```
# Study started 2022-11-21 ~ Present
# This is my study note where I write and store my SQL codes that I
learned from Udemy course.
# The purpose of writing this code is to share my learning, record my
progression, and to lookup codes when I need to.

# 2022-11-21 Study note

# How to create database
CREATE DATABASE hello_my_db;

# How to show database list.
SHOW DATABASE;

# How to delete a database
DROP DATABASE hello_my_db;

# How to create and use database
CREATE DATABASE database_1;
USE database_1;

# How to create a table
# A table with two columns: name and age.
CREATE TABLE customers(
    name varchar(100),
    age int);

# How to show table, table columns, and datatypes.
SHOW TABLES;
SHOW COLUMNS FROM Customers;
DESC Customers;

# How to delete table.
DROP TABLE Customers;

# How to insert data.
INSERT INTO Customers(name, age)
VALUES ("Tina", 13), ("Taylor", 15), ("Park Jin Seok", 24)

# How to show warnings. (Only when we get a warning message)
SHOW WARNINGS;

# How to set NULL and NOT NULL for columns, Default value, Primary Key,
and AUTO_INCREMENT.
# 6 columns: id, last name, first name, middle name, age, and status.
# Id, last name, first name, age and status cannot be NULL and id is our
unique primary key. Set default value to 'employed' for status. Use
AUTO_INCREMENT for id column.
CREATE TABLE Employees(
    id int AUTO_INCREMENT NOT NULL PRIMARY KEY,
    first_name varchar(255) NOT NULL,
    last_name varchar(255) NOT NULL,
    middle_name varchar(255),
    age int NOT NULL,
```

```
    current_status varchar(50) NOT NULL DEFAULT "Employed"
);

# 2022-11-21
# Before we start with SELECT clause, let's create a dataset. The table
name is called "Cats" and it contains cat_id, name, breed, and age.
CREATE TABLE Cats(
    cat_id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    name varchar(100),
    breed varchar(100),
    age INT);

# Then, we insert values into the table.
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);

# Now, let's use SELECT clause to call out some columns that we want to
inspect from Cats table.
SELECT * FROM Cats; # Select all columns from Cats table.
SELECT name FROM Cats; # Select name column from Cats table.
SELECT cat_id, name, age FROM Cats; # Select multiple columns from Cats
table.

# Use WHERE clause to specify conditions to filter out data values.
SELECT * FROM Cats
WHERE age = 4; # Get all data values with age equals to 4.

SELECT * FROM Cats
WHERE name = "Egg" # Get all data values with name equals to Egg.

SELECT cat_id, name, age FROM Cats
WHERE breed = "Tabby";

# Use UPDATE clause to change values in Cats table. Change "Tabby" to
"Shorthair".
UPDATE Cats SET breed = "Shorthair"
WHERE breed = "Tabby";

# Change Misty's age to 14.
UPDATE Cats SET age = 14
WHERE name = "Misty";

# Change Jackson's name to "Jack".
UPDATE Cats SET name = "Jack"
WHERE name = "Jackson";

# Change Maine Coons' ages to be 12.
UPDATE Cats SET age = 12
```

```
WHERE breed = "Maine Coon";

# We can also use DELETE clause to delete data values.
DELETE FROM Cats
WHERE name = "Egg";

# Delete all 4 year old cats.
DELETE FROM Cats
WHERE age = 4;

# Delete cats whose age is the same as their cat_id.
DELETE FROM Cats
WHERE age = Cat_id;

# 2022-11-25
# How to open sql file to our terminal? We can use SOURCE clause.
source book_data.sql;

# How can we merge two columns together? we can use CONCAT clause to
concatenate two columns. For example,
SELECT CONCAT(auther_fname, " ", auther_lname) AS author_fullname
FROM books; # We can merge author's first name and last name together to
create another column named author's full name.

# We can use SUBSTRING to extract specific number of strings.
SELECT SUBSTRING("Hello, my name is Taehwan Kim", 1,10); # This would
result "Hello, my ".

# Or we can cut out number of strings from the start.
SELECT SUBSTRING("Hello, my name is Taehwan Kim",8); # This would result
"my name is Taehwan Kim:.

