1. SHOW DATABASES; - self explanatory

2. CREATE DATABASE <NAME>; -- TO CREATE DATABASE UNDER <NAME>

3. DROP DATABASE <database_name> --- to delete database<database_name>

4. USE <database> -- to engage with the database

5. SELECT DATABASE(); -- to check which database is currently in use.
6. CREATE TABLE <table_name>
        (
                column_name data_type,
                column_name data_type
        );

e.g: 1. CREATE TABLE cats (name VARCHAR(100), age INT);
        2. CREATE TABLE pastries (name VARCHAR(50), quantity INT);

7. SHOW TABLES; --- displays table from selected database.

8. SHOW COLUMNS FROM cats; --- displays headers from cats table.

9. DESC cats; --- displays headers from cats table.

10. DROP TABLE <tablename>. ---- delete a table

11. INSERT INTO cats (name, age) VALUES(Jetson, 7);

12. INSERT INTO table_name(column_name) VALUES(data);
        e.g: INSERT INTO cats(name, age) VALUES('Jetson', 7);

13. SELECT * FROM cats; --- to view entries made into cats table.

14.INSERT INTO table_name (column_name, column_name) VALUES(value, value), (value, value), (value, value);

15. "This text has 'quotes' in it" or 'This text has "quotes" in it' --- how to insert a string (VARCHAR) value that contains quotations

16. SHOW WARNINGS ---- to display if there is any warning in the last query output

17. CREATE TABLE cats2 (name VARCHAR(100) NOT NULL, age INT NOT NULL);

18. CREATE TABLE cats3 (name VARCHAR(100) DEFAULT 'unnamed', age INT DEFAULT 99); -- to specify not null column

19. CREATE TABLE cats4 (name VARCHAR(50) DEFAULT 'UNNAMED' NOT NULL, age INT DEFAULT 99 NOT NULL);

19. CREATE TABLE meow (name VARCHAR(100) DEFAULT 'unnamed' NOT NULL, age INT

DEFAULT 99); --- example for default as well as not null

20. Primary key is a unique identifier in a row.

21. CREATE TABLE unique_cats(cat_id INT NOT NULL, name VARCHAR(50), age INT,
PRIMARY KEY (cat_id));

22. CREATE TABLE company(user_id INT NOT NULL PRIMARY KEY, age INT NOT NULL,
address VARCHAR(100) DEFAULT 'unspecified');

23. CREATE TABLE unique_cats2(cat_id INT NOT NULL PRIMARY KEY AUTO_INCREMENT, name
VARCHAR(50), age INT);

24. CREATE TABLE cats (cat_id INT NOT NULL AUTO_INCREMENT, name VARCHAR(100), breed
 VARCHAR(100), age INT, PRIMARY KEY (cat_id));

25. INSERT INTO cats(name, breed, age)
VALUES ('Ringo', 'Tabby', 4),
       ('Cindy', 'Maine Coon', 10),
       ('Dumbledore', 'Maine Coon', 11),
       ('Egg', 'Persian', 4),
       ('Misty', 'Tabby', 13),
       ('George Michael', 'Ragdoll', 9),
       ('Jackson', 'Sphynx', 7);

