CODE: Creating Databases

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

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

```
USE <database name>;

-- example:
USE dog_walking_app;

SELECT database();
```

CODE: Creating Your Own Tables

```
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?

```
SHOW TABLES;

SHOW COLUMNS FROM tablename;

DESC tablename;
```

CODE: Dropping Tables

Dropping Tables

`DROP TABLE <tablename>;`

A specific example:

`DROP TABLE cats;`

Be careful with this command!

CODE: Creating Your Own Tables Challenge

```
CREATE TABLE pastries
  (
    name VARCHAR(50),
    quantity INT
  );

SHOW TABLES;

DESC pastries;

DROP TABLE pastries;
```

CODE: Inserting Data

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

```
SELECT * FROM cats;
```

CODE: Multiple Insert

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

Note about using quotes inside of inserted values

Hey everyone!
If you're wondering how to insert a string (VARCHAR) value that contains quotations, then here's how.

You can do it a couple of ways:

- Escape the quotes with a backslash: `"This text has \"quotes\" in it"` or `'This text has \'quotes\' in it'`
- Alternate single and double quotes: `"This text has 'quotes' in it"` or `'This text has "quotes" in it'`

CODE: INSERT Challenges Solution

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;`

NOTE: MySQL Warnings

Hello everyone,
In the next lecture Colt will introduce warnings in MySQL.

If you happen to encounter an error instead of a warning then please see here for a discussion in the Q&A that covers what's happening there.

The solution is to run the following command in your mysql shell: `set sql_mode='';`

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
```

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

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);
```

CODE: Setting Default Values

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

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: Introduction to CRUD

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

CODE: Preparing Our Data

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

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

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

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

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

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';
```

Solution

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

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

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

```
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

```
1. SELECT article, color FROM shirts;
2.
3. SELECT * FROM shirts WHERE shirt_size='M';
4.
5. SELECT article, color, shirt_size, last_worn FROM shirts WHERE shirt_size='M';
```

CODE: CRUD Exercise Update Solution

```sql
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='

SELECT * FROM shirts WHERE color='white';

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

SELECT * FROM shirts;
```

CODE: CRUD Exercise Delete Solution

```sql
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

```
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

1. First create your book_data.sql file with the following code:

```
DROP DATABASE IF EXISTS book_shop;
CREATE DATABASE book_shop;
USE book_shop;

CREATE TABLE books
        (
                book_id INT NOT NULL AUTO_INCREMENT,
                title VARCHAR(100),
                author_fname VARCHAR(100),
                author_lname VARCHAR(100),
                released_year INT,
                stock_quantity INT,
                pages INT,
                PRIMARY KEY(book_id)
        );
```

```
INSERT INTO books (title, author_fname, author_lname, released_yea
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,
('The Circle', 'Dave', 'Eggers', 2013, 26, 504),
('The Amazing Adventures of Kavalier & Clay', 'Michael', 'Chabon',
('Just Kids', 'Patti', 'Smith', 2010, 55, 304),
('A Heartbreaking Work of Staggering Genius', 'Dave', 'Eggers', 20
('Coraline', 'Neil', 'Gaiman', 2003, 100, 208),
('What We Talk About When We Talk About Love: Stories', 'Raymond',
("Where I'm Calling From: Selected Stories", 'Raymond', 'Carver',
('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)
```

2. Then source that file

`source book_data.sql`

3. Now check your work:

```
DESC books;
SELECT * FROM books;
```

CODE: Working With CONCAT

```sql
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;
```

```sql

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

```sql
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;

```

```sql
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

```sql
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;

```

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

Notes:

- Use `cmd + /` (mac) or `ctrl + /` (pc) to comment out SQL in c9.
- The REPLACE() function, as well as the other string functions, only change the query output, they don't affect the actual data in the database.

CODE: Using REVERSE

```
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

```
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 boo
ks;
```

Resource: sql-format.com

CODE: Changing Case with UPPER and LOWER

```
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;
```

Note about string functions

Hi everyone,

I have two important notes for you.

Firstly, before you move onto the next lecture, please remember that order is important when dealing with combining/wrapping certain string functions.

For example:

This works:

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

While this does not:

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

UPPER only takes one argument and CONCAT takes two or more arguments, so they can't be switched in that way.

