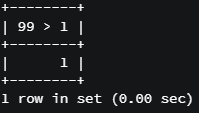
* The **greater than or equal to** operator, represented by “>=”, is inclusive of the defining value.
* Let’s do an example where we want to *select all books with stock of greater than or equal to 100*





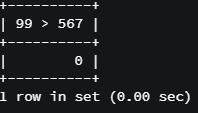
* What happens if we try the following?





* + What returns is the Boolean value of 1, representing “True”. Since 99 is in fact greater than 1, this selection will return “1”, which is equivalent to “True”.
  + To verify, let’s write a false statement. This will return “0”, equivalent to “False”





* Exercise: What will these evaluate to?
  + 100 > 5
    - 1 (True)
  + -15 > 15
    - 0 (False)
  + 9 > -10
    - 1 (True)
  + 1 > 1
    - 0 (False)
  + ‘a’ > ‘b’
    - 0 (False)
  + ‘A’ > ‘a’
    - 0 (False)
    - A and a are actually equivalent in MySQL. We’ve seen this in instances were we select text in a case-insensitive manner
* It is best to avoid logical operators on strings because it is confusing and inconsistent between different languages
* Code summary

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

SELECT title, released\_year FROM books

WHERE released\_year > 2000 ORDER BY released\_year;

SELECT title, released\_year FROM books

WHERE released\_year >= 2000 ORDER BY released\_year;

SELECT title, stock\_quantity FROM books;

SELECT title, stock\_quantity FROM books WHERE stock\_quantity >= 100;

SELECT 99 > 1;

SELECT 99 > 567;

100 > 5

-- true

-15 > 15

-- false

9 > -10

-- true

1 > 1

-- false

'a' > 'b'

-- false

'A' > 'a'

-- false

'A' >= 'a'

-- true

SELECT title, author\_lname FROM books WHERE author\_lname = 'Eggers';

SELECT title, author\_lname FROM books WHERE author\_lname = 'eggers';

SELECT title, author\_lname FROM books WHERE author\_lname = 'eGGers';

# LESS THAN and LESS THAN OR EQUAL TO

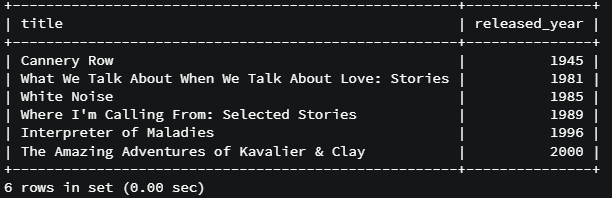
* The LESS THAN operator “<” works similarly to GREATER THAN
* We can do things like *select* *all books that were released before the year 2000*





* The LESS THAN OR EQUAL TO operator, “<=”, also functions similarly. If we wanted to include books published in 2000, we could do the following





* Exercises: What will these evaluate to?
  + 3 < -10
    - False
  + -10 < -9
    - True
  + 42 <= 42
    - True
  + ‘h’ < ‘p’
    - True
    - This is because ‘h’ comes before ‘p’ and both are lower-case
  + ‘Q’ <= ‘q’
    - True
    - Remember that case does not matter in MySQL
* Code summary

SELECT title, released\_year FROM books;

SELECT title, released\_year FROM books

WHERE released\_year < 2000;

SELECT title, released\_year FROM books

WHERE released\_year <= 2000;

SELECT 3 < -10;

-- false

SELECT -10 < -9;

-- true

SELECT 42 <= 42;

-- true

SELECT 'h' < 'p';

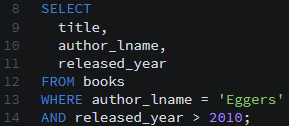
-- true

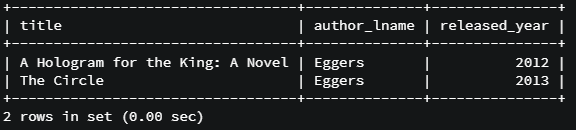
SELECT 'Q' <= 'q';