# Or we can start counting from the back of the string.
SELECT SUBSTRING("Hello, my name is Taehwan Kim", -11); # This would
result "Taehwan Kim".

# What if we want to replace string with some other text? We can use
REPLACE clause to replace text.
SELECT REPLACE("Hello, my name is Taehwan Kim", "Taehwan", "Chris"); #
This would result "Hello, my name is Chris Kim".

# When we need to count total length of string, we can use CHAR_LENGTH
clause to check.
SELECT CHAR_LENGTH("Hello, my name is Taehwan Kim"); # The total length of
this string is 29.

# Combining all these clauses together can do...
SELECT SUBSTRING(CONCAT("Hello, my name is", " ", REPLACE("Taehwan
Kim","Taehwan", "Chris")),7),
        CONCAT("The total length of this string is", " ",
CHAR_LENGTH(SUBSTRING(CONCAT("Hello, my name is", " ", REPLACE("Taehwan
Kim","Taehwan", "Chris")),7)), ".");

# Exercise
```

```
source book_data.sql;

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;

# 2022-11-28
# We can use DISTINCT clause to remove duplicate records from the table.
For instance, if we want to check all the name of authors in the table and
the fact that some of authors in the table published multiple books,
# We might want to use DISTINCT clause to avoid to have duplicate author's
name.

SELECT DISTINCT author_fname AS First_name, author_lname AS Last_name
FROM books;

# We can use ORDER BY clause to order values in specific orders.
# ORDER BY author_lname (TEXT).
SELECT author_lname
FROM books
ORDER BY author_lname; # We can add DESC at the end to choose dsecending
order.

# ORDER BY released_year (NUMERIC).
SELECT released_year
FROM books
ORDER BY released_year;

# We can use LIMIT clause to limit number of results. For example,
retrieving 2 bestselling books and 2 is our limit to our result.
SELECT released_year
FROM books
ORDER BY released_year DESC
LIMIT 3;

# Using Wildcard like %, _ and LIKE clause to search values;
SELECT title, author_fname FROM books
WHERE author_fname LIKE "%da%"; # Retrieving values with "da" in it.

SELECT stock_quantity FROM books
WHERE stock_quantity LIKE "____"; # Retreiving values with 4 digits.

SELECT title FROM books
WHERE title LIKE "%\%%"; # Retrieving values with "%" in it.

# Exercise
#Select all story collections titles that contain "Stories".
SELECT title FROM books
WHERE title LIKE "%stories%";

# Find the longest title of book. Print out the Title and page count.
SELECT title, pages FROM books
ORDER BY CHAR_LENGTH(title) DESC
```

```
LIMIT 1;

# Find the longest book. Print out the Title and page count.
SELECT title, pages FROM books
ORDER BY pages DESC
LIMIT 1;

# Print a summary containing the title and year, for the 3 most recent
books.
SELECT CONCAT(title," - ",released_year) AS summary
FROM books
ORDER BY released_year DESC
LIMIT 3;

# Find all books with an author_lname that contains a space(" ").
SELECT title, author_lname
FROM books
WHERE author_lname LIKE "% %";

# Find the 3 books with the lowest stock. Select title, year, and stock.
SELECT title, released_year, stock_quantity
FROM books
ORDER BY stock_quantity
LIMIT 3;

# Print title and author_lname, sorted first by author_lname and then by
title
SELECT title, author_lname
FROM books
ORDER BY author_lname, title;

# Yell "MY FAVORITE AUTHOR IS [author_fullname]!" and sorted
alphabetically by last name.
SELECT CONCAT("MY FAVORITE AUTHOR IS ",UPPER(CONCAT(author_fname, "
",author_lname)), "!") AS Yell
FROM books
ORDER BY author_lname;

#2022-11-29
# We can use COUNT clause to count all the number of rows in the table.
SELECT COUNT(*) FROM books;

# How many DISTINCT author_fnames?
SELECT COUNT(DISTINCT(author_fname)) FROM books;

# How many titles contain "the"?
SELECT COUNT(title)
FROM books
WHERE title LIKE "%the%";

# GROUP BY clause: Count how many books each author has written.
SELECT author_fname,author_lname, COUNT(*) AS number_of_books
FROM books
GROUP BY author_lname, author_fname
```

```
ORDER BY COUNT(*) DESC;

# Subquary using MIN and MAX functions.
# How to find longest book and what is the title of the book?
SELECT * FROM books
WHERE pages = (SELECT MAX(pages)
               FROM books); # This is our subquary and the computer will
read this subquary first before it reads the first SELECT clause.