You could do it this way, however:

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

But, the first example above would be better (more DRY*) because you wouldn't need to call UPPER two times.

*Don't Repeat Yourself

And secondly, the last exercise in the lecture that follows will show only 2 rows in the table from the slide where Colt describes the exercise. However, in the solution video Colt shows more than 2 rows/results in the query's output. Please feel free to return all the rows from the table (to match Colt's solution), then if you want an extra challenge you can try to modify your query to return just the two rows from the slide deck example.

CODE: String Function Challenges Solution

```
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

```
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

```
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

```sql
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_ye

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

```sql
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

```
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

```
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

```sql
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;

```

```sql

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

CODE: The Count Function

```sql
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

```
SELECT title, author_lname FROM books;

SELECT title, author_lname FROM books
GROUP BY author_lname;

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


SELECT title, author_fname, author_lname FROM books;

SELECT title, author_fname, author_lname FROM books GROUP BY author_lname;

SELECT author_fname, author_lname, COUNT(*) FROM books GROUP BY author_lname;

SELECT author_fname, author_lname, COUNT(*) FROM books GROUP BY author_lname, author_fname;

SELECT released_year FROM books;

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

SELECT CONCAT('In ', released_year, ' ', COUNT(*), ' book(s) released') AS year FROM books
GROUP BY released_year;
```

CODE: MIN and MAX Basics

```
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

```
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

```
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

```
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

```sql
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

```sql
FROM books;

SELECT COUNT(*) FROM books GROUP BY released_year;

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

SELECT Sum(stock_quantity) FROM BOOKS;

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

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

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

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

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

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

CODE: CHAR and VARCHAR

```
1.  CREATE TABLE dogs (name CHAR(5), breed VARCHAR(10));
2.
3.  INSERT INTO dogs (name, breed) VALUES ('bob', 'beagle');
4.
5.  INSERT INTO dogs (name, breed) VALUES ('robby', 'corgi');
6.
7.  INSERT INTO dogs (name, breed) VALUES ('Princess Jane', 'Retriever');
8.
9.  SELECT * FROM dogs;
10.
11. INSERT INTO dogs (name, breed) VALUES ('Princess Jane', 'Retrievesadfdsafdasfsafr');
12.
13. SELECT * FROM dogs;
```

CODE: DECIMAL

```
1   CREATE TABLE items(price DECIMAL(5,2));
2
3   INSERT INTO items(price) VALUES(7);
4
5   INSERT INTO items(price) VALUES(7987654);
6
7   INSERT INTO items(price) VALUES(34.88);
8
9   INSERT INTO items(price) VALUES(298.9999);
10
11  INSERT INTO items(price) VALUES(1.9999);
12
13  SELECT * FROM items;
```

CODE: FLOAT and DOUBLE

```
CREATE TABLE thingies (price FLOAT);

INSERT INTO thingies(price) VALUES (88.45);

SELECT * FROM thingies;

INSERT INTO thingies(price) VALUES (8877.45);

SELECT * FROM thingies;

INSERT INTO thingies(price) VALUES (8877665544.45);

SELECT * FROM thingies;
```

CODE: Creating Our DATE data

```
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;
```

Note about formatting dates

Hello Everyone!

In the following lecture, titled "Formatting Dates", there's a small typo around the 13 minute and 50 second mark.

It should say %i for minute, instead of %m, the correction can also be seen in the CODE lecture.

CODE: Formatting Dates

```
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

```
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

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

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

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

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

SELECT * FROM comments ORDER BY created_at DESC;

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

```
);

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

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

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

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

SELECT * FROM comments2;

SELECT * FROM comments2 ORDER BY changed_at;

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

CODE: Data Types Exercises Solution

```
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

```
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

```
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

1. SELECT title, released_year FROM books ORDER BY released_year;
2.