-- true

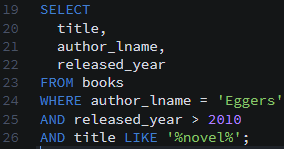
# Logical AND

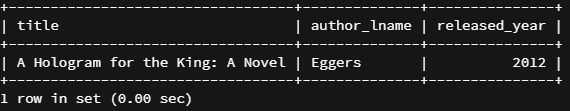
* The logical AND (“AND” or “&&”) necessitates that selections satisfy all conditions
  + The “&&” operator is deprecated and will be removed from a future version of MySQL
* Documentation: <https://dev.mysql.com/doc/refman/8.0/en/logical-operators.html#operator_and>
* For example, suppose we want to *select books written by Dave Eggers published after the year 2010*. How would we do this?
  + This is a compound selection, so we need to use two conditions and the AND operator





* AND logical exercises
  + 1 < 5 && 7 = 9
    - False
  + -10 > -20 && 0 <= 0
    - True
  + -40 <= -0 AND 10 > 40
    - False
  + 54 <= 54 && ‘a’ = ‘A’
    - True
* The AND operator is NOT limited to just 2 components. There is no practical upper limit, though we generally will use only 2 or 3
* Example – *select all books whose author’s last name is Eggers, was released after 2010, and has the world “novel” in the title*





* Code Summary

SELECT title, author\_lname, released\_year FROM books

WHERE author\_lname='Eggers';

SELECT title, author\_lname, released\_year FROM books

WHERE released\_year > 2010;

SELECT

title,

author\_lname,

released\_year FROM books

WHERE author\_lname='Eggers'

AND released\_year > 2010;

SELECT 1 < 5 && 7 = 9;

-- false

SELECT -10 > -20 && 0 <= 0;

-- true

SELECT -40 <= 0 AND 10 > 40;

--false

SELECT 54 <= 54 && 'a' = 'A';

-- true

SELECT \*

FROM books

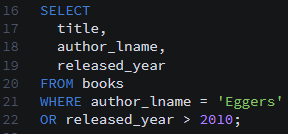
WHERE author\_lname='Eggers'

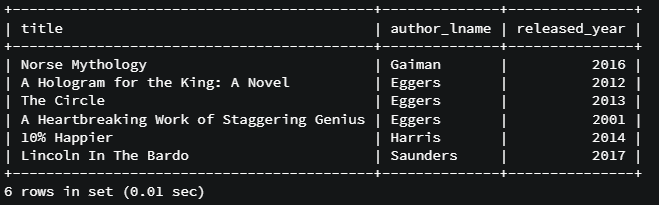
AND released\_year > 2010

AND title LIKE '%novel%';

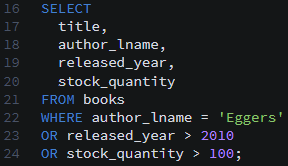
# Logical OR

* The logical OR operator can be represented either by the work “OR” or by the double pipes “||”
  + “||” is deprecated as of MySQL 8.0.17
* Documentation: <https://dev.mysql.com/doc/refman/8.0/en/logical-operators.html#operator_or>
* The logical OR operator tests all conditions on all sides of the OR operator. If *any* of the conditions resolves to True, then the entire statement will resolve to True
* Let’s do a quick example where we select *books whose author’s last name is “Eggers” OR was released after the year 2010*





* + We can clearly see how this works: if either condition is satisfied, that line of data will be included in the selection. We have some books before BEFORE 2010 but selected because Dave Eggers is the author, and we have some books whose author is not Eggers but were released after 2010 and were thus selected
* Logical exercises with OR
  + 40 <= 100 || -2 > 0
    - True
  + 10 > 5 || 5 = 5
    - True
  + ‘a’ = 5 || 3000 > 2000
    - True
* We can chain as many OR’s together as we want. In the example below, for each of the selections, at least one of the conditions is true.





* Code summary

SELECT

title,

author\_lname,

released\_year

FROM books

WHERE author\_lname='Eggers' || released\_year > 2010;

SELECT 40 <= 100 || -2 > 0;

-- true

SELECT 10 > 5 || 5 = 5;

-- true

SELECT 'a' = 5 || 3000 > 2000;

-- true

SELECT title,

author\_lname,

released\_year,

stock\_quantity

FROM books

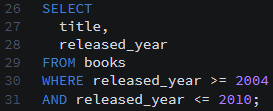
WHERE author\_lname = 'Eggers'

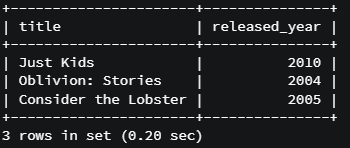
|| released\_year > 2010

OR stock\_quantity > 100;

