CODE: Your First 5 Minutes of SQL

Section 1, Lecture 5

Your First 5 Minute of SQL CODE

Resources:

- SQL Try-It Editor
- Lecture Slides

Step 1.

```
SELECT * FROM customers;
```

Step 2.

```
SELECT * FROM orders;
```

Step 3.

```
SELECT *
FROM products
ORDER BY Price DESC;
```

Step 4.

```
SELECT
 customerName,
 COUNT(*) AS 'number of orders'
FROM customers
INNER JOIN orders
 ON orders.customerID = customers.customerID
GROUP BY customers.customerID;
```

CODE: Installing MySQL on Cloud9

Section 2, Lecture 16

```
mysql-ctl start
```

```
mysql-ctl cli
```

```
mysql-ctl stop
```

```
exit;
```

```
quit;
```

```
\q;
```

```
ctrl-c
```

CODE: Your First MySQL Activity

Section 2, Lecture 18

```
help;

show databases;

select @@hostname;
```

CODE: Mac Installation

Section 2, Lecture 20

MAC INSTALLATION INSTRUCTIONS - CODE FROM THE VIDEO

ONLY RECOMMENDED FOR EXPERIENCED DEVELOPERS.
SERIOUSLY, JUST USE CLOUD9
AND THEN RETURN TO THIS VIDEO AFTER YOU GRADUATE!

Once you've downloaded and started up the MySQL server...

Add the following line to your .bash_profile or .zshrc file

```
export PATH=${PATH}:/usr/local/mysql/bin/
```

Then run:

```
Mysql -u root -p
```

And enter the password you received when you initially installed.

Finally, create a new password for the root user by running:

```
ALTER USER 'root'@'localhost' IDENTIFIED BY
 'yournewpassword'
```

Note: Some students have mentioned that they get errors instead of warnings when using MySQL for Mac (and possibly Windows/Linux). If you run into this error then you can try the following solution to resolve the issue:

Take your setup out of strict mode with the following command:

```
SET @@global.sql_mode= '';
```

CODE: Creating Databases

Section 3, Lecture 24

Creating Databases Code

Start the CLI:

```
mysql-ctl cli;
```

List available databases:

```
show databases;
```

The general command for creating a database:

```
CREATE DATABASE database_name;
```

A specific example:

```
CREATE DATABASE soap_store;
```

CODE: Dropping Databases

Section 3, Lecture 26

To drop a database:

```
DROP DATABASE database_name;
```

For Example:

```
DROP DATABASE hello_world_db;
```

Remember to be careful with this command! Once you drop a database, it's gone!

CODE: Using Databases

Section 3, Lecture 28

```
USE <database name>;

-- example:
USE dog_walking_app;

SELECT database();
```

CODE: Creating Your Own Tables

Section 3, Lecture 34

```
CREATE TABLE tablename
  (
    column_name data_type,
    column_name data_type
  );
```

```
CREATE TABLE cats
  (
    name VARCHAR(100),
    age INT
  );
```

CODE: How Do We Know It Worked?

Section 3, Lecture 36

```
SHOW TABLES;

SHOW COLUMNS FROM tablename;

DESC tablename;
```

CODE: Dropping Tables

Section 3, Lecture 38

Dropping Tables

`DROP TABLE <tablename>;`

A specific example:

`DROP TABLE cats;`

Be careful with this command!

CODE: Creating Your Own Tables Challenge

Section 3, Lecture 40

```
CREATE TABLE pastries
  (
    name VARCHAR(100),
    quantity INT
  );

SHOW TABLES;

DESC pastries;

DROP TABLE pastries;
```

CODE: Inserting Data

Section 4, Lecture 43

Inserting Data

The "formula":

```
INSERT INTO table_name(column_name) VALUES (data);
```

For example:

```
INSERT INTO cats(name, age) VALUES ('Jetson', 7);
```

CODE: Super Quick Intro To SELECT

Section 4, Lecture 45

```
SELECT * FROM cats;
```

CODE: Multiple Insert

Section 4, Lecture 47

```
INSERT INTO table_name
            (column_name, column_name)
VALUES      (value, value),
            (value, value),
            (value, value);
```

