**10-SQL Summary**

?s

How do I activate auto-complete?

How does 2.6 illustrate GUI?

Remind me what we linked with Heroku?

* 1. Created Local Host

1.2

CREATE DATABASE animals\_db;

Note the semicolon at the end of the statement.

This character tells MySQL that the line of code is complete. This is an important facet of SQL syntax:

forgetting the semicolon will result in errors and non-functional code.

1.3 Create table

What is the best way to see the results? SELECT \* FROM people OR grid in left pane

USE animals\_db;

CREATE TABLE people (name VARCHAR(30) NOT NULL,

has\_pet BOOLEAN DEFAULT false,

pet\_name VARCHAR(30),

pet\_age INT);

id INTEGER(11) AUTO\_INCREMENT NOT NULL,

PRIMARY KEY (id)

SELECT \* FROM people;

HOW DO I MAKE THE CODE SLIGHTLY LARGER? Preferences…font

1.4 Favorite DB

-- Drops the favorite\_db if it exists currently --

DROP DATABASE IF EXISTS favorite\_db;

-- Creates the "favorite\_db" database --

CREATE DATABASE favorite\_db;

-- Makes it so all of the following code will affect favorite\_db --

USE favorite\_db;

-- Creates the table "favorite\_foods" within favorite\_db --

CREATE TABLE favorite\_foods (

-- Makes a string column called "food" which cannot contain null --

food VARCHAR(50) NOT NULL,

-- Makes a numeric column called "score" --

score INT);

CREATE TABLE favorite\_songs (

song VARCHAR(100) NOT NULL,

artist VARCHAR(50),

score INT);

CREATE TABLE favorite\_movies (

movie VARCHAR(100) NOT NULL,

-- Creates a boolean column called "five\_times" that sets the default value to false if nothing is entered --

five\_times BOOLEAN DEFAULT false,

score INT);

1.5 Adding Rows

USE animals\_db;

INSERT INTO people (name, has\_pet, pet\_name, pet\_age)

VALUES ("Jacob", true, "Misty", 10);

INSERT INTO people (name, has\_pet, pet\_name, pet\_age)

VALUES ("Ahmed", true, "Rockington", 100);

INSERT INTO people (name)

VALUES ("Peter");

UPDATE people

SET has\_pet = true

WHERE name = "Peter";

SELECT \* FROM people;

1.6 Favorite DB data

USE favorite\_db;

INSERT INTO favorite\_foods (food, score)

VALUES ("Spaghetti", 8);

INSERT INTO favorite\_foods (food, score)

VALUES ("Pizza", 10);

INSERT INTO favorite\_foods (food, score)

VALUES ("Tuna Casserole", 2);

SELECT \* FROM favorite\_foods;

---

INSERT INTO favorite\_songs (song, artist, score)

VALUES ("Papercut", "Zedd", 8);

INSERT INTO favorite\_songs (song, artist, score)

VALUES ("Pinball Wizard", "The Who", 7);

INSERT INTO favorite\_songs (song, artist, score)

VALUES ("Sad Machine", "Porter Robinson", 10);

SELECT \* FROM favorite\_songs;

---

INSERT INTO favorite\_movies (movie, five\_times, score)

VALUES ("Citizen Kane", true, 9),("Bladerunner", true, 10), ("Pup Star", false, 2);

SELECT \* FROM favorite\_movies;

1.7 Value of uniques

-- Drops the animals\_db if it exists currently --

DROP DATABASE IF EXISTS animals\_db;

-- Creates the "animals\_db" database --

CREATE DATABASE animals\_db;

-- Makes it so all of the following code will affect animals\_db --

USE animals\_db;

-- Creates the table "people" within animals\_db --

CREATE TABLE people (

-- Creates a numeric column called "id" which will automatically increment its default value as we create new rows --

id INT AUTO\_INCREMENT NOT NULL,

-- Makes a string column called "name" which cannot contain null --

name VARCHAR(30) NOT NULL,

-- Makes a boolean column called "has\_pet" with a default value of false --

has\_pet BOOLEAN DEFAULT false,

-- Makes a sting column called "pet\_name" --

pet\_name VARCHAR(30),

-- Makes an numeric column called "pet\_age" --

pet\_age INT,

-- Sets id as this table's primary key which means all data contained within it will be unique --

PRIMARY KEY (id));

-- Creates new rows containing data in all named columns --

INSERT INTO people (name, has\_pet, pet\_name, pet\_age)

VALUES ("Ahmed", true, "Rockington", 100);

INSERT INTO people (name, has\_pet, pet\_name, pet\_age)

VALUES ("Ahmed", true, "Rockington", 100);

INSERT INTO people (name, has\_pet, pet\_name, pet\_age)