                            # Which means, whatever value that we're
getting from subquary will be our values to execute in our first SELECT
cluase.

# Find the year each author published their first book
SELECT author_fname, author_lname, MIN(released_year)
FROM books
GROUP BY author_lname, author_fname;

# Find the longest page count for each author
SELECT author_fname, author_lname, MAX(pages)
FROM books
GROUP BY author_lname, author_fname;

# Sum all pages each author has written by using SUM and GROUP BY clause.
SELECT CONCAT(author_fname, " ", author_lname), sum(pages)
FROM books
GROUP BY author_lname, author_fname;

# Calculate the average stock quantity for books released in the same
year.
SELECT released_year, AVG(stock_quantity)
FROM books
GROUP BY released_year;

# Calcuate the average pages each author writes.
SELECT author_fname, author_lname, AVG(pages) AS Average_pages
FROM books
GROUP BY author_lname, author_fname;

# Exercise
# Print the number of books in the database
SELECT COUNT(*) FROM books;

# Print out how many books were released in each year
SELECT released_year, COUNT(*) FROM books
GROUP BY released_year
ORDER BY released_year DESC;

# Print out the total number of books in stock
SELECT SUM(stock_quantity) AS total_number_of_books FROM books;

# Find the average released_year for each author
SELECT author_fname, author_lname, AVG(released_year)
FROM books
GROUP BY author_lname, author_fname;
```

```
# Find the full name of the author who wrote the longest book
SELECT CONCAT(author_fname," ", author_lname) AS Name, title
FROM books
WHERE pages = (SELECT MAX(pages) from books);


#2022-11-30
# Formatting Dates (and times)
# DAY() - Extract day. Ex. 11th = 11, 30th = 30 etc.
# MONTHNAME() - Extract what month it is. Ex. 2012-11-23 = November etc.
# YEAR() - Extract what year it is. Ex. 2012-11-23 = 2012 etc.
# DAYNAME() - Extract day in text. Ex. Monday, Tuesday etc.
# DAYOFWEEK() - Extract day in a number format 1-7. Ex. Friday = 6,
Saturday = 7 etc.
# DAYOFYEAR() - Extract number of days passed from beginning of the year.
EX. 2017-04-21 = 111 days etc.

# HOUR() - Extract what hour it is. Ex. 22:12:43 = 22 hours etc.
# MINUTE() - Extract what minute it is. Ex. 22:12:43 = 12 minutes etc.

# We can use DATE_FORMAT clause to change the format of date.
SELECT DATE_FORMAT("2012-10-30 22:23:12", "%W %M %Y");
SELECT DATE_FORMAT("2012-10-30 22:23:12", "%m/%d/%Y");

# We can calculate the date difference between two datetime with DATEDIFF
cluase. For example, if we want to calculate the number of days between
2022-01-01 and 2022-12-31, we can do...
SELECT DATEDIFF("2022-12-31", "2021-12-31"); # Number of days in one year.
SELECT DATEDIFF(NOW(), "1997-01-27"); # I lived 9439 days from days that I
was born.

# DATE_ADD clause can be used to add interval of dates from the time we
choose.
SELECT "2022-11-30" AS Today, DATE_ADD("2022-11-30", INTERVAL 2 MONTH) AS
After_today;

# Exercise
# Create a table with price < 1,000,000.
CREATE TABLE inventory(
    item_name VARCHAR(50),
    price DECIMAL(8,2),
    quantity INT
);

# Print out the current time.
SELECT CURRENT_TIME;

# Print out the current day of the week.
SELECT DAYOFWEEK(NOW());

# Print out the current day of the week (The day name).
SELECT DAYNAME(NOW());
```

```
# Print out the current day and time using this format: mm/dd/yyyy.
SELECT DATE_FORMAT(NOW(), "%m/%d/%Y");

# Print out the current day and time using this format: January 2nd at
3:!5.
