# start file

mysql-ctl cli

#ask for databases

create database dog;

use dog;

select database();

show databases;

# ask for tables

create table pastries

(

name varchar (50),

age int

);

show tables;

desc pastries;

drop table pastries;

# insert data

insert into pastries(name, age) values ('cute', 99);

INSERT INTO pastries

(name, age)

VALUES ('love', 88),

('beautiful', 20),

('adorable', 988);

SELECT \* FROM pastries;

SHOW WARNINGS;

# if errors instead of warning, then use set sql\_mode='';

CREATE TABLE pastries2

(

name VARCHAR(50) NOT NULL,

age INT NOT NULL

);

CREATE TABLE pastries3

(

name VARCHAR(50) DEFAULT 'no name provided',

age INT DEFAULT 99

);

# add a primary key

CREATE TABLE unique\_pastries4

(

person\_id INT NOT NULL auto\_increment,

name VARCHAR(50),

age INT,

PRIMARY KEY (person\_id)

);

INSERT INTO unique\_pastries4(person\_id, name, age) VALUES(1, 'Fred', 23);

INSERT INTO unique\_pastries4(person\_id, name, age) VALUES(2, 'Louise', 3);

INSERT INTO unique\_pastries4(person\_id, name, age) VALUES(1, 'James', 3);

SELECT \* FROM unique\_pastries4;

SELECT name FROM unique\_pastries4;

SELECT name, age FROM unique\_pastries4;

SELECT \* FROM unique\_pastries4 WHERE age=3;

SELECT person\_id, age FROM unique\_pastries4 WHERE person\_id<=age;

SELECT name AS 'person name', age AS 'birthday' FROM unique\_pastries4;

UPDATE unique\_pastries4 SET age=999 WHERE person\_id='2';

DELETE FROM cats WHERE name='Egg';

DELETE FROM cats; # delete all items in that table, not in that database

# use some files

source xxx.sql

# data type

drop database if exists dog;

create database dog;

use dog;

create table dogs (name char(5), breed varchar(10));

insert into dogs (name, breed)

value

('bob', 'beagle'),

('robby', 'corgi'),

('princess jane', 'retriever');

select \* from dogs;

# Used for text that we know has a fixed length, e.g., State abbreviations,

# abbreviated company names, sex M/F, etc.

# decimal

create table items(price decimal(5,2)); # (p,s), p is the precision = all digits, s = the digital number

insert into items (price)

value

(7), (10000), (23.8022222), (66.66);

# float and double

CREATE TABLE thingies (price FLOAT);

INSERT INTO thingies(price) VALUES (88.45);

SELECT \* FROM thingies;

INSERT INTO thingies(price) VALUES (8877.45);

SELECT \* FROM thingies;

INSERT INTO thingies(price) VALUES (8877665544.45);

SELECT \* FROM thingies;

# date, time and datetime

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, DAY(birthdate) FROM people;

SELECT name, birthdate, DAYNAME(birthdate) FROM people; # monday...

SELECT name, birthdate, DAYOFWEEK(birthdate) FROM people; # Sunday = 1

SELECT name, birthdate, DAYOFYEAR(birthdate) 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;

# format functin for time

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

SELECT DATE\_FORMAT(birthdt, 'Was born on a %W') FROM people; # Was born on a Friday

SELECT DATE\_FORMAT(birthdt, '%m/%d/%Y') FROM people; # 11/11/1983

SELECT DATE\_FORMAT(birthdt, '%m/%d/%Y at %h:%i') FROM people; # 11/11/1983 at 10:07

# data math

SELECT \* FROM people;

SELECT name, birthdate, DATEDIFF(NOW(), birthdate) FROM people; # date from now

SELECT birthdt, DATE\_ADD(birthdt, INTERVAL 1 MONTH) FROM people; # + one month

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;

# time stamp

# 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.

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');

SELECT \* FROM comments ORDER BY created\_at DESC;

CREATE TABLE comments2 (

content VARCHAR(100),

changed\_at TIMESTAMP DEFAULT NOW() ON UPDATE CURRENT\_TIMESTAMP # if changed,updated the current time

);

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

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

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

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

SELECT \* FROM comments2 ORDER BY changed\_at;

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');

# an example for between function

SELECT

name,

birthdt

FROM people

WHERE

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

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

# CAST (expression AS data\_type) It helps to transfer data type

# joint functions

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);

# question one:

SELECT first\_name, title, grade

FROM students

INNER JOIN papers

ON students.id = papers.student\_id

ORDER BY grade DESC;

# or

SELECT first\_name, title, grade

FROM students

RIGHT JOIN papers

ON students.id = papers.student\_id

ORDER BY grade DESC;