26. SELECT name FROM cats; --- to display name column from the cats table.

27. SELECT name,age FROM cats; ---- to display name,age from the cats table.

28. SELECT * FROM cats WHERE age=4;

29. SELECT * FROM cats WHERE name='Egg';

30. SELECT cat_id, age FROM cats WHERE cat_id=age;

31. SELECT cat_id AS id, name FROM cats;

32. SELECT name AS 'cat name', breed AS 'kitty breed' FROM cats;

33. UPDATE cats SET breed='Shorthair' WHERE breed='Tabby'; --- to update the table

34. UPDATE cats SET age=14 WHERE name='Misty';

35. DELETE FROM cats WHERE name='Egg';

36. DELETE FROM cats; ---- removes all entries from the cats table.

37. SELECT article, color, shirt_size, last_worn FROM shirts WHERE shirt_size='M';

38. UPDATE shirts SET color='off white', shirt_size='XS' WHERE color='white';

```
39. SELECT CONCAT (first_name, ' ', last_name) FROM books;

40. SELECT CONCAT (first_name, ' ', last_name) AS full_name FROM books;

41. SELECT CONCAT(author_fname, ' ', author_lname) AS 'full name' FROM books;

42. SELECT author_fname AS first, author_lname AS last, CONCAT(author_fname, ' ',
author_lname) AS full FROM books;

43. SELECT author_fname AS first, author_lname AS last, CONCAT(author_fname, ', ',
author_lname) AS full FROM books;

44. SELECT SUBSTRING('Hello World', 1, 4); ---- Hell

45. SELECT SUBSTRING('Hello World', 7);

46. SELECT SUBSTRING('Hello World', 3, 8);

47. SELECT SUBSTRING('Hello World', 3);

48. SELECT SUBSTRING('Hello World', -3);

49. SELECT SUBSTRING('Hello World', -7)

50. SELECT SUBSTRING("Where I'm Calling From: Selected Stories", 1, 10);

51. SELECT CONCAT(SUBSTRING(title, 1, 10), '...') FROM books;

52. SELECT SUBSTRING(REPLACE(title, 'e', '3'), 1, 10) FROM books;

53. SELECT SUBSTRING(REPLACE(title, 'e', '3'), 1, 10) AS 'weird string' FROM books;

54. SELECT REVERSE('Hello World');

55. SELECT REVERSE('meow meow');

56. SELECT REVERSE(author_fname) FROM books;

57. SELECT CONCAT('woof', REVERSE('woof'));

58. SELECT CONCAT(author_fname, REVERSE(author_fname)) FROM books;

59. SELECT REVERSE('Hello World');

60. SELECT REVERSE('meow meow');

61. SELECT REVERSE(author_fname) FROM books;

62. SELECT CONCAT('woof', REVERSE('woof'));
```

```
63. SELECT CONCAT(author_fname, REVERSE(author_fname)) FROM books;

64. SELECT UPPER('Hello World');

65. SELECT LOWER('Hello World');

66. SELECT UPPER(title) FROM books;

67. SELECT CONCAT('MY FAVORITE BOOK IS ', UPPER(title)) FROM books;

68. SELECT CONCAT('MY FAVORITE BOOK IS ', LOWER(title)) FROM books;

69. SELECT UPPER(CONCAT(author_fname, ' ', author_lname)) AS "full name in caps"
FROM books;


70. SELECT REVERSE(UPPER('Why does my cat look at me with such hatred?'));


71. SELECT UPPER(REVERSE('Why does my cat look at me with such hatred?'));

72. SELECT REPLACE(CONCAT('I', ' ', 'like', ' ', 'cats'), ' ', '-'); ------
I-like-cats


73.SELECT REPLACE(title, ' ', '->') AS title FROM books;



74. SELECT author_lname AS forwards, REVERSE(author_lname) AS backwards FROM books;


75. SELECT UPPER (CONCAT(author_fname, ' ', author_lname)) AS 'full name in caps'
FROM books;


76. SELECT CONCAT(title, ' was released in ', released_year) AS blurb FROM books;


77. SELECT title, CHAR_LENGTH(title) AS 'character count' FROM books;


78. SELECT CONCAT(SUBSTRING(title, 1, 10), '...') AS 'short title',
   CONCAT(author_lname, ',', author_fname) AS author,
   CONCAT(stock_quantity, ' in stock') AS quantity
   FROM books;

79. SELECT author_lname FROM books;
```