VALUES ("Jacob",true,"Misty",10);

INSERT INTO people (name)

VALUES ("Peter");

-- Updates the row where the column name is peter --

UPDATE people

SET has\_pet = true, pet\_name = "Franklin", pet\_age = 2

WHERE id = 4;

SELECT \* FROM people;

1.8 Programming DB

-- Drops the programming\_db if it already exists --

DROP DATABASE IF EXISTS programming\_db;

-- Create a database called programming\_db --

CREATE DATABASE programming\_db;

USE programming\_db;

CREATE TABLE programming\_languages(

-- Creates a numeric column called "id" which will automatically increment its default value as we create new rows. --

id INT AUTO\_INCREMENT NOT NULL,

language VARCHAR(20),

rating INT,

-- Creates a boolean column called "mastered" which will automatically fill --

-- with true when a new row is made and the value isn't otherwise defined. --

mastered BOOLEAN DEFAULT true,

PRIMARY KEY (id));

-- Creates new rows --

INSERT INTO programming\_languages (language, rating)

VALUES ("HTML", 95);

INSERT INTO programming\_languages (language, rating)

VALUES ("JS", 99);

INSERT INTO programming\_languages (language, rating)

VALUES ("JQuery", 98);

INSERT INTO programming\_languages (language, rating)

VALUES ("MySQL", 70);

1.9 Books DB

DROP DATABASE IF EXISTS books\_db;

CREATE DATABASE books\_db;

USE books\_db;

CREATE TABLE books(

id INT AUTO\_INCREMENT NOT NULL,

authorId INT,

title VARCHAR(100),

PRIMARY KEY (id));

CREATE TABLE authors(

id INT AUTO\_INCREMENT NOT NULL,

firstName VARCHAR(100),

lastName VARCHAR(100),

PRIMARY KEY (id));

INSERT INTO authors (firstName, lastName) values ('Jane', 'Austen');

INSERT INTO authors (firstName, lastName) values ('Mark', 'Twain');

INSERT INTO authors (firstName, lastName) values ('Lewis', 'Carroll');

INSERT INTO authors (firstName, lastName) values ('Andre', 'Asselin');

INSERT INTO books (title, authorId) values ('Pride and Prejudice', 1);

INSERT INTO books (title, authorId) values ('Emma', 1);

INSERT INTO books (title, authorId) values ('The Adventures of Tom Sawyer', 2);

INSERT INTO books (title, authorId) values ('Adventures of Huckleberry Finn', 2);

INSERT INTO books (title, authorId) values ('Alice''s Adventures in Wonderland', 3);

INSERT INTO books (title, authorId) values ('Dracula', null);

SELECT \* FROM authors;

SELECT \* FROM books;

-- show ALL books with authors

-- INNER JOIN will only return all matching values from both tables

SELECT title, firstName, lastName

FROM books

INNER JOIN authors ON books.authorId = authors.id;

-- show ALL books, even if we don't know the author

-- LEFT JOIN returns all of the values from the left table, and the matching ones from the right table

SELECT title, firstName, lastName

FROM books

LEFT JOIN authors ON books.authorId = authors.id;

-- show ALL books, even if we don't know the author

-- RIGHT JOIN returns all of the values from the right table, and the matching ones from the left table

SELECT title, firstName, lastName

FROM books

RIGHT JOIN authors ON books.authorId = authors.id;

2.1 Warmup

DROP DATABASE IF EXISTS Second\_International\_Bank;

CREATE DATABASE Second\_International\_Bank;

USE Second\_International\_Bank;

CREATE TABLE Customers (

    ID int(50) AUTO\_INCREMENT,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Loan BOOLEAN,

Checking DECIMAL(20,2),

Savings DECIMAL(20,2),

primary KEY(ID));

INSERT INTO Customers(FirstName,LastName,Loan,Checking,Savings)

VALUES ("Richard", "Rich", TRUE, 1000.00, 20000.50),

("Bob", "Someone", TRUE, 10.75, 2.05),

("Shelly", "RichRich", FALSE, 1000000.00, 50000025.00),

("Ryan", "Middleman", FALSE, 250.00, 10000.00),

("Ryan", "Middleman", FALSE, 250.00, 10000.00),

("Shannon", "Waffles", TRUE, 1000.00, 20000.50);

SELECT \* FROM Customers;

2.2 Import wizard

**used the workbench import wizard**

right click on the db in the left pane…

DROP DATABASE IF EXISTS Miscellaneous\_DB;

-- Create Database to import csv into

CREATE DATABASE Miscellaneous\_DB;

USE Miscellaneous\_DB;

-- Check data was uploaded

SELECT \* FROM birdsong;