SELECT DATE_FORMAT(NOW(), "%M %D at %h:%m");

# Create a tweets table that stores: - The tweet content, a username, time
it was created.
CREATE TABLE TWEETS(
    content VARCHAR(500),
    username VARCHAR(25),
    time_at TIMESTAMP DEFAULT NOW()
);

INSERT INTO TWEETS(content, username)
VALUES ("I know how to code this", "mysql.123");

SELECT * FROM TWEETS;

# 2022-12-02
# Logical Operators
# Not equal != clause
SELECT title FROM books
WHERE released_year != 2017;

# NOT LIKE clause
SELECT title FROM books
WHERE title NOT LIKE "W%";

# Greater than >, less than < clauses
SELECT released_year FROM books
WHERE released_year >= 2000 AND released_year <= 2017;

# CASE STATEMENTS ++ We can use case statement as a conditional statement
that we're familiar with in programming language.
SELECT title, released_year,
        CASE
            WHEN released_year >= 2000 THEN "Modern Lit"
            ELSE "20th Centuary 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;

# Exercise
# Select all books written by Eggers or Chabon.
SELECT title FROM books
```

```
WHERE author_lname = "Eggers" OR author_lname = "Chabon";

# Select all books written by Lahiri Published after 2000.
SELECT title from books
WHERE released_year >= 2000;

# Select all books with page counts between 100 and 200.
SELECT title FROM books
WHERE pages BETWEEN 100 AND 200;

# Select all books where author_lname starts with a "C" or an "S".
SELECT title FROM books
WHERE author_lname LIKE "C%" OR "S%"

# Select title, author_lname and make a column named "type" that if title
contains "stories", set it as "Short Stories", if title contains "Just
kids" or "A heartbreaking Work", set it as "Memoir", and
# everything else, set it as "Novel".
SELECT title, author_lname,
        CASE
            WHEN title LIKE "%stories%" THEN "Short Stories"
            WHEN title LIKE "%Just Kids%" OR title LIKE "%A Heartbreaking
Work%" THEN "Memoir"
            ELSE "Novel"
        END AS type
FROM books;

# CASE STATEMENT + GROUP BY clauses.
SELECT title, author_lname,
        CASE
            WHEN COUNT(author_lname) = 1 THEN "1 book"
            WHEN COUNT(author_lname) >= 2 THEN CONCAT(COUNT(author_lname),
" books")
        END AS COUNT
FROM books
GROUP BY author_lname, author_fname;

# 2022-12-04
# NEW 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);

# Interaction between two tables with using subquary.
SELECT * FROM orders
WHERE customer_id = (SELECT * FROM customers
                     WHERE last_name = "George");

# Simple version of above coding with INNER JOIN clause.
SELECT first_name, last_name, order_date, amount FROM customers
INNER JOIN orders
            ON customers.id = orders.customer_id
ORDER BY order_date DESC;

# Using LEFT JOIN clause to find the total amount of money that the
customers spent.
SELECT first_name, last_name, IFNULL(SUM(amount), 0) AS Total
FROM customers
LEFT JOIN orders
            ON customers.id = orders.customer_id
GROUP BY customers.id
ORDER BY Total DESC;

# Exercise
# Write this schema. Two tables of students and papers. The student table
contains id and first_name and paper table contains title, grade, and
student_id but the student_id is a foregin key referencing to students.id.

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

CREATE TABLE papers(
    title VARCHAR(100),
    grade INT(3),
    student_id INT,
    FOREIGN KEY(student_id) REFERENCES students(id)
);

# INSERT THIS 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);

# Print first_name, title, grade with matching id from both tables and
order them by highest to lowest grade.
SELECT first_name, title, grade
FROM students
INNER JOIN papers
           ON students.id = papers.student_id
ORDER BY grade DESC;

# LEFT JOIN students to papers and then change NULL with appropriate value
such as "Missing" for title and "0" for grade.