CODE: INSERT Challenges Solution

Section 4, Lecture 50

INSERT Challenge Solution Code

```
CREATE TABLE people
  (
    first_name VARCHAR(20),
    last_name VARCHAR(20),
    age INT
  );

INSERT INTO people(first_name, last_name, age)
VALUES ('Tina', 'Belcher', 13);

INSERT INTO people(age, last_name, first_name)
VALUES (42, 'Belcher', 'Bob');

INSERT INTO people(first_name, last_name, age)
VALUES('Linda', 'Belcher', 45)
  ,('Phillip', 'Frond', 38)
  ,('Calvin', 'Fischoeder', 70);
```

`DROP TABLE people;`

`SELECT * FROM people;`

`show tables;`

CODE: MySQL Warnings

Section 4, Lecture 52

MySQL Warnings Code

`DESC cats;`

Try Inserting a cat with a super long name:

```
INSERT INTO cats(name, age)
VALUES('This is some text blah blah blah blah blah text text text something about cats lalala
lal meowwwwwwwwwwww', 10);
```

Then view the warning:

`SHOW WARNINGS;`

Try inserting a cat with incorrect data types:

```
INSERT INTO cats(name, age) VALUES('Lima',
 'dsfasdfdas');
```

Then view the warning:

`SHOW WARNINGS;`

CODE: NULL and NOT NULL

Section 4, Lecture 54

NULL and NOT NULL Code

Try inserting a cat without an age:

`INSERT INTO cats(name) VALUES('Alabama');`

`SELECT * FROM cats;`

Try inserting a nameless and ageless cat:

`INSERT INTO cats() VALUES();`

Define a new cats2 table with NOT NULL constraints:

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

`DESC cats2;`

Now try inserting an ageless cat:

`INSERT INTO cats2(name) VALUES('Texas');`

View the new warnings:

`SHOW WARNINGS;`

`SELECT * FROM cats2;`

Do the same for a nameless cat:

`INSERT INTO cats2(age) VALUES(7);`

`SHOW WARNINGS;`

CODE: Setting Default Values

Section 4, Lecture 56

CODE: Setting Default Values

Define a table with a DEFAULT name specified:

```
CREATE TABLE cats3
  (
    name VARCHAR(20) DEFAULT 'no name provided',
    age INT DEFAULT 99
  );
```

Notice the change when you describe the table:

`DESC cats3;`

Insert a cat without a name:

`INSERT INTO cats3(age) VALUES(13);`

Or a nameless, ageless cat:

`INSERT INTO cats3() VALUES();`

Combine NOT NULL and DEFAULT:

```
CREATE TABLE cats4
  (
    name VARCHAR(20) NOT NULL DEFAULT ‘unnamed’,
    age INT NOT NULL DEFAULT 99
  );
```

Notice The Difference:

```
INSERT INTO cats() VALUES();

SELECT * FROM cats;

INSERT INTO cats3() VALUES();

SELECT * FROM cats3;

INSERT INTO cats3(name, age) VALUES('Montana', NULL);

SELECT * FROM cats3;
INSERT INTO cats4(name, age) VALUES('Cali', NULL);
```

CODE: A Primer on Primary Keys

Section 4, Lecture 58

CODE: Primary Keys

Define a table with a PRIMARY KEY constraint:

```
CREATE TABLE unique_cats
  (
    cat_id INT NOT NULL,
    name VARCHAR(100),
    age INT,
    PRIMARY KEY (cat_id)
  );
```

`DESC unique_cats;`

Insert some new cats:

```
INSERT INTO unique_cats(cat_id, name, age) VALUES(1, 'Fred', 23);

INSERT INTO unique_cats(cat_id, name, age) VALUES(2, 'Louise', 3);

INSERT INTO unique_cats(cat_id, name, age) VALUES(1, 'James', 3);
```

Notice what happens:

`SELECT * FROM unique_cats;`

Adding in AUTO_INCREMENT:

```
CREATE TABLE unique_cats2 (
    cat_id INT NOT NULL AUTO_INCREMENT,
    name VARCHAR(100),
    age INT,
    PRIMARY KEY (cat_id)
);
```

INSERT a couple new cats:

```
INSERT INTO unique_cats2(name, age) VALUES('Skippy', 4);
INSERT INTO unique_cats2(name, age) VALUES('Jiff', 3);
INSERT INTO unique_cats2(name, age) VALUES('Jiff', 3);
INSERT INTO unique_cats2(name, age) VALUES('Jiff', 3);
INSERT INTO unique_cats2(name, age) VALUES('Skippy', 4);
```