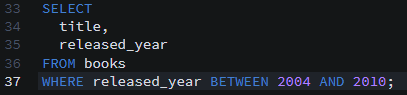
# Between

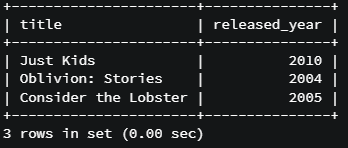
* BETWEEN allows us to make selections based on two bounds
  + This is an alternative to using greater than, less than, and AND
* Examples: <https://www.w3schools.com/mysql/mysql_between.asp>
* Documentation: <https://dev.mysql.com/doc/refman/8.0/en/comparison-operators.html#operator_between>
* Suppose we want to *select all books published between 2004 and 2015.*
  + First, the old school way using comparative logicals and the AND operator



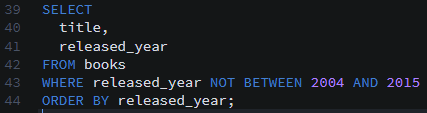


* + Now let’s try the BETWEEN operator to accomplish the exact same task





* The BETWEEN operator is always paired with an AND operator
* BETWEEN is inclusive – both the beginning and ending values are included in the selection
  + If you don’t want it to be inclusive, you’ll have to just change the bounds
* NOT BETWEEN is also a thing – it does the opposite of between by giving you the inverse of the values between the beginning and end values
* Example: *Select books whose released\_year is not between 2004 and 2015*. Basically, select all books outside that range.



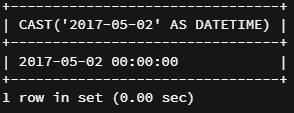


* A special note about comparing **dates**: the documentation suggests using the CAST() function to convert the date values to the desired data type
  + In other words, try to avoid comparing a datetime to a date, or a datetime to a time. It’s always best to work with the same datatype

## How does CAST() work?

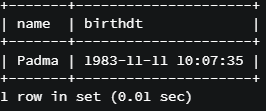
* Documentation: <https://dev.mysql.com/doc/refman/8.0/en/cast-functions.html#function_cast>
* **CAST(*expr*, *type*)** takes an expression of any type and returns a result value of the specified type
* Trivial example:



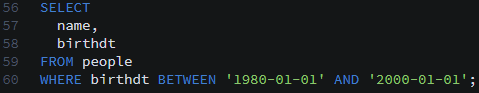


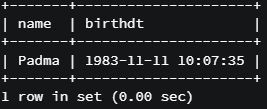
* Returning to our *people* table under the “new\_testing\_db” database, let’s use CAST() on our birthdt data. Suppose we want to *select people whose birthdt is between January 1, 1980 and January 1, 2000*





* + Note that MySQL has some failsafe coding that allows the non-casted string dates to work as well. However, it is always “safer” to cast your data into the desired datatypes.





* Code summary

SELECT title, released\_year FROM books WHERE released\_year >= 2004 && released\_year <= 2015;

SELECT title, released\_year FROM books

WHERE released\_year BETWEEN 2004 AND 2015;

SELECT title, released\_year FROM books

WHERE released\_year NOT BETWEEN 2004 AND 2015;

SELECT CAST('2017-05-02' AS DATETIME);

show databases;

use new\_testing\_db;

SELECT name, birthdt FROM people WHERE birthdt BETWEEN '1980-01-01' AND '2000-01-01';

SELECT

name,

birthdt

FROM people

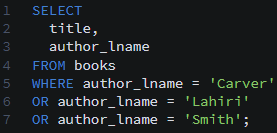
WHERE

birthdt BETWEEN CAST('1980-01-01' AS DATETIME)

AND CAST('2000-01-01' AS DATETIME);

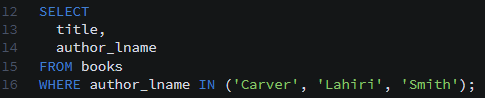
# IN and NOT IN

* This function allows us to provide a set of values and then make a selection based on whether a given column is in that set
* Examples: <https://www.w3schools.com/mysql/mysql_in.asp>
* Let’s illustrate with an example: *select all books written by Carber or Lahiri or Smith*
  + First, remember that we could do that with the OR logical operator



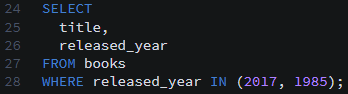


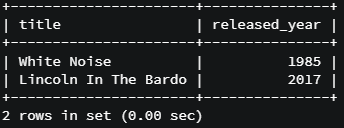
* + But we can also do this using IN, where the syntax is much shorter and easier