80. SELECT DISTINCT author_lname FROM books;

81. SELECT author_fname, author_lname FROM books;

82. SELECT DISTINCT CONCAT(author_fname,' ', author_lname) FROM books;

83. SELECT DISTINCT author_fname, author_lname FROM books;

84. SELECT author_lname FROM books ORDER BY author_lname;

85. SELECT title FROM books ORDER BY title;

86. SELECT author_lname FROM books ORDER BY author_lname DESC;

87. SELECT released_year FROM books ORDER BY released_year;

88. SELECT released_year FROM books ORDER BY released_year DESC;

89. SELECT released_year FROM books ORDER BY released_year ASC;

90. SELECT title, released_year, pages FROM books ORDER BY released_year;

91. SELECT title, pages FROM books ORDER BY released_year;

92. SELECT title, author_fname, author_lname FROM books ORDER BY 2;

93. SELECT title, author_fname, author_lname FROM books ORDER BY 3;

94. SELECT title, author_fname, author_lname
FROM books ORDER BY 1 DESC;

96. SELECT author_lname, title
FROM books ORDER BY 2;

97. SELECT author_fname, author_lname FROM books
ORDER BY author_lname, author_fname

98. SELECT DISTINCT author_lname FROM books ORDER BY author_lname.

100. SELECT title FROM books LIMIT 3;

101. SELECT title FROM books LIMIT 1;

102. SELECT title FROM books LIMIT 10;

103. SELECT * FROM books LIMIT 1;

104. SELECT title, released_year FROM books
ORDER BY released_year DESC LIMIT 5;

```
105. SELECT title, released_year FROM books
ORDER BY released_year DESC LIMIT 1;

106. SELECT title, released_year FROM books
ORDER BY released_year DESC LIMIT 14;

107. SELECT title, released_year FROM books
ORDER BY released_year DESC LIMIT 0,5;

108. SELECT title, released_year FROM books
ORDER BY released_year DESC LIMIT 0,3;

109. SELECT title, released_year FROM books
ORDER BY released_year DESC LIMIT 1,3;

110 . SELECT title, released_year FROM books
ORDER BY released_year DESC LIMIT 10,1;

111. SELECT * FROM tbl LIMIT 95,18446744073709551615;

112. SELECT title FROM books LIMIT 5;

113. SELECT title FROM books LIMIT 5, 123219476457;

114. SELECT title FROM books LIMIT 5, 50;

115. SELECT title, author_fname FROM books WHERE author_fname LIKE '%da%';

116. SELECT title, author_fname FROM books WHERE author_fname LIKE 'da%';

117. SELECT title FROM books WHERE  title LIKE 'the';

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

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

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 '%\_%'
```

```
SELECT title FROM books WHERE title LIKE '%stories%';

SELECT title, pages FROM books ORDER BY pages DESC LIMIT 1;

SELECT
    CONCAT(title, ' - ', released_year) AS summary
FROM books ORDER BY released_year DESC LIMIT 3;

SELECT title, author_lname FROM books WHERE author_lname LIKE '% %';

SELECT title, released_year, stock_quantity
FROM books ORDER BY stock_quantity LIMIT 3;

SELECT title, author_lname
FROM books ORDER BY author_lname, title;

SELECT title, author_lname
FROM books ORDER BY 2,1;

SELECT
    CONCAT(
        'MY FAVORITE AUTHOR IS ',
        UPPER(author_fname),
        ' ',
        UPPER(author_lname),
        '!'
    ) AS yell
FROM books ORDER BY author_lname;

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%';

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;

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;

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;


SELECT author_fname,
       author_lname,
       Min(released_year)
FROM   books
GROUP  BY author_lname,
          author_fname;

SELECT
  author_fname,
  author_lname,
  Max(pages)
FROM books
GROUP BY author_lname,
         author_fname;

SELECT
  CONCAT(author_fname, ' ', author_lname) AS author,
  MAX(pages) AS 'longest book'
FROM books
GROUP BY author_lname,

SELECT SUM(pages)
FROM books;

SELECT SUM(released_year) FROM books;

SELECT author_fname,
       author_lname,
       Sum(pages)
FROM books
GROUP BY
    author_lname,
    author_fname;

SELECT author_fname,
       author_lname,
       Sum(released_year)
FROM books
GROUP BY
    author_lname,
    author_fname;


SELECT AVG(released_year)
```

```
FROM books;

SELECT AVG(pages)
FROM books;

SELECT AVG(stock_quantity)
FROM books
GROUP BY released_year;

SELECT released_year, AVG(stock_quantity)
FROM books
GROUP BY released_year;

SELECT author_fname, author_lname, AVG(pages) FROM books
GROUP BY author_lname, author_fname;


SELECT COUNT(*) FROM books;

SELECT COUNT(*) FROM books GROUP BY released_year;

SELECT released_year, COUNT(*) FROM books GROUP BY released_year;

SELECT Sum(stock_quantity) FROM BOOKS;

SELECT AVG(released_year) FROM books GROUP BY author_lname, author_fname;

SELECT author_fname, author_lname, AVG(released_year) FROM books GROUP BY
author_lname, author_fname;

SELECT CONCAT(author_fname, ' ', author_lname) FROM books
WHERE pages = (SELECT Max(pages) FROM books);

SELECT CONCAT(author_fname, ' ', author_lname) FROM books
ORDER BY pages DESC LIMIT 1;

SELECT pages, CONCAT(author_fname, ' ', author_lname) FROM books
ORDER BY pages DESC;

SELECT released_year AS year,
    COUNT(*) AS '# of books',
    AVG(pages) AS 'avg pages'
FROM books
    GROUP BY released_year;

CREATE TABLE dogs (name CHAR(5), breed VARCHAR(10));

INSERT INTO dogs (name, breed) VALUES ('bob', 'beagle');

INSERT INTO dogs (name, breed) VALUES ('robby', 'corgi');
```

```
INSERT INTO dogs (name, breed) VALUES ('Princess Jane', 'Retriever');

SELECT * FROM dogs;

INSERT INTO dogs (name, breed) VALUES ('Princess Jane',
'Retrievesadfdsafdasfsafr');

SELECT * FROM dogs;