Notice the difference:

`SELECT * FROM unique_cats2;`

CODE: Table Constraints Exercise Solution

Section 4, Lecture 61

Table Constraints Exercise Solution

Defining The employees table:

```
CREATE TABLE employees (
    id INT AUTO_INCREMENT NOT NULL,
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    middle_name VARCHAR(255),
    age INT NOT NULL,
    current_status VARCHAR(255) NOT NULL DEFAULT 'employed',
    PRIMARY KEY(id)
);
```

Another way of defining a primary key:

```
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(255) NOT NULL DEFAULT 'employed'
);
```

A test INSERT:

```
INSERT INTO employees(first_name, last_name, age) VALUES
('Dora', 'Smith', 58);
```

CODE: Introduction to CRUD

Section 5, Lecture 64

```
INSERT INTO cats(name, age) VALUES(‘Taco’, 14);
```

CODE: Preparing Our Data

Section 5, Lecture 66

CODE: Preparing Our Data

Let's drop the existing cats table:

```
DROP TABLE cats;
```

Recreate a new cats table:

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

```
DESC cats;
```

And finally insert some new cats:

```
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);
```

CODE: Official Introduction to SELECT

Section 5, Lecture 68

Various Simple SELECT statements:

```
SELECT * FROM cats;
```

```
SELECT name FROM cats;
```

```
SELECT age FROM cats;
```

```
SELECT cat_id FROM cats;
```

```
SELECT name, age FROM cats;
```

```
SELECT cat_id, name, age FROM cats;
```

```
SELECT age, breed, name, cat_id FROM cats;
```

```
SELECT cat_id, name, age, breed FROM cats;
```

CODE: Introduction to WHERE

Section 5, Lecture 70

CODE: Introduction to WHERE

Select by age:

```
SELECT * FROM cats WHERE age=4;
```

Select by name:

```
SELECT * FROM cats WHERE name='Egg';
```

Notice how it deals with case:

```
SELECT * FROM cats WHERE name='egG';
```

CODE: SELECT Challenges Solution

Section 5, Lecture 73

CODE: Select Challenges Solution

```
SELECT cat_id FROM cats;
```

```
SELECT name, breed FROM cats;
```

```
SELECT name, age FROM cats WHERE breed='Tabby';
```

```
SELECT cat_id, age FROM cats WHERE cat_id=age;
```

```
SELECT * FROM cats WHERE cat_id=age;
```

CODE: Introduction to Aliases

Section 5, Lecture 75

CODE: Introduction to Aliases

```
SELECT cat_id AS id, name FROM cats;

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

DESC cats;
```

CODE: The UPDATE Command

Section 5, Lecture 77

CODE: Updating Data

Change tabby cats to shorthair:

```
UPDATE cats SET breed='Shorthair' WHERE breed='Tabby';
```

Another update:

```
UPDATE cats SET age=14 WHERE name='Misty';
```

CODE: UPDATE Challenges Solution

Section 5, Lecture 80

CODE: Update Challenges Solution

```
SELECT * FROM cats WHERE name='Jackson';

UPDATE cats SET name='Jack' WHERE name='Jackson';

SELECT * FROM cats WHERE name='Jackson';

SELECT * FROM cats WHERE name='Jack';

SELECT * FROM cats WHERE name='Ringo';

UPDATE cats SET breed='British Shorthair' WHERE name='Ringo';

SELECT * FROM cats WHERE name='Ringo';

SELECT * FROM cats;

SELECT * FROM cats WHERE breed='Maine Coon';

UPDATE cats SET age=12 WHERE breed='Maine Coon';

SELECT * FROM cats WHERE breed='Maine Coon';
```

CODE: Introduction to DELETE

Section 5, Lecture 82

CODE: DELETING DATA

```
DELETE FROM cats WHERE name='Egg';

SELECT * FROM cats;

SELECT * FROM cats WHERE name='egg';

DELETE FROM cats WHERE name='egg';

SELECT * FROM cats;

DELETE FROM cats;
```

CODE: DELETE Challenges Solution

Section 5, Lecture 85

CODE: DELETE Challenges Solution

```
SELECT * FROM cats WHERE age=4;

DELETE FROM cats WHERE age=4;

SELECT * FROM cats WHERE age=4;

SELECT * FROM cats;

SELECT *  FROM cats WHERE cat_id=age;

DELETE FROM cats WHERE cat_id=age;

DELETE FROM cats;

SELECT * FROM cats;
```