# question two:

SELECT first\_name, title, grade

FROM students

LEFT JOIN papers

ON students.id = papers.student\_id;

# question three:

SELECT

first\_name,

IFNULL(title, 'MISSING'),

IFNULL(grade, 0)

FROM students

LEFT JOIN papers

ON students.id = papers.student\_id;

# question four:

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;

# question five:

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;

# logic operations

# unequal & like function

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

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%';

# greater than & less than function

SELECT title, released\_year FROM books

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

SELECT title, released\_year FROM books

WHERE released\_year <= 2000;

SELECT -10 < -9; # ture

# logic: and, or, between

SELECT

title,

author\_lname,

released\_year FROM books

WHERE author\_lname='Eggers'

AND released\_year > 2010; # and

SELECT

title,

author\_lname,

released\_year

FROM books

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

# advanced

SELECT title,

author\_lname,

released\_year,

stock\_quantity

FROM books

WHERE author\_lname = 'Eggers'

|| released\_year > 2010

OR stock\_quantity > 100;

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

SELECT title, released\_year FROM books

WHERE released\_year NOT BETWEEN 2004 AND 2015;

# logic: in and out

SELECT title, author\_lname FROM books

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

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 % 2 != 0; # IT means the year could be divided by 2

SELECT title, author\_lname FROM books

WHERE SUBSTR(author\_lname,1,1) IN ('C', 'S');

# case function

SELECT title, released\_year,

CASE

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

ELSE '20th Century Lit'

END AS GENRE

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;

# string function

DROP DATABASE IF EXISTS book\_shop;

CREATE DATABASE book\_shop;

USE book\_shop;

CREATE TABLE books

(

book\_id INT NOT NULL AUTO\_INCREMENT,

title VARCHAR(100),

author\_fname VARCHAR(100),

author\_lname VARCHAR(100),

released\_year INT,

stock\_quantity INT,

pages INT,

PRIMARY KEY(book\_id)

);

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

VALUES

('The Namesake', 'Jhumpa', 'Lahiri', 2003, 32, 291),

('Norse Mythology', 'Neil', 'Gaiman',2016, 43, 304),

('American Gods', 'Neil', 'Gaiman', 2001, 12, 465),

('Interpreter of Maladies', 'Jhumpa', 'Lahiri', 1996, 97, 198),

('A Hologram for the King: A Novel', 'Dave', 'Eggers', 2012, 154, 352),

('The Circle', 'Dave', 'Eggers', 2013, 26, 504),

('The Amazing Adventures of Kavalier & Clay', 'Michael', 'Chabon', 2000, 68, 634),

('Just Kids', 'Patti', 'Smith', 2010, 55, 304),

('A Heartbreaking Work of Staggering Genius', 'Dave', 'Eggers', 2001, 104, 437),

('Coraline', 'Neil', 'Gaiman', 2003, 100, 208),

('What We Talk About When We Talk About Love: Stories', 'Raymond', 'Carver', 1981, 23, 176),

("Where I'm Calling From: Selected Stories", 'Raymond', 'Carver', 1989, 12, 526),

('White Noise', 'Don', 'DeLillo', 1985, 49, 320),

('Cannery Row', 'John', 'Steinbeck', 1945, 95, 181),

('Oblivion: Stories', 'David', 'Foster Wallace', 2004, 172, 329),

('Consider the Lobster', 'David', 'Foster Wallace', 2005, 92, 343);

# strings function

select concat('hello',' ', 'world'); #have a try first

select author\_fname as first, author\_lname as last,

concat (author\_lname, ' ', author\_fname) as full #no comma betwee these two lines

from books;

# substring function

SELECT SUBSTRING(title, 1, 10) AS 'short title' FROM books;

SELECT

CONCAT

(

SUBSTRING(title, 1, 10),

'...'

) AS 'short title'

FROM books;

# replace function

# The REPLACE() function, as well as the other string functions, only change the query output, they don't affect the actual data in the database.

SELECT

SUBSTRING(REPLACE(title, 'e', '3'), 1, 10) AS 'weird string'

FROM books;

# reverse function

SELECT CONCAT(author\_fname, REVERSE(author\_fname)) FROM books;

# length functions

SELECT author\_lname, CHAR\_LENGTH(author\_lname) AS 'length' FROM books;

SELECT CONCAT(author\_lname, ' is ', CHAR\_LENGTH(author\_lname), ' characters long') FROM books;

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

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);

# table operation

# select unique ones

SELECT DISTINCT author\_lname FROM books;

# order function;

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

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

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

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

SELECT title, author\_fname, author\_lname

FROM books ORDER BY 3; # = by which volumn

SELECT title, author\_fname, author\_lname

FROM books ORDER BY 1 DESC;

SELECT author\_fname, author\_lname FROM books

ORDER BY author\_lname, author\_fname;