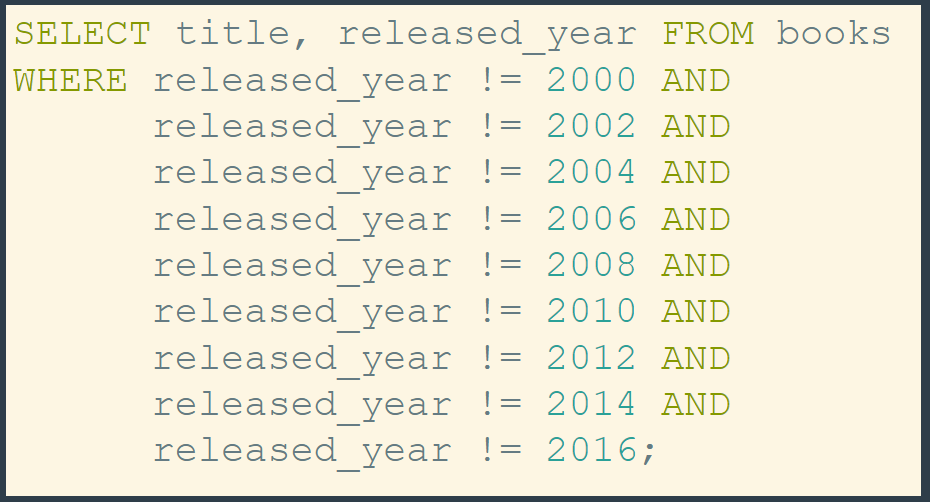


* Another example using *released\_year*

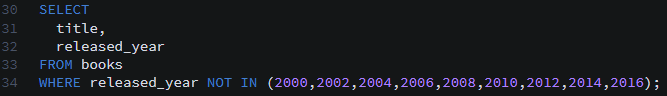




* The NOT IN operator does the opposite of IN by returning items that are NOT in the provided set
* As an example, let’s *select all books not published in any even-numbered year from 2000 through 2016 (even-numbered years before or after 2016 are acceptable)*
  + One way to do this is to use a long series of NOT EQUAL operators coupled with AND operators:

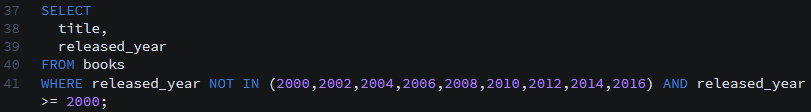


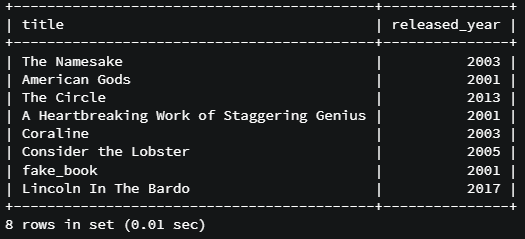
* + But the better way is to use NOT IN



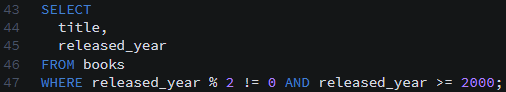


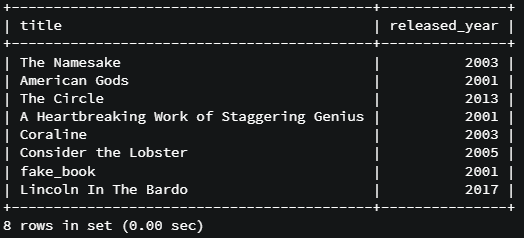
* + Suppose we want to *select only books released in odd-numbered years after the year 2000*. We can easily do this by using an AND operator





* + But there’s still a better way to do this! There’s a pattern in the years we want, and we don’t need to use IN or NOT IN. We won’t even need to explicitly type out the years that we want to exclude.
    - We can use the **modulo** or **remainder** operator, which provides the remainder (if there is one) after dividing one value by another
    - Documentation: <https://www.tutorialspoint.com/mysql/mysql_modulus_operator.htm>
    - Thus, we can use the modulo operator to ensure that the released\_year is odd





* Code summary

show databases();

use book\_shop;

SELECT

title,

author\_lname

FROM books

WHERE author\_lname='Carver' OR

author\_lname='Lahiri' OR

author\_lname='Smith';