CODE: CRUD Exercise Create Solution

Section 6, Lecture 89

```
SELECT database();

CREATE DATABASE shirts_db;

use shirts_db;

SELECT database();

CREATE TABLE shirts
  (
    shirt_id INT NOT NULL AUTO_INCREMENT,
    article VARCHAR(100),
    color VARCHAR(100),
    shirt_size VARCHAR(100),
    last_worn INT,
    PRIMARY KEY(shirt_id)
  );

DESC shirts;

INSERT INTO shirts(article, color, shirt_size, last_worn) VALUES
('t-shirt', 'white', 'S', 10),
('t-shirt', 'green', 'S', 200),
('polo shirt', 'black', 'M', 10),
('tank top', 'blue', 'S', 50),
('t-shirt', 'pink', 'S', 0),
('polo shirt', 'red', 'M', 5),
('tank top', 'white', 'S', 200),
('tank top', 'blue', 'M', 15);

SELECT * FROM shirts;

INSERT INTO shirts(color, article, shirt_size, last_worn)
VALUES('purple', 'polo shirt', 'medium', 50);

SELECT * FROM shirts;
```

CODE: CRUD Exercise Read Solution

Section 6, Lecture 91

```
SELECT article, color FROM shirts;

SELECT * FROM shirts WHERE shirt_size='M';

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

CODE: CRUD Exercise Update Solution

Section 6, Lecture 93

```
SELECT * FROM shirts WHERE article='polo shirt';

UPDATE shirts SET shirt_size='L' WHERE article='polo shirt';

SELECT * FROM shirts WHERE article='polo shirt';

SELECT * FROM shirts;

SELECT * FROM shirts WHERE last_worn=15;

UPDATE shirts SET last_worn=0 WHERE last_worn=15;

SELECT * FROM shirts WHERE last_worn=15;

SELECT * FROM shirts WHERE last_worn=0;

SELECT * FROM shirts WHERE color='white';

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

SELECT * FROM shirts WHERE color='white';

SELECT * FROM shirts WHERE color='off white';

SELECT * FROM shirts;
```

CODE: CRUD Exercise Delete Solution

Section 6, Lecture 95

```
SELECT * FROM shirts;

SELECT * FROM shirts WHERE last_worn=200;

DELETE FROM shirts WHERE last_worn=200;

SELECT * FROM shirts WHERE article='tank top';

DELETE FROM shirts WHERE article='tank top';

SELECT * FROM shirts WHERE article='tank top';

SELECT * FROM shirts;

DELETE FROM shirts;

SELECT * FROM shirts;

DROP TABLE shirts;

show tables;

DESC shirts;
```

CODE: Running SQL Files

Section 7, Lecture 98

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

mysql-ctl cli

use cat_app;

source first_file.sql

DESC cats;



INSERT INTO cats(name, age)
VALUES('Charlie', 17);

INSERT INTO cats(name, age)
VALUES('Connie', 10);

SELECT * FROM cats;

source testing/insert.sql
```

CODE: Loading Our Book Data

Section 7, Lecture 100

```
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);


SELECT database();

CREATE DATABASE book_shop;

use book_shop;

show tables;

source book_data.sql

DESC books;

SELECT * FROM books;
```

CODE: Working With CONCAT

Section 7, Lecture 102

```
SELECT author_fname, author_lname FROM books;

CONCAT(x,y,z) // from slides

CONCAT(column, anotherColumn) // from slides

CONCAT(author_fname, author_lname)

CONCAT(column, 'text', anotherColumn, 'more text')

CONCAT(author_fname, ' ', author_lname)

CONCAT(author_fname, author_lname); // invalid syntax

SELECT CONCAT('Hello', 'World');

SELECT CONCAT('Hello', '...', 'World');

SELECT
  CONCAT(author_fname, ' ', author_lname)
FROM books;

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

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

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

SELECT CONCAT(title, '-', author_fname, '-', author_lname) FROM books;

SELECT
    CONCAT_WS(' - ', title, author_fname, author_lname)
FROM books;
```