CREATE TABLE items(price DECIMAL(5,2));

INSERT INTO items(price) VALUES(7);

INSERT INTO items(price) VALUES(7987654);

INSERT INTO items(price) VALUES(34.88);

INSERT INTO items(price) VALUES(298.9999);

INSERT INTO items(price) VALUES(1.9999);

SELECT * FROM items

CREATE TABLE people (name VARCHAR(100), birthdate DATE, birthtime TIME, birthdt
DATETIME);

INSERT INTO people (name, birthdate, birthtime, birthdt)
VALUES('Padma', '1983-11-11', '10:07:35', '1983-11-11 10:07:35');

INSERT INTO people (name, birthdate, birthtime, birthdt)
VALUES('Larry', '1943-12-25', '04:10:42', '1943-12-25 04:10:42');

SELECT * FROM people;

SELECT name, birthdate FROM people;

SELECT name, DAY(birthdate) FROM people;

SELECT name, birthdate, DAY(birthdate) FROM people;

SELECT name, birthdate, DAYNAME(birthdate) FROM people;

SELECT name, birthdate, DAYOFWEEK(birthdate) FROM people;

SELECT name, birthdate, DAYOFYEAR(birthdate) FROM people;

SELECT name, birthtime, DAYOFYEAR(birthtime) FROM people;

SELECT name, birthdt, DAYOFYEAR(birthdt) FROM people;
```

```
SELECT name, birthdt, MONTH(birthdt) FROM people;

SELECT name, birthdt, MONTHNAME(birthdt) FROM people;

SELECT name, birthtime, HOUR(birthtime) FROM people;

SELECT name, birthtime, MINUTE(birthtime) FROM people;

SELECT CONCAT(MONTHNAME(birthdate), ' ', DAY(birthdate), ' ', YEAR(birthdate)) FROM
people;

SELECT DATE_FORMAT(birthdt, 'Was born on a %W') FROM people;

SELECT DATE_FORMAT(birthdt, '%m/%d/%Y') FROM people;

SELECT DATE_FORMAT(birthdt, '%m/%d/%Y at %h:%i') FROM people;



SELECT * FROM people;

SELECT DATEDIFF(NOW(), birthdate) FROM people;

SELECT name, birthdate, DATEDIFF(NOW(), birthdate) FROM people;

SELECT birthdt FROM people;

SELECT birthdt, DATE_ADD(birthdt, INTERVAL 1 MONTH) FROM people;

SELECT birthdt, DATE_ADD(birthdt, INTERVAL 10 SECOND) FROM people;

SELECT birthdt, DATE_ADD(birthdt, INTERVAL 3 QUARTER) FROM people;

SELECT birthdt, birthdt + INTERVAL 1 MONTH FROM people;

SELECT birthdt, birthdt - INTERVAL 5 MONTH FROM people;

SELECT birthdt, birthdt + INTERVAL 15 MONTH + INTERVAL 10 HOUR FROM people;

CREATE TABLE comments (
    content VARCHAR(100),
    created_at TIMESTAMP DEFAULT NOW()
);

INSERT INTO comments (content) VALUES('lol what a funny article');

INSERT INTO comments (content) VALUES('I found this offensive');
```

```
INSERT INTO comments (content) VALUES('Ifasfsadfsadfsad');

SELECT * FROM comments ORDER BY created_at DESC;

CREATE TABLE comments2 (
    content VARCHAR(100),
    changed_at TIMESTAMP DEFAULT NOW() ON UPDATE CURRENT_TIMESTAMP
);

INSERT INTO comments2 (content) VALUES('dasdasdasd');

INSERT INTO comments2 (content) VALUES('lolololololo');

INSERT INTO comments2 (content) VALUES('I LIKE CATS AND DOGS');

UPDATE comments2 SET content='THIS IS NOT GIBBERISH' WHERE content='dasdasdasd';

SELECT * FROM comments2;

SELECT * FROM comments2 ORDER BY changed_at;

CREATE TABLE comments2 (
    content VARCHAR(100),
    changed_at TIMESTAMP DEFAULT NOW() ON UPDATE NOW()
);

What's a good use case for CHAR?
------
Used for text that we know has a fixed length, e.g., State abbreviations,
abbreviated company names, sex M/F, etc.

CREATE TABLE inventory (
    item_name VARCHAR(100),
    price DECIMAL(8,2),
    quantity INT
);

What's the difference between DATETIME and TIMESTAMP?
------
They both store datetime information, but there's a difference in the range,
TIMESTAMP has a smaller range. TIMESTAMP also takes up less space.
TIMESTAMP is used for things like meta-data about when something is created
or updated.

SELECT CURTIME();

SELECT CURDATE()';

SELECT DAYOFWEEK(CURDATE());
SELECT DAYOFWEEK(NOW());
```

```
SELECT DATE_FORMAT(NOW(), '%w') + 1;

SELECT DAYNAME(NOW());
SELECT DATE_FORMAT(NOW(), '%W');

SELECT DATE_FORMAT(CURDATE(), '%m/%d/%Y');

SELECT DATE_FORMAT(NOW(), '%M %D at %h:%i');

CREATE TABLE tweets(
    content VARCHAR(140),
    username VARCHAR(20),
    created_at TIMESTAMP DEFAULT NOW()
);

INSERT INTO tweets (content, username) VALUES('this is my first tweet',
'coltscat');
SELECT * FROM tweets;

INSERT INTO tweets (content, username) VALUES('this is my second tweet',
'coltscat');
SELECT * FROM tweets;
Finding Orders Placed By George: Using a subquery

SELECT * FROM orders WHERE customer_id =
    (
        SELECT id FROM customers
        WHERE last_name='George'
    );

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';

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%';
```

```
CODE: Greater Than
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';

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


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%';
```

```
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;


SELECT 10 != 10;
-- false

SELECT 15 > 14 && 99 - 5 <= 94;
-- true

SELECT 1 IN (5,3) || 9 BETWEEN 8 AND 10;
-- true

SELECT title, released_year FROM books WHERE released_year < 1980;

SELECT title, author_lname FROM books WHERE author_lname='Eggers' OR
author_lname='Chabon';

SELECT title, author_lname FROM books WHERE author_lname IN ('Eggers','Chabon');

SELECT title, author_lname, released_year FROM books WHERE author_lname = 'Lahiri'
&& released_year > 2000;

SELECT title, pages FROM books WHERE pages >= 100 && pages <=200;
```

```
SELECT title, pages FROM books WHERE pages BETWEEN 100 AND 200;

SELECT
    title,
    author_lname
FROM books
WHERE
    author_lname LIKE 'C%' OR
    author_lname LIKE 'S%';

SELECT
    title,
    author_lname
FROM books
WHERE
    SUBSTR(author_lname,1,1) = 'C' OR