```
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

```
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
```

```
17.
18. SELECT 'h' < 'p';
19. -- true
20.
21. SELECT 'Q' <= 'q';
22. -- true
```

CODE: Logical AND

```sql
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
```

```
22
23  SELECT 54 <= 54 && 'a' = 'A';
24  -- true
25
26  SELECT *
27  FROM books
28  WHERE author_lname='Eggers'
29      AND released_year > 2010
30      AND title LIKE '%novel%';
```

Please note, as of MySQL 8.0.17, the `&&` operator is deprecated and support for it will be removed in a future MySQL version. Applications should be adjusted to use the standard SQL `AND` operator.
If you're using MySQL 5.7 or older, which most of you are if you're using GoormIDE, then you don't have to worry about this right now. But, in newer versions of MySQL (8.0.17 and newer) you will need to replace `&&` with `AND`.
- source

CODE: Logical OR

```
1   SELECT
2       title,
3       author_lname,
4       released_year
5   FROM books
6   WHERE author_lname='Eggers' || released_year > 2010;
7
8
9   SELECT 40 <= 100 || -2 > 0;
10  -- true
11
12  SELECT 10 > 5 || 5 = 5;
13  -- true
14
15  SELECT 'a' = 5 || 3000 > 2000;
16  -- true
17
```

```
SELECT title,
       author_lname,
       released_year,
       stock_quantity
FROM   books
WHERE  author_lname = 'Eggers'
              || released_year > 2010
OR     stock_quantity > 100;
```

Please note, as of MySQL 8.0.17, the `||` operator is deprecated and support for it will be removed in a future MySQL version. Applications should be adjusted to use the standard SQL `OR` operator.
If you're using MySQL 5.7 or older, which most of you are if you're using GoormIDE, then you don't have to worry about this right now. But, in newer versions of MySQL (8.0.17 and newer) you will need to replace `||` with `OR`.
- source

CODE: Between

```
SELECT title, released_year FROM books WHERE released_year >= 2004

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

SELECT
```

```
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

```sql
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);

```

```sql
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

```
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

1. `SELECT 10 != 10;`

```
-- false

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

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

SELECT title, released_year FROM books WHERE released_year < 1980;

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

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

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

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

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

```

CODE: Working With Foreign Keys

-- 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

-- 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

Note: please see here for an animated visual of how inner joins work.

-- 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

-- Getting Fancier (Inner Joins Still)

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

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

Note: please see here for an animated visual of how left/right joins work.

-- LEFT JOINS

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

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

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

CODE: Right Joins Part 1

Note: please see here for an animated visual of how left/right joins work.

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

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

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

```
CREATE TABLE customers(
    id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    email VARCHAR(100)
);
CREATE TABLE orders(
    id INT AUTO_INCREMENT PRIMARY KEY,
    order_date DATE,
    amount DECIMAL(8,2),
    customer_id INT
);
```

-- INSERTING NEW DATA (no longer bound by foreign key constraint)

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

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

INSERT INTO orders (order_date, amount, customer_id) VALUES
('2017/11/05', 23.45, 45),
(CURDATE(), 777.77, 109);
```

CODE: Right Joins Part 2

Note: please see here for an animated visual of how left/right joins work.

--A MORE COMPLEX RIGHT JOIN

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

-- WORKING WITH ON DELETE CASCADE

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

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

```
18
19  INSERT INTO customers (first_name, last_name, email)
20  VALUES ('Boy', 'George', 'george@gmail.com'),
21         ('George', 'Michael', 'gm@gmail.com'),
22         ('David', 'Bowie', 'david@gmail.com'),
23         ('Blue', 'Steele', 'blue@gmail.com'),
24         ('Bette', 'Davis', 'bette@aol.com');
25
26  INSERT INTO orders (order_date, amount, customer_id)
27  VALUES ('2016/02/10', 99.99, 1),
28         ('2017/11/11', 35.50, 1),
29         ('2014/12/12', 800.67, 2),
30         ('2015/01/03', 12.50, 2),
31         ('1999/04/11', 450.25, 5);
```

CODE: Right and Left Joins FAQ

Note: please see here for an animated visual of how left/right joins work.

```
1  SELECT * FROM customers
2  LEFT JOIN orders
3      ON customers.id = orders.customer_id;
```

```
1  SELECT * FROM orders
2  RIGHT JOIN customers
3      ON customers.id = orders.customer_id;
```

```
1  SELECT * FROM orders
2  LEFT JOIN customers
3      ON customers.id = orders.customer_id;
```

```
1  SELECT * FROM customers
2  RIGHT JOIN orders
3      ON customers.id = orders.customer_id;
```