CODE: Introducing SUBSTRING

Section 7, Lecture 104

```
SELECT SUBSTRING('Hello World', 1, 4);

SELECT SUBSTRING('Hello World', 7);

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

SELECT SUBSTRING('Hello World', 3);

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

SELECT SUBSTRING('Hello World', -7);

SELECT title FROM books;

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

SELECT SUBSTRING(title, 1, 10) FROM books;

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

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

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

source book_code.sql

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

source book_code.sql
```

CODE: Introducing REPLACE

Section 7, Lecture 106

```
SELECT REPLACE('Hello World', 'Hell', '%$#@');

SELECT REPLACE('Hello World', 'l', '7');

SELECT REPLACE('Hello World', 'o', '0');

SELECT REPLACE('HellO World', 'o', '*');

SELECT
  REPLACE('cheese bread coffee milk', ' ', ' and ');

SELECT REPLACE(title, 'e ', '3') FROM books;

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

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

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

*Note: Use `cmd + /` (mac) or `ctrl + /` (pc) to comment out SQL in c9.

CODE: Using REVERSE

Section 7, Lecture 108

```
SELECT REVERSE('Hello World');

SELECT REVERSE('meow meow');

SELECT REVERSE(author_fname) FROM books;

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

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

CODE: Working with CHAR LENGTH

Section 7, Lecture 110

```
SELECT CHAR_LENGTH('Hello World');

SELECT author_lname, CHAR_LENGTH(author_lname) AS 'length' FROM books;

SELECT CONCAT(author_lname, ' is ', CHAR_LENGTH(author_lname), ' characters long') FROM books
;
```

Resource: sql-format.com

CODE: Changing Case with UPPER and LOWER

Section 7, Lecture 112

```
SELECT UPPER('Hello World');

SELECT LOWER('Hello World');

SELECT UPPER(title) FROM books;

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

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

CODE: String Function Challenges Solution

Section 7, Lecture 115

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

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

```
I-like-cats
```

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

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

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

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

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

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

```
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;
```

CODE: Seed Data: Adding A Couple New Books

Section 8, Lecture 118

```
INSERT INTO books
    (title, author_fname, author_lname, released_year, stock_quantity, pages)
    VALUES ('10% Happier', 'Dan', 'Harris', 2014, 29, 256),
           ('fake_book', 'Freida', 'Harris', 2001, 287, 428),
           ('Lincoln In The Bardo', 'George', 'Saunders', 2017, 1000, 367);


SELECT title FROM books;
```

CODE: Using DISTINCT

Section 8, Lecture 120

```
SELECT author_lname FROM books;

SELECT DISTINCT author_lname FROM books;

SELECT author_fname, author_lname FROM books;

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

SELECT DISTINCT author_fname, author_lname FROM books;
```

CODE: Sorting Data with ORDER BY

Section 8, Lecture 122

```
SELECT author_lname FROM books;

SELECT author_lname FROM books ORDER BY author_lname;

SELECT title FROM books;

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

SELECT released_year FROM books;

SELECT released_year FROM books ORDER BY released_year;

SELECT released_year FROM books ORDER BY released_year DESC;

SELECT released_year FROM books ORDER BY released_year ASC;

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

SELECT title, pages FROM books ORDER BY released_year;

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

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

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

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

SELECT author_lname, title
FROM books ORDER BY 2;

SELECT author_fname, author_lname FROM books
ORDER BY author_lname, author_fname;
```

CODE: Using LIMIT

Section 8, Lecture 124

```
SELECT title FROM books LIMIT 3;

SELECT title FROM books LIMIT 1;

SELECT title FROM books LIMIT 10;

SELECT * FROM books LIMIT 1;

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

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

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

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

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

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

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

SELECT * FROM tbl LIMIT 95,18446744073709551615;

SELECT title FROM books LIMIT 5;

SELECT title FROM books LIMIT 5, 123219476457;

SELECT title FROM books LIMIT 5, 50;
```

CODE: Better Searches with LIKE

Section 8, Lecture 126

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

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

SELECT title FROM books WHERE  title LIKE 'the';

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

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

CODE: LIKE Part 2: More Wildcards

Section 8, Lecture 128

```
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 '%\_%'
```

CODE: Refining Selections Exercises Solution

Section 8, Lecture 131

```
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;
```

CODE: The Count Function

Section 9, Lecture 134

```
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%';
```

CODE: The Joys of Group By

Section 9, Lecture 136

```
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 GR
OUP BY released_year;
```

CODE: MIN and MAX Basics

Section 9, Lecture 138