    SUBSTR(author_lname,1,1) = 'S';

SELECT title, author_lname FROM books
WHERE SUBSTR(author_lname,1,1) IN ('C', 'S');

SELECT
    title,
    author_lname,
    CASE
        WHEN title LIKE '%stories%' THEN 'Short Stories'
        WHEN title = 'Just Kids' OR title = 'A Heartbreaking Work of Staggering
Genius' THEN 'Memoir'
        ELSE 'Novel'
    END AS TYPE
FROM books;

SELECT author_fname, author_lname,
    CASE
        WHEN COUNT(*) = 1 THEN '1 book'
        ELSE CONCAT(COUNT(*), ' books')
    END AS COUNT
FROM books
GROUP BY author_lname, author_fname;

-- IMPLICIT INNER JOIN


-- IMPLICIT INNER JOIN

SELECT first_name, last_name, order_date, amount
FROM customers, orders
    WHERE customers.id = orders.customer_id;

-- EXPLICIT INNER JOINS
```

```
SELECT * FROM customers
JOIN orders
    ON customers.id = orders.customer_id;

SELECT first_name, last_name, order_date, amount
FROM customers
JOIN orders
    ON customers.id = orders.customer_id;

SELECT *
FROM orders
JOIN customers
    ON customers.id = orders.customer_id;
-- ARBITRARY JOIN - meaningless, but still possible

SELECT * FROM customers
JOIN orders ON customers.id = orders.id;


CODE: Left Joins
-- Getting Fancier (Inner Joins Still)

SELECT first_name, last_name, order_date, amount
FROM customers
JOIN orders
    ON customers.id = orders.customer_id
ORDER BY order_date;
SELECT
    first_name,
    last_name,
    SUM(amount) AS total_spent
FROM customers
JOIN orders
    ON customers.id = orders.customer_id
GROUP BY orders.customer_id
ORDER BY total_spent DESC;
Note: please see here for an animated visual of how left/right joins work.

-- LEFT JOINS

SELECT * FROM customers
LEFT JOIN orders
    ON customers.id = orders.customer_id;
SELECT first_name, last_name, order_date, amount
FROM customers
LEFT JOIN orders
    ON customers.id = orders.customer_id;

SELECT
```

```
    first_name,
    last_name,
    IFNULL(SUM(amount), 0) AS total_spent
FROM customers
LEFT JOIN orders
    ON customers.id = orders.customer_id
GROUP BY customers.id
ORDER BY total_spent;


CODE: Right Joins Part 1
Note: please see here for an animated visual of how left/right joins work.

-- OUR FIRST RIGHT JOIN (seems the same as a left join?)

SELECT * FROM customers
RIGHT JOIN orders
    ON customers.id = orders.customer_id;

-- ALTERING OUR SCHEMA to allow for a better example (optional)

CREATE TABLE customers(
    id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    email VARCHAR(100)
);
CREATE TABLE orders(
    id INT AUTO_INCREMENT PRIMARY KEY,
    order_date DATE,
    amount DECIMAL(8,2),
    customer_id INT
);
-- INSERTING NEW DATA (no longer bound by foreign key constraint)

INSERT INTO customers (first_name, last_name, email)
VALUES ('Boy', 'George', 'george@gmail.com'),
       ('George', 'Michael', 'gm@gmail.com'),
       ('David', 'Bowie', 'david@gmail.com'),
       ('Blue', 'Steele', 'blue@gmail.com'),
       ('Bette', 'Davis', 'bette@aol.com');

INSERT INTO orders (order_date, amount, customer_id)
VALUES ('2016/02/10', 99.99, 1),
       ('2017/11/11', 35.50, 1),
       ('2014/12/12', 800.67, 2),
       ('2015/01/03', 12.50, 2),
       ('1999/04/11', 450.25, 5);

INSERT INTO orders (order_date, amount, customer_id) VALUES
```

```
('2017/11/05', 23.45, 45),
(CURDATE(), 777.77, 109);

CODE: Right Joins Part 2
Note: please see here for an animated visual of how left/right joins work.

--A MORE COMPLEX RIGHT JOIN

SELECT
    IFNULL(first_name,'MISSING') AS first,
    IFNULL(last_name,'USER') as last,
    order_date,
    amount,
    SUM(amount)
FROM customers
RIGHT JOIN orders
    ON customers.id = orders.customer_id
GROUP BY first_name, last_name;
-- WORKING WITH ON DELETE CASCADE

CREATE TABLE customers(
    id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    email VARCHAR(100)
);

CREATE TABLE orders(
    id INT AUTO_INCREMENT PRIMARY KEY,
    order_date DATE,
    amount DECIMAL(8,2),
    customer_id INT,
    FOREIGN KEY(customer_id)
        REFERENCES customers(id)
        ON DELETE CASCADE
);


INSERT INTO customers (first_name, last_name, email)
VALUES ('Boy', 'George', 'george@gmail.com'),
       ('George', 'Michael', 'gm@gmail.com'),
       ('David', 'Bowie', 'david@gmail.com'),
       ('Blue', 'Steele', 'blue@gmail.com'),
       ('Bette', 'Davis', 'bette@aol.com');

INSERT INTO orders (order_date, amount, customer_id)
VALUES ('2016/02/10', 99.99, 1),
       ('2017/11/11', 35.50, 1),
       ('2014/12/12', 800.67, 2),
       ('2015/01/03', 12.50, 2),
```

```
       ('1999/04/11', 450.25, 5);

SELECT * FROM customers
LEFT JOIN orders
    ON customers.id = orders.customer_id;
SELECT * FROM orders
RIGHT JOIN customers
    ON customers.id = orders.customer_id;
SELECT * FROM orders
LEFT JOIN customers
    ON customers.id = orders.customer_id;
SELECT * FROM customers
RIGHT JOIN orders
    ON customers.id = orders.customer_id;


CODE: Our First Joins Exercise
-- The Schema

CREATE TABLE students (
    id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100)
);


CREATE TABLE papers (
    title VARCHAR(100),
    grade INT,
    student_id INT,
    FOREIGN KEY (student_id)
        REFERENCES students(id)
        ON DELETE CASCADE
);
-- The Starter Data

INSERT INTO students (first_name) VALUES
('Caleb'),
('Samantha'),
('Raj'),
('Carlos'),
('Lisa');

INSERT INTO papers (student_id, title, grade ) VALUES
(1, 'My First Book Report', 60),
(1, 'My Second Book Report', 75),
(2, 'Russian Lit Through The Ages', 94),
(2, 'De Montaigne and The Art of The Essay', 98),
(4, 'Borges and Magical Realism', 89);
```