SELECT title, author\_lname FROM books

WHERE author\_lname IN ('Carver', 'Lahiri', 'Smith');

SELECT title, released\_year FROM books

WHERE released\_year IN (2017, 1985);

SELECT title, released\_year FROM books

WHERE released\_year != 2000 AND

released\_year != 2002 AND

released\_year != 2004 AND

released\_year != 2006 AND

released\_year != 2008 AND

released\_year != 2010 AND

released\_year != 2012 AND

released\_year != 2014 AND

released\_year != 2016;

SELECT title, released\_year FROM books

WHERE released\_year NOT IN

(2000,2002,2004,2006,2008,2010,2012,2014,2016);

SELECT title, released\_year FROM books

WHERE released\_year >= 2000

AND released\_year NOT IN

(2000,2002,2004,2006,2008,2010,2012,2014,2016);

SELECT title, released\_year FROM books

WHERE released\_year >= 2000 AND

released\_year % 2 != 0;

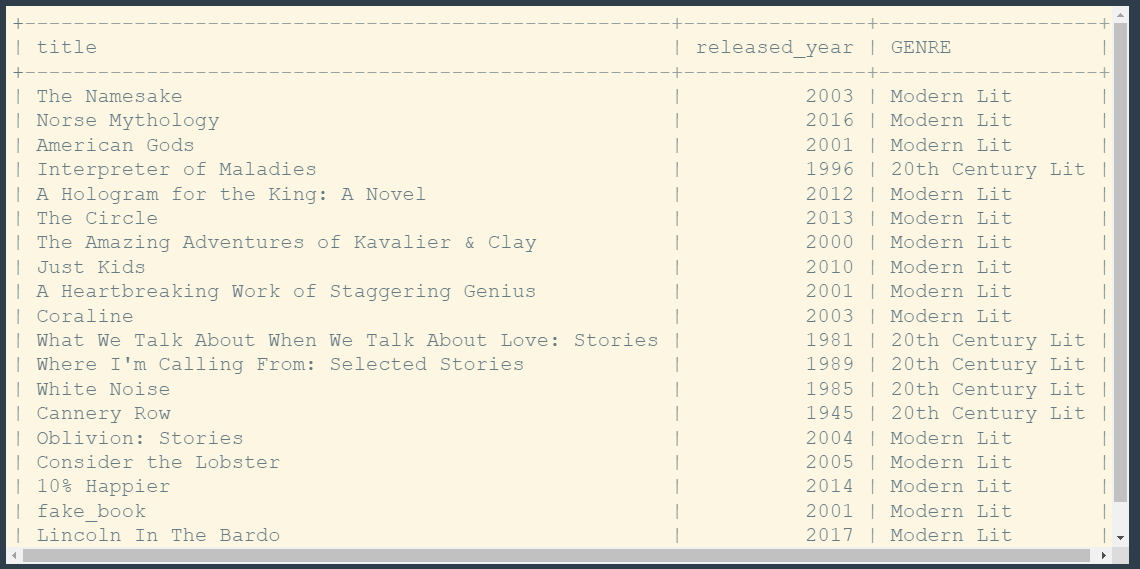
SELECT title, released\_year FROM books

WHERE released\_year >= 2000 AND

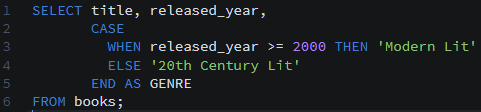
released\_year % 2 != 0 ORDER BY released\_year;

# Case Statements

* Case statements allow you to add logic to conditionally perform some action
  + Not unlike an “if” statement in other programming languageses
* For example, in this table below, the GENRE column is populated with “Modern Lit” if the released\_year is the year 2000 or later. Otherwise, it is labeled with “20th Century Lit”
  + That is, the value of GENRE depends on the value of “released\_year”