```
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;
```

CODE: A Problem with Min and Max

Section 9, Lecture 140

```
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;
```

CODE: Using Min and Max with Group By

Section 9, Lecture 142

```
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,
         author_fname;
```

CODE: The Sum Function

Section 9, Lecture 144

```
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;
```

CODE: The Avg Function

Section 9, Lecture 146

```
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;
```

CODE: Aggregate Functions Challenges Solution

Section 9, Lecture 149

```
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, autho
r_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;
```

CODE: CHAR and VARCHAR

Section 10, Lecture 153

```
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;
```

CODE: DECIMAL

Section 10, Lecture 155

```
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(34.2989999);

INSERT INTO items(price) VALUES(1.9999);

SELECT * FROM items;
```

CODE: FLOAT and DOUBLE

Section 10, Lecture 157

```
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;
```

CODE: Creating Our DATE data

Section 10, Lecture 160

```
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;
```

CODE: Formatting Dates

Section 10, Lecture 163

```
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;
```

CODE: Date Math

Section 10, Lecture 165

```
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;
```

CODE: Working with TIMESTAMPS

Section 10, Lecture 167

```
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('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;

SELECT * FROM comments2 ORDER BY changed_at;

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

CODE: Data Types Exercises Solution

Section 10, Lecture 170

```
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;
```

CODE: Not Equal

Section 11, Lecture 173

```
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';
```

CODE: Not Like

Section 11, Lecture 175

```
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

Section 11, Lecture 177

```
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';
```

CODE: Less Than

Section 11, Lecture 179

```
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
```

CODE: Logical AND

Section 11, Lecture 181

```
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%';
```

CODE: Logical OR

Section 11, Lecture 183

```
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;
```

CODE: Between

Section 11, Lecture 185

```
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);
```

CODE: In And Not In

Section 11, Lecture 187

```
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;
```

CODE: Case Statements

Section 11, Lecture 189

```
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;
```

CODE: Logical Operators Exercises Solution

Section 11, Lecture 192

```
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' && release
d_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;
```

CODE: Working With Foreign Keys

Section 12, Lecture 198

-- Creating the customers and orders 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)
);
```

-- Inserting some customers and orders

```
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);
```

-- This INSERT fails because of our fk constraint. No user with id: 98

```
INSERT INTO orders (order_date, amount, customer_id)
VALUES ('2016/06/06', 33.67, 98);
```

CODE: Cross Joins

Section 12, Lecture 200

-- Finding Orders Placed By George: 2 Step Process

```
SELECT id FROM customers WHERE last_name='George';
SELECT * FROM orders WHERE customer_id = 1;
```

-- Finding Orders Placed By George: Using a subquery

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

-- Cross Join Craziness

```
SELECT * FROM customers, orders;
```

CODE: Inner Joins

Section 12, Lecture 202

-- IMPLICIT INNER JOIN

```
SELECT * FROM customers, orders
WHERE customers.id = orders.customer_id;
```

-- 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

Section 12, Lecture 204

-- 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;
```

-- 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

Section 12, Lecture 206

-- 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

Section 12, Lecture 208

--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);
```

CODE: Right and Left Joins FAQ

Section 12, Lecture 210

```
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

Section 12, Lecture 213

-- 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

Section 12, Lecture 215

-- 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;
```

CODE: Creating Our Tables

Section 13, Lecture 219

-- 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

Section 13, Lecture 221

-- TV Joins Challenge 1 SOLUTION

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

CODE: TV Joins Challenge 2 SOLUTION

Section 13, Lecture 223

-- 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;
```

CODE: TV Joins Challenge 3 SOLUTION

Section 13, Lecture 225

-- 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;
```

CODE: TV Joins Challenge 4 SOLUTION

Section 13, Lecture 227

-- CHALLENGE 4 - UNREVIEWED SERIES

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

CODE: TV Joins Challenge 5 SOLUTION

Section 13, Lecture 229

-- 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;
```

CODE: TV Joins Challenge 6 SOLUTION

Section 13, Lecture 231

-- 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;
```

CODE: TV Joins Challenge 7 SOLUTION

Section 13, Lecture 233

-- 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;
```

CODE: IG Clone Users Schema

Section 14, Lecture 237

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

CODE: IG Clone Photos Schema

Section 14, Lecture 239

```
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)
);
```

CODE: IG Clone Comments Schema

Section 14, Lecture 241