# limit function

SELECT title FROM books LIMIT 3; # the first three rows

SELECT title, released\_year FROM books

ORDER BY released\_year DESC LIMIT 5; # the latest top-five

SELECT title, released\_year FROM books

ORDER BY released\_year DESC LIMIT 0,5; # from the first one, exstract the following five

# like function

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

SELECT title, stock\_quantity FROM books WHERE stock\_quantity LIKE '\_\_\_\_'; # how many digits

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

SELECT title FROM books WHERE title LIKE '%\\_%' # since % and \_ have special meanings, so we need to add \ before each of them

# count function

SELECT COUNT(\*) FROM books; # count rows

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; # to check how many different persons here; it needs to be different for both

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

# group function:

SELECT title, author\_lname FROM books

GROUP BY author\_lname;

SELECT author\_lname, COUNT(\*)

FROM books GROUP BY author\_lname; # frequency of lastnames

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

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

# min and max function

SELECT MIN(released\_year)

FROM books;

SELECT MIN(released\_year) FROM books; # the same

SELECT MAX(pages), title

FROM books;

select \* from books

where pages = (select min(pages) from books)；

select \* from books

order by pages asc limit 1; # the same function for last one

select author\_fname, author\_lname, min(released\_year) # notice there is no space between min and ()

from books

group by author\_lname, author\_fname; # it means the most early work for each author

# sum functions

select sum(pages) from books;

select author\_fname, author\_lname, sum(pages)

from books

group by author\_lname, author\_fname;

# average function

select avg(released\_year)

from books;

select released\_year, avg(stock\_quantity) from books # notice it is not ave, it's avg

group by released\_year;

# one to many tables

# use primary key to link all tables

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)

);

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);

# Finding Orders Placed By George: 2 Step Process

SELECT id FROM customers WHERE last\_name='George';

SELECT \* FROM orders WHERE customer\_id = 1;

# also could be programmed in this way:

SELECT \* FROM orders WHERE customer\_id =

(

SELECT id FROM customers

WHERE last\_name='George'

);

# inner join

SELECT \* FROM customers, orders

WHERE customers.id = orders.customer\_id;

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;

# left join

# by using inner join

SELECT first\_name, last\_name, order\_date, amount

FROM customers

JOIN orders

ON customers.id = orders.customer\_id

ORDER BY order\_date;

SELECT # how many spent for each person

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;

# using left join

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; # we have null rows in this case

# right join function

SELECT \* FROM customers

RIGHT JOIN orders

ON customers.id = orders.customer\_id;

# an example for right join

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

);

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);

# start to use 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;

# many to many tables

# preparation

CREATE TABLE reviewers (

id INT AUTO\_INCREMENT PRIMARY KEY,

first\_name VARCHAR(100),

last\_name VARCHAR(100)

);

CREATE TABLE series(

id INT AUTO\_INCREMENT PRIMARY KEY,

title VARCHAR(100),

released\_year YEAR(4),

genre VARCHAR(100)

);

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)

);

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

# question 1

SELECT

title,

rating

FROM series

JOIN reviews

ON series.id = reviews.series\_id;

# question 2

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;

# question 3

SELECT

first\_name,

last\_name,

rating

FROM reviewers

INNER JOIN reviews

ON reviewers.id = reviews.reviewer\_id;

SELECT # the same

first\_name,

last\_name,

rating

FROM reviews

INNER JOIN reviewers

ON reviewers.id = reviews.reviewer\_id;

# question four

SELECT title AS unreviewed\_series

FROM series

LEFT JOIN reviews

ON series.id = reviews.series\_id

WHERE rating IS NULL;

# question five

SELECT genre,

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

FROM series

INNER JOIN reviews

ON series.id = reviews.series\_id

GROUP BY genre;

# question six

# 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;

# notice: case function is always for multi situations but if function is yes or not

# 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;

# question seven: combine all 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;

# example one

use instragram;

# cloning instagtam's DB: users schema

CREATE TABLE users (

id INTEGER AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(255) UNIQUE NOT NULL,

created\_at TIMESTAMP DEFAULT NOW()

);

# cloning instagtam's DB: phone schema

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)

);

# cloning instagtam's DB: comment schema

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)

);

# cloning ins's DB: Likes Schema

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), # reference + other tables

PRIMARY KEY(user\_id, photo\_id)

);

# A foreign key is a column or group of columns in a relational database table

# that provides a link between data in two tables. It acts as a cross-reference

# between tables because it references the primary key of another table,

# thereby establishing a link between them.

# foreign key could be multi, but primary is unique

# cloning ins's DB: 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)

);

source case1\_instragram\_data.sql

# find five oldest users

select username

from users limit 5

order by created\_at;