CODE: Our First Joins Exercise SOLUTION PT. 2

```
-- EXERCISE 1

SELECT first_name, title, grade
FROM students
INNER JOIN papers
    ON students.id = papers.student_id
ORDER BY grade DESC;
-- ALT SOLUTION

SELECT first_name, title, grade
FROM students
RIGHT JOIN papers
    ON students.id = papers.student_id
ORDER BY grade DESC;
-- PROBLEM 2

SELECT first_name, title, grade
FROM students
LEFT JOIN papers
    ON students.id = papers.student_id;
-- PROBLEM 3

SELECT
    first_name,
    IFNULL(title, 'MISSING'),
    IFNULL(grade, 0)
FROM students
LEFT JOIN papers
    ON students.id = papers.student_id;

-- PROBLEM 4

SELECT
    first_name,
    IFNULL(AVG(grade), 0) AS average
FROM students
LEFT JOIN papers
    ON students.id = papers.student_id
GROUP BY students.id
ORDER BY average DESC;
-- PROBLEM 5

SELECT first_name,
       Ifnull(Avg(grade), 0) AS average,
       CASE
         WHEN Avg(grade) IS NULL THEN 'FAILING'
         WHEN Avg(grade) >= 75 THEN 'PASSING'
         ELSE 'FAILING'
```

```
       end                   AS passing_status
FROM   students
       LEFT JOIN papers
              ON students.id = papers.student_id
GROUP  BY students.id
ORDER  BY average DESC;
```

```
INSERT INTO reviewers (fname, lname) VALUES
    ('Thomas', 'Stoneman'),
    ('Wyatt', 'Skaggs'),
    ('Kimbra', 'Masters'),
    ('Domingo', 'Cortes'),
    ('Colt', 'Steele'),
    ('Pinkie', 'Petit'),
    ('Marlon', 'Crafford');
```