```
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)
);
```

CODE: IG Clone Likes Schema

Section 14, Lecture 243

```
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)
);
```

CODE: IG Clone Followers Schema

Section 14, Lecture 245

```
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

Section 14, Lecture 248

```
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

Section 14, Lecture 249

```
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)
);
```

Instagram Challenge 1 Solution CODE

Section 15, Lecture 254

-- 1. Finding 5 oldest users

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

Instagram Challenge 2 Solution CODE

Section 15, Lecture 256

-- 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;
```

Instagram Challenge 3 Solution CODE

Section 15, Lecture 258

-- 3. Identify Inactive Users (users with no photos)

```
SELECT username
FROM users
LEFT JOIN photos
    ON users.id = photos.user_id
WHERE photos.id IS NULL;
```

Instagram Challenge 4 Solution CODE

Section 15, Lecture 260

-- 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;
```

Instagram Challenge 5 Solution CODE

Section 15, Lecture 262

-- 5. Calculate average number of photos per user

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

Instagram Challenge 6 Solution CODE

Section 15, Lecture 264

-- 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;
```

Instagram Challenge 7 Solution CODE

Section 15, Lecture 266

-- 7. Finding the bots - the users who have liked every single photo

```
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);
```

CODE: 5 Minute Node Crash Course

Section 16, Lecture 273

//Print "HELLO WORLD" 500 times using Node

```
for(var i = 0; i < 500; i++){
  console.log("HELLO WORLD!");
}
```

// Execute file with:

`node filename.js`

CODE: Introduction to NPM and Faker

Section 16, Lecture 275

Find Faker Docs Here: https://github.com/marak/Faker.js/

STEP 1: Install and Require Faker

// Install Faker via command line:

```
npm install faker
```

// Require it inside of a JS file:

```
var faker = require('faker');
```

STEP 2: Use Faker!

// Print a random email

```
console.log(faker.internet.email());
```

// Print a random past date

```
console.log(faker.date.past());
```

// Print a random city

```
console.log(faker.address.city());
```

// We can define a new function

```
function generateAddress(){
  console.log(faker.address.streetAddress());
  console.log(faker.address.city());
  console.log(faker.address.state());
}
```

// And then execute that function:

```
generateAddress();
```

CODE: Connecting Node to MySQL

Section 16, Lecture 278

Documentation for the MySQL Node Package:

Step 1: Install the MySQL Node Package

`npm install mysql`

Step 2: Connect to Database

```
var mysql = require('mysql');

var connection = mysql.createConnection({
  host     : 'localhost',
  user     : 'learnwithcolt',  //your username
  database : 'join_us'         //the name of your db
});
```

Step 3: Run Queries

Running a super simple SQL query like:

`SELECT 1 + 1;`

Using the MySQL Node Package:

```
connection.query('SELECT 1 + 1 AS solution', function (error, results, fields) {
   if (error) throw error;
   console.log('The solution is: ', results[0].solution);
});
```

Another sample query, this time selecting 3 things:

```
var q = 'SELECT CURTIME() as time, CURDATE() as date, NOW() as now';
connection.query(q, function (error, results, fields) {
  if (error) throw error;
  console.log(results[0].time);
  console.log(results[0].date);
  console.log(results[0].now);
});
```

The equivalent SQL query:

```
SELECT CURTIME() as time, CURDATE() as date, NOW() as now;
```

CODE: Creating Our Users Table

Section 16, Lecture 280

Simple SQL To Create The Users Table

```
CREATE TABLE users (
    email VARCHAR(255) PRIMARY KEY,
    created_at TIMESTAMP DEFAULT NOW()
);
```

CODE: Selecting Using Node

Section 16, Lecture 282

To SELECT all users from database:

```
var q = 'SELECT * FROM users ';
connection.query(q, function (error, results, fields) {
  if (error) throw error;
  console.log(results);
});
```

To count the number of users in the database:

```
var q = 'SELECT COUNT(*) AS total FROM users ';
connection.query(q, function (error, results, fields) {
  if (error) throw error;
  console.log(results[0].total);
});
```

CODE: Inserting Using Node

Section 16, Lecture 284

Inserting Data Using Node

Approach #1

```
var q = 'INSERT INTO users (email) VALUES ("rusty_the_dog@gmail.com")';

connection.query(q, function (error, results, fields) {
  if (error) throw error;
  console.log(results);
});
```