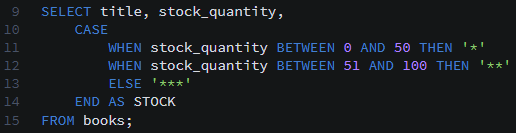


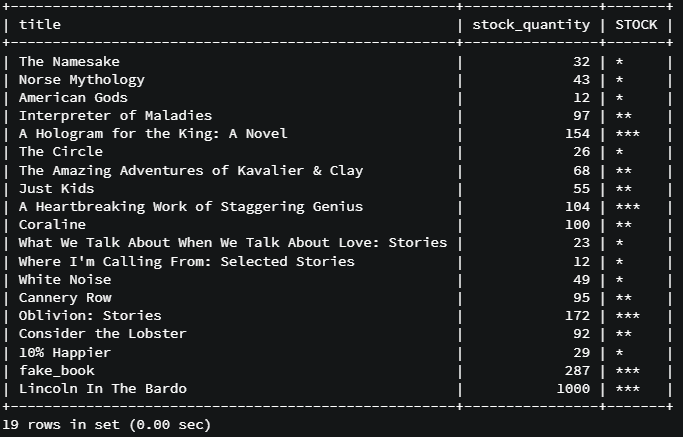
* Here is what the SQL syntax looks like, as well as the result:
  + Case statements always have a WHEN, THEN, and ELSE
  + The statement below says “when released year is greater than or equal to 2000, then set GENRE to ‘Modern Lit’. Otherwise, set GENRE to ‘20th Century Lit’”.
  + The “AS” statement allows you to label the resulting column with something nice, instead of that gross CASE statement



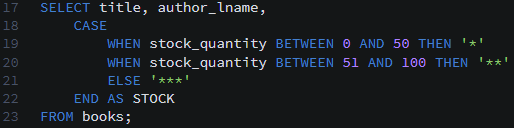


* Another example where we *print a graphical representation of how many books we have in stock, with the number of asterisks indicating qualitatively how many books we have*



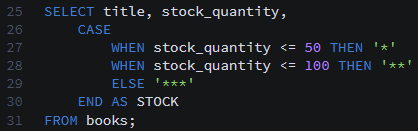


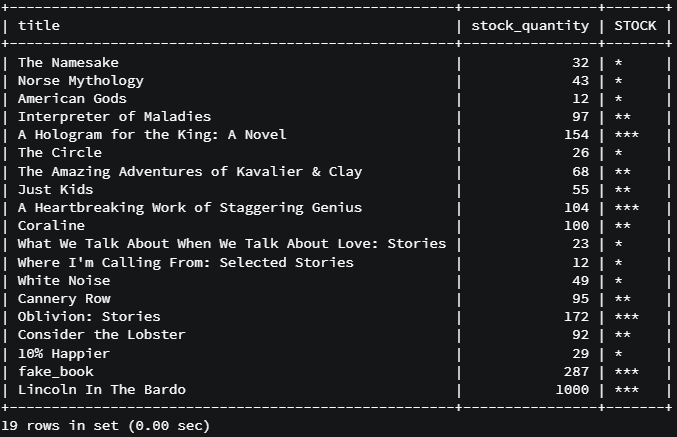
* You DO NOT need to select the values that you are referencing the case statement if you don’t want to





* You cannot put commas (“,”) between your WHEN statements
* CASE statements work according to a common execution order, similar to IF statements. The CASE statement assesses and tests conditions one line at a time. We can make our selection more succinct by relying on this execution order.
  + The example code below performs the exact same action





* Code summary

SELECT title, released\_year,

CASE

WHEN released\_year >= 2000 THEN 'Modern Lit'

ELSE '20th Century Lit'

END AS GENRE

FROM books;

SELECT title, stock\_quantity,

CASE

WHEN stock\_quantity BETWEEN 0 AND 50 THEN '\*'

WHEN stock\_quantity BETWEEN 51 AND 100 THEN '\*\*'

ELSE '\*\*\*'

END AS STOCK

FROM books;

SELECT title,

CASE

WHEN stock\_quantity BETWEEN 0 AND 50 THEN '\*'

WHEN stock\_quantity BETWEEN 51 AND 100 THEN '\*\*'

ELSE '\*\*\*'

END AS STOCK

FROM books;

SELECT title, stock\_quantity,

CASE

WHEN stock\_quantity BETWEEN 0 AND 50 THEN '\*'

WHEN stock\_quantity BETWEEN 51 AND 100 THEN '\*\*'

WHEN stock\_quantity BETWEEN 101 AND 150 THEN '\*\*\*'

ELSE '\*\*\*\*'

END AS STOCK

FROM books;

SELECT title, stock\_quantity,

CASE

WHEN stock\_quantity <= 50 THEN '\*'

WHEN stock\_quantity <= 100 THEN '\*\*'

ELSE '\*\*\*'

END AS STOCK

FROM books;