CODE: Creating Our Tables

```
-- 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);



CODE: TV Joins Challenge 1 Solution
-- TV Joins Challenge 1 SOLUTION



SELECT
```

```
    title,
    rating
FROM series
JOIN reviews
    ON series.id = reviews.series_id;

 Challenge 2 AVG rating

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;


-- CHALLENGE 3  - Two Solutions

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;

- CHALLENGE 4 - UNREVIEWED SERIES

SELECT title AS unreviewed_series
FROM series
LEFT JOIN reviews
    ON series.id = reviews.series_id
WHERE rating IS NULL;

-- Challenge 5 - GENRE AVG RATINGS

SELECT genre,
       Round(Avg(rating), 2) AS avg_rating
FROM   series
```

```
       INNER JOIN reviews
               ON series.id = reviews.series_id
GROUP  BY genre;
```

```
select
fname,
lname,
count(rating) as count,
IFNULL(MIN(rating),0) as MIN,
IFNULL(max(rating),0) as MAX,
IFNULL(avg(rating),0) as AVERAGE,
case when count(rating) = 0 then'ACTIVE' else 'INACTIVE' end as 'STATUS',
from reviewers left join reviews on reviewers.id = reviews.reviewer_id
group by fname;
```

```
select
fname,
lname,
count(rating) as count,
IFNULL(MIN(rating),0) as MIN,
IFNULL(max(rating),0) as MAX,
IFNULL(avg(rating),0) as AVERAGE,
IF(count(rating) = 0, 'INACTIVE','ACTIVE') as STATUS
from reviewers left join reviews on reviewers.id = reviews.reviewer_id
group by fname;
```

```
select
fname,
lname,
count(rating) as count,
IFNULL(MIN(rating),0) as MIN,
IFNULL(max(rating),0) as MAX,
IFNULL(avg(rating),0) as AVERAGE,
case
when count(rating) >= 10 then 'POWER USER'
when count(rating) = 0 then 'INACIVE'
else 'ACTIVE'
end as 'STATUS',
from reviewers left join reviews on reviewers.id = reviews.reviewer_id
group by fname;
```

```
select fname,lname,count(rating) as count,IFNULL(MIN(rating),0) as
MIN,IFNULL(max(rating),0) as MAX,IFNULL(avg(rating),0) as AVERAGE,
```

```
case when count(rating) = 0 then 'INACTIVE' when count(rating) >=10 then
'POWER_USER' else 'ACTIVE' end as 'STATUS'
from reviewers left join reviews on reviewers.id = reviews.reviewer_id group by
fname;



-- 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;



-- CHALLENGE 7 - 3 TABLES!

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;



CREATE TABLE users (
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) UNIQUE NOT NULL,
    created_at TIMESTAMP DEFAULT NOW()
);


CREATE TABLE photos (
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    image_url VARCHAR(255) NOT NULL,
    user_id INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    FOREIGN KEY(user_id) REFERENCES users(id)
);



CREATE TABLE comments (
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    comment_text VARCHAR(255) NOT NULL,
    photo_id INTEGER NOT NULL,
    user_id INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    FOREIGN KEY(photo_id) REFERENCES photos(id),
    FOREIGN KEY(user_id) REFERENCES users(id)
);

CREATE TABLE likes (
    user_id INTEGER NOT NULL,
    photo_id INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    FOREIGN KEY(user_id) REFERENCES users(id),
    FOREIGN KEY(photo_id) REFERENCES photos(id),
    PRIMARY KEY(user_id, photo_id)
);


CREATE TABLE follows (
    follower_id INTEGER NOT NULL,
    followee_id INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    FOREIGN KEY(follower_id) REFERENCES users(id),
    FOREIGN KEY(followee_id) REFERENCES users(id),
    PRIMARY KEY(follower_id, followee_id)
);
```