An easier approach that allows for dynamic data

```
var person = {
    email: faker.internet.email(),
    created_at: faker.date.past()
};

var end_result = connection.query('INSERT INTO users SET ?', person, function(err, result) {
  if (err) throw err;
  console.log(result);
});
```

CODE: Bulk Inserting 500 Users

Section 16, Lecture 287

The Code To INSERT 500 Random Users

```
var mysql = require('mysql');
var faker = require('faker');


var connection = mysql.createConnection({
  host     : 'localhost',
  user     : 'learnwithcolt',
  database : 'join_us'
});


var data = [];
for(var i = 0; i < 500; i++){
    data.push([
        faker.internet.email(),
        faker.date.past()
    ]);
}


var q = 'INSERT INTO users (email, created_at) VALUES ?';

connection.query(q, [data], function(err, result) {
  console.log(err);
  console.log(result);
});

connection.end();
```

CODE: 500 Users Exercises Solutions

Section 16, Lecture 290

Solutions To 500 Users Exercises

-- Challenge 1

```
SELECT
    DATE_FORMAT(MIN(created_at), "%M %D %Y") as earliest_date
FROM users;
```

-- Challenge 2

```
SELECT *
FROM   users
WHERE  created_at = (SELECT Min(created_at)
                     FROM   users);
```

-- Challenge 3

```
SELECT Monthname(created_at) AS month,
       Count(*)              AS count
FROM   users
GROUP  BY month
ORDER  BY count DESC;
```

-- Challenge 4

```
SELECT Count(*) AS yahoo_users
FROM   users
WHERE  email LIKE '%@yahoo.com';
```

-- Challenge 5

```
SELECT CASE
         WHEN email LIKE '%@gmail.com' THEN 'gmail'
         WHEN email LIKE '%@yahoo.com' THEN 'yahoo'
         WHEN email LIKE '%@hotmail.com' THEN 'hotmail'
         ELSE 'other'
       end      AS provider,
       Count(*) AS total_users
FROM   users
GROUP  BY provider
ORDER  BY total_users DESC;
```

CODE: Our First Simple Web App

Section 17, Lecture 296

CODE: Our First Simple Web App:

Add to your app.js file:

```
var express = require('express');

var app = express();

app.get("/", function(req, res){
 res.send("HELLO FROM OUR WEB APP!");
});

app.listen(8080, function () {
 console.log('App listening on port 8080!');
});
```

Remember to start the server up:

```
node app.js
```

CODE: Adding Multiple Routes

Section 17, Lecture 298

CODE: Adding Multiple Routes

Add a /joke route:

```
app.get("/joke", function(req, res){
 var joke = "What do you call a dog that does magic tricks? A labracadabrador.";
 res.send(joke);
});
```

Add a /random_num route:

```
app.get("/random_num", function(req, res){
 var num = Math.floor((Math.random() * 10) + 1);
 res.send("Your lucky number is " + num);
});
```

CODE: Connecting Express and MySQL

Section 17, Lecture 300

CODE: Connecting Express and MySQL

Add the MySQL code inside of the root route:

```
app.get("/", function(req, res){
 var q = 'SELECT COUNT(*) as count FROM users';
 connection.query(q, function (error, results) {
 if (error) throw error;
 var msg = "We have " + results[0].count + " users";
 res.send(msg);
 });
});
```

CODE: Adding EJS Templates

Section 17, Lecture 302

CODE: Adding EJS Templates

```
<h1>JOIN US</h1>

<p class="lead">Enter your email to join <strong>518</strong>
others on our waitlist. We are 100% not a cult. </p>

<form method="POST" action='/register'>
 <input type="text" class="form" placeholder="Enter Your Email">
 <button>Join Now</button>
</form>
```

CODE: Connecting the Form

Section 17, Lecture 304

CODE: Connecting the Form

The '/register' post route:

```
app.post('/register', function(req,res){
 var person = {email: req.body.email};
 connection.query('INSERT INTO users SET ?', person, function(err, result) {
 console.log(err);
 console.log(result);
 res.redirect("/");
 });
});
```

CODE: HTML AND CSS FILES

Section 17, Lecture 306

JOIN US: HTML AND CSS FILES

Download the attached zip file containing the completed HTML and CSS.

Resources for this lecture

JOIN_US_HTML_AND_CSS.zip