SELECT first_name, IFNULL(title, "Missing") AS title , IFNULL(grade, 0) AS
grade FROM students
LEFT JOIN papers
            ON students.id = papers.student_id;

# Print first name and average of student's grade.
SELECT first_name, IFNULL(AVG(grade),0) AS average
FROM students
LEFT JOIN papers
            ON students.id = papers.student_id
GROUP BY first_name
ORDER BY average DESC;

# Print first name, average grade, and passing_status which is the student
who achieved above 75 in average receives "PASSING" and below 75 will
receives "FAILING".
SELECT first_name, IFNULL(AVG(grade),0) AS average,
        CASE
            WHEN IFNULL(AVG(grade),0) < 75 THEN "FAILING"
            WHEN IFNULL(AVG(grade),0) >= 75 THEN "PASSING"
        END AS passing_status
FROM students
LEFT JOIN papers
            ON students.id = papers.student_id
GROUP BY first_name
ORDER BY average DESC;

# 2022-12-05
# Many to Many
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);

# Print title and rating from series and reviews.
SELECT title, rating FROM series
INNER JOIN reviews
            ON series.id = reviews.series_id;

# Print title and average rating from series and reviews. Then, group them
by title.
SELECT title, AVG(rating) AS average_rating
FROM series
INNER JOIN reviews
            ON series.id = reviews.series_id
GROUP BY title
ORDER BY average_rating;

# Print first name and last name, and rating.
SELECT first_name, last_name, rating
FROM reviewers
INNER JOIN reviews
            ON reviews.reviewer_id = reviewers.id;

# Print unreviewed_Series.
SELECT title AS Unreviewed_series
FROM series
LEFT JOIN reviews
            ON series.id = reviews.series_id
WHERE rating is NULL;

# Print genre and average_rating.
SELECT genre, AVG(rating) AS average_rating
FROM series
INNER JOIN reviews
            ON series.id = reviews.series_id
GROUP BY genre;

# Analytic table of reviewers with first_name, last_name, count of
reviews, Min rating, MAX rating, AVG rating, Status of reviewers.
SELECT first_name,last_name, count(rating) AS COUNT, IFNULL(MIN(rating),0)
AS MIN, IFNULL(MAX(rating),0) AS MAX,IFNULL(AVG(rating),0) 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
ORDER BY COUNT DESC;
```

```
# Exercise
# Print title, rating, and reviewers full name. Interconnect 3 tables
together with 2 INNER JOIN clauses.
SELECT title, rating, CONCAT(first_name, " ", last_name) AS reviewers
FROM series
INNER JOIN reviews
            ON series.id = reviews.series_id
INNER JOIN reviewers
            ON series.id = reviewers.id
ORDER BY title;

# 2022-12-09
# INSTAGRAM Datasets.
# Creating multiple tables. Users, photos, comments, likes, follows, tags,
and photo_tags.
CREATE DATABASE ig_database;
USE ig_database;

CREATE TABLE users(
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) UNIQUE,
    created_at TIMESTAMP DEFAULT NOW()
);

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

CREATE TABLE comments(
    id INT AUTO_INCREMENT PRIMARY KEY,
    comment_text VARCHAR(255) NOT NULL,
    user_id INT NOT NULL,
    photo_id INT 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 INT AUTO_INCREMENT,
    photo_id INT 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 INT NOT NULL,
```

```
    followee_id INT NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    FOREIGN KEY(follower_id) REFERENCES user(id),
    FOREIGN KEY(followee_id) REFERENCES user(id),
    PRIMARY KEY(follower_id, followee_id)
);

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

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

# 2022-12-09
# INSTAGRAM Datasets.
# We're going to use ig_clone_data.sql file to load tables inside and
values to insert in those tables.
# Create a database for this exercise and use it.
DROP DATABASE ig_clone;
CREATE DATABASE ig_clone;
USE ig_clone;

# Load the file.
source ig_clone_data.sql;

# Exercise
# We want to reward our users who have been around the longest. Find the 5
oldest users.
SELECT * FROM users
ORDER BY created_at
LIMIT 5;

# What day of the week do most users registered on? We need to figure out
when to schedule an ad campaign.