CODE: Our First Joins Exercise

-- The Schema

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


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

-- The Starter Data

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

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

CODE: Our First Joins Exercise SOLUTION PT. 2

-- EXERCISE 1

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

-- ALT SOLUTION

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

-- PROBLEM 2

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

-- PROBLEM 3

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

-- PROBLEM 4

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

-- PROBLEM 5

```
SELECT first_name,
       Ifnull(Avg(grade), 0) AS average,
       CASE
         WHEN Avg(grade) IS NULL THEN 'FAILING'
         WHEN Avg(grade) >= 75 THEN 'PASSING'
         ELSE 'FAILING'
       end                      AS passing_status
FROM   students
       LEFT JOIN papers
              ON students.id = papers.student_id
GROUP  BY students.id
ORDER  BY average DESC;
```

CODE: Creating Our Tables

-- CREATING THE REVIEWERS TABLE

```
CREATE TABLE reviewers (
    id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100),
    last_name VARCHAR(100)
);
```

-- CREATING THE SERIES TABLE

```
CREATE TABLE series(
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(100),
    released_year YEAR(4),
    genre VARCHAR(100)
);
```

-- CREATING THE REVIEWS TABLE

```
CREATE TABLE reviews (
    id INT AUTO_INCREMENT PRIMARY KEY,
    rating DECIMAL(2,1),
    series_id INT,
    reviewer_id INT,
    FOREIGN KEY(series_id) REFERENCES series(id),
    FOREIGN KEY(reviewer_id) REFERENCES reviewers(id)
);
```

-- INSERTING A BUNCH OF DATA

```
INSERT INTO series (title, released_year, genre) VALUES
    ('Archer', 2009, 'Animation'),
    ('Arrested Development', 2003, 'Comedy'),
    ("Bob's Burgers", 2011, 'Animation'),
    ('Bojack Horseman', 2014, 'Animation'),
    ("Breaking Bad", 2008, 'Drama'),
    ('Curb Your Enthusiasm', 2000, 'Comedy'),
    ("Fargo", 2014, 'Drama'),
    ('Freaks and Geeks', 1999, 'Comedy'),
    ('General Hospital', 1963, 'Drama'),
    ('Halt and Catch Fire', 2014, 'Drama'),
    ('Malcolm In The Middle', 2000, 'Comedy'),
    ('Pushing Daisies', 2007, 'Comedy'),
    ('Seinfeld', 1989, 'Comedy'),
    ('Stranger Things', 2016, 'Drama');

```

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


INSERT INTO reviews(series_id, reviewer_id, rating) VALUES
    (1,1,8.0),(1,2,7.5),(1,3,8.5),(1,4,7.7),(1,5,8.9),
    (2,1,8.1),(2,4,6.0),(2,3,8.0),(2,6,8.4),(2,5,9.9),
    (3,1,7.0),(3,6,7.5),(3,4,8.0),(3,3,7.1),(3,5,8.0),
    (4,1,7.5),(4,3,7.8),(4,4,8.3),(4,2,7.6),(4,5,8.5),
    (5,1,9.5),(5,3,9.0),(5,4,9.1),(5,2,9.3),(5,5,9.9),
    (6,2,6.5),(6,3,7.8),(6,4,8.8),(6,2,8.4),(6,5,9.1),
    (7,2,9.1),(7,5,9.7),
    (8,4,8.5),(8,2,7.8),(8,6,8.8),(8,5,9.3),
    (9,2,5.5),(9,3,6.8),(9,4,5.8),(9,6,4.3),(9,5,4.5),
    (10,5,9.9),
    (13,3,8.0),(13,4,7.2),
    (14,2,8.5),(14,3,8.9),(14,4,8.9);
```

CODE: TV Joins Challenge 1 Solution

-- TV Joins Challenge 1 SOLUTION

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

CODE: TV Joins Challenge 2 SOLUTION

```
-- Challenge 2 AVG rating
```

```sql
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

```
-- CHALLENGE 3  - Two Solutions
```

```sql
SELECT
    first_name,
    last_name,
    rating
FROM reviewers
INNER JOIN reviews
    ON reviewers.id = reviews.reviewer_id;
```

```sql
SELECT
    first_name,
    last_name,
    rating
FROM reviews
INNER JOIN reviewers
    ON reviewers.id = reviews.reviewer_id;
```

CODE: TV Joins Challenge 4 SOLUTION

-- 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

-- 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

-- 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

-- CHALLENGE 7 - 3 TABLES!

```sql
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;
```