CODE: IG Clone Hashtags Schema

```
CREATE TABLE tags (
  id INTEGER AUTO_INCREMENT PRIMARY KEY,
  tag_name VARCHAR(255) UNIQUE,
  created_at TIMESTAMP DEFAULT NOW()
);
CREATE TABLE photo_tags (
    photo_id INTEGER NOT NULL,
    tag_id INTEGER NOT NULL,
    FOREIGN KEY(photo_id) REFERENCES photos(id),
    FOREIGN KEY(tag_id) REFERENCES tags(id),
    PRIMARY KEY(photo_id, tag_id)
);
```

CODE: Complete IG Clone Schema

```
CREATE TABLE users (
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) UNIQUE NOT NULL,
    created_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE photos (
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    image_url VARCHAR(255) NOT NULL,
    user_id INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    FOREIGN KEY(user_id) REFERENCES users(id)
);

CREATE TABLE comments (
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    comment_text VARCHAR(255) NOT NULL,
    photo_id INTEGER NOT NULL,
    user_id INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    FOREIGN KEY(photo_id) REFERENCES photos(id),
    FOREIGN KEY(user_id) REFERENCES users(id)
);

CREATE TABLE likes (
    user_id INTEGER NOT NULL,
    photo_id INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    FOREIGN KEY(user_id) REFERENCES users(id),
    FOREIGN KEY(photo_id) REFERENCES photos(id),
    PRIMARY KEY(user_id, photo_id)
);
```

```
CREATE TABLE follows (
    follower_id INTEGER NOT NULL,
    followee_id INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    FOREIGN KEY(follower_id) REFERENCES users(id),
    FOREIGN KEY(followee_id) REFERENCES users(id),
    PRIMARY KEY(follower_id, followee_id)
);

CREATE TABLE tags (
  id INTEGER AUTO_INCREMENT PRIMARY KEY,
  tag_name VARCHAR(255) UNIQUE,
  created_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE photo_tags (
    photo_id INTEGER NOT NULL,
    tag_id INTEGER NOT NULL,
    FOREIGN KEY(photo_id) REFERENCES photos(id),
    FOREIGN KEY(tag_id) REFERENCES tags(id),
    PRIMARY KEY(photo_id, tag_id)
);
```

Finding 5 oldest users

```
SELECT *
FROM users
ORDER BY created_at
LIMIT 5;
```

Most Popular Registration Date

```
SELECT
    DAYNAME(created_at) AS day,
    COUNT(*) AS total
FROM users
GROUP BY day
ORDER BY total DESC
LIMIT 2;
```

Most Popular Registration Date

```
SELECT
    DAYNAME(created_at) AS day,
    COUNT(*) AS total
FROM users
GROUP BY day
ORDER BY total DESC
```

```
LIMIT 2;


- 4. Identify most popular photo (and user who created it)

SELECT
    username,
    photos.id,
    photos.image_url,
    COUNT(*) AS total
FROM photos
INNER JOIN likes
    ON likes.photo_id = photos.id
INNER JOIN users
    ON photos.user_id = users.id
GROUP BY photos.id
ORDER BY total DESC
LIMIT 1;


Calculate average number of photos per user

SELECT (SELECT Count(*)
        FROM   photos) / (SELECT Count(*)
                          FROM   users) AS avg;


 6. Find the five most popular hashtags

SELECT tags.tag_name,
       Count(*) AS total
FROM   photo_tags
       JOIN tags
         ON photo_tags.tag_id = tags.id
GROUP  BY tags.id
ORDER  BY total DESC
LIMIT  5;


SELECT username,
       Count(*) AS num_likes
FROM   users
       INNER JOIN likes
               ON users.id = likes.user_id
GROUP  BY likes.user_id
HAVING num_likes = (SELECT Count(*)
                    FROM   photos);
```