SELECT DAYNAME(created_at) AS day_name, COUNT(*) AS COUNT
FROM users
GROUP BY day_name
ORDER BY COUNT DESC;

# We want to target out inactive users with an email campaign. Find the
users who have never posted a photo.
SELECT username, IFNULL(image_url, "INACTIVE") AS image_url
FROM photos
RIGHT JOIN users
            ON photos.user_id = users.id
WHERE image_url IS NULL;
```

```
# We're running a new contest to see who can get the most likes on a
single photo. Who won?
SELECT username, photos.id , photos.image_url, COUNT(*)  AS likes
FROM photos
INNER JOIN likes
            ON likes.photo_id = photos.id
INNER JOIN users
            ON
            users.id = photos.user_id
GROUP BY id
ORDER BY likes DESC
LIMIT 1;

# How many times does the average user post?
SELECT (SELECT COUNT(*) FROM photos)/(SELECT COUNT(*) FROM users) AS
Average;

# A brand wants to know which hashtags to use in a post. What are the top
5 most commonly used hashtags?
SELECT tag_name, COUNT(*) AS Total
FROM tags
INNER JOIN photo_tags
            ON tags.id = photo_tags.tag_id
GROUP BY tag_name
ORDER BY Total DESC
LIMIT 5;

# We have a small problem with bots on our site. Find users who have liked
every single photo on the site.
SELECT username, users.created_at,
        CASE
            WHEN COUNT(*) = (SELECT COUNT(*) FROM photos) THEN "Bot"
            ELSE "User"
        END AS STATUS
FROM users
INNER JOIN likes
            ON users.id = likes.user_id
GROUP BY username
HAVING STATUS = "BOT"
ORDER BY STATUS;

#2022-12-12
# MySQL Database Triggers
# The Syntax
# CREATE TRIGGER trigger_name
#    trigger_time trigger_event ON table_name FOR EACH ROW
#    BEGIN
#    ...
#    END;

# Example 1: A Simple Validation
CREATE TABLE users(
    username VARCHAR(255),
```

```
    age INT,
    created_at TIMESTAMP DEFAULT NOW()
);

DELIMITER $$

CREATE TRIGGER must_be_adult
    BEFORE INSERT ON users FOR EACH ROW
    BEGIN
        IF NEW.age < 18
        THEN
            SIGNAL SQLSTATE '45000' # 45000 is a genertic state
representing unhandled user-defined exception.
                    SET MESSAGE_TEXT = "Must be an adult.";
        END IF;
    END;
$$

DELIMITER    ;

# This will prevent us from adding underage user to the table and whenver
we're trying to insert in underage user to the table, it will appear a
warning message to our system.

# Example 2: Preventing Self-Follows
use ig_clone;

DELIMITER $$

CREATE TRIGGER prevent_self_follows
    BEFORE INSERT ON follows FOR EACH ROW
    BEGIN
        IF NEW.follower_id = NEW.followee_id
        THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = "You cannot follow yourself.";
        END IF;
    END;
$$

DELIMITER    ;

INSERT INTO follows(follower_id, followee_id)
VALUES (5,5);

# This will show a warning message that says "You cannot follow yourself."
since follower_id and followee_id are identical.

# Exercise 3: logging unfollower.
# We're going to add a table to our ig_clone database named unfollows.
This table will going to interect with follows table that whenever we
delete someone from follows table,
# We're going to add that person to unfollows table so that we can logging
unfollowers.
```

```
USE ig_clone

CREATE TABLE unfollows(
    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)
);

DELIMITER $$

CREATE TRIGGER capture_unfollow
    AFTER DELETE ON follows FOR EACH ROW
    BEGIN
        INSERT INTO unfollows
        SET follower_id = OLD.follower_id,
            followee_id = OLD.followee_id;
    END;
$$
DELIMITER    ;

DELETE FROM follows
WHERE follower_id = 2 AND followee_id = 1; # Unfollowed.

SELECT * FROM unfollows; # Unfollow captured in unfollow table.

SHOW TRIGGERS;  # This will show the list of what triggers we set.
DROP TRIGGERS trigger_name;  # This will delete triggers we have
previously set before.

#2022-12-12 Course ended.
```