2.3 Intro to Queries

-- Select everything...TABLE IS ALREADY CREATED

SELECT \* FROM birdsong;

-- One condition

SELECT \* FROM birdsong WHERE genus = "Acanthis";

-- Two conditions with "and"

SELECT \* FROM birdsong WHERE genus = "Acanthis" AND country = "Netherlands";

-- Either condition with "or"

SELECT \* FROM birdsong WHERE genus = "Acanthis" OR genus = "Acrocephalus";

-- Where condition is not met

SELECT \* FROM birdsong WHERE NOT genus = "Anthus";

2.4 Hide & seek

-- Alter imported table to add an id

ALTER TABLE wordassociation

ADD COLUMN id INT AUTO\_INCREMENT PRIMARY KEY FIRST;

-- USED "table data import wizard" to import multiple csv's to 1 table

-- Select all with the source of AC

SELECT \* FROM wordassociation

WHERE source = "AC";

-- Select all with the source of BC

SELECT \* FROM wordassociation

WHERE source = "BC";

-- Select all with the source of CC

SELECT \* FROM wordassociation

WHERE source = "CC";

-- Select all where the author is greater than 0 and less than 20

SELECT \* FROM wordassociation

WHERE author >= 0 AND author <= 20;

-- Select all where either word is pie

SELECT \* FROM wordassociation

WHERE word1 = "pie" OR word2 = "pie";

-- Find the lowest id whose source is CC

SELECT MIN(id) FROM wordassociation

WHERE source = "AC";

-- Find how many rows have an author of 12

SELECT COUNT(author) FROM wordassociation

WHERE author = 12;

There are some functions in MySQL that allow users to perform simple calculations like MIN(), MAX(), COUNT(), AVG(), and SUM().

2.5 SCUD

-- Add primary key

ALTER TABLE globalfirepower

ADD COLUMN id INT AUTO\_INCREMENT PRIMARY KEY FIRST;

-- Turn off safe updates

SET SQL\_SAFE\_UPDATES = 0;

-- Delete and update data

DELETE FROM globalfirepower

WHERE ReservePersonnel = 0;

UPDATE globalfirepower

SET FighterAircraft = 1

WHERE FighterAircraft = 0;

UPDATE globalfirepower

SET TotalAircraftStrength = TotalAircraftStrength + 1

WHERE FighterAircraft = 1;

-- Turn safe updates on

SET SQL\_SAFE\_UPDATES = 1;

-- Select Averages

SELECT AVG(TotalMilitaryPersonnel),

    AVG(TotalAircraftStrength),

    AVG(TotalHelicopterStrength),

    AVG(TotalPopulation)

FROM globalfirepower;

-- Insert new data

INSERT INTO globalfirepower(Country, TotalPopulation, TotalMilitaryPersonnel, TotalAircraftStrength, TotalHelicopterStrength)

VALUES ("GlobalLand", 60069024, 524358, 457, 183);

SELECT \* FROM globalfirepower;

2.6 Workbench GUI (graphical user interface)

-- Alter imported table to add an id

ALTER TABLE comcastfcccomplaints

ADD COLUMN id INT AUTO\_INCREMENT PRIMARY KEY FIRST;

-- Select all from the table

SELECT \* FROM comcastfcccomplaints;

2.7 Joins

SELECT players.first\_name, players.last\_name, players.hand, matches.loser\_rank

FROM matches

INNER JOIN players ON

players.player\_id=matches.loser\_id;

<http://www.sql-join.com/sql-join-types>

<http://www.codeproject.com/KB/database/Visual_SQL_Joins/Visual_SQL_JOINS_orig.jpg>

2.8 Joining Team

CREATE DATABASE NbaJoin;

-- right click-table data import wizard for 2 tables

USE NbaJoin;

-- Join players with seasons\_stats

SELECT players.player, players.height, players.weight, players.college, players.born, seasons\_stats.pos, seasons\_stats.tm

FROM players

INNER JOIN seasons\_stats ON

players.player = seasons\_stats.player;

-- Join seasons\_stats with players

SELECT seasons\_stats.player, players.college, seasons\_stats.year, seasons\_stats.pos, seasons\_stats.`2P%`,

seasons\_stats.`FG%`, seasons\_stats.`FT%`, seasons\_stats.`TS%`

FROM seasons\_stats

INNER JOIN players ON

players.player = seasons\_stats.player;

2.9 SQL Witchcraft…SKIP? Lots of csv files

SELECT c.caseref, d.devil\_text, d.devil\_type, a.accusedref

FROM wdb\_accused AS a

INNER JOIN wdb\_case AS c ON a.accusedref = c.accusedref

INNER JOIN wdb\_devilappearance AS d ON d.caseref = c.caseref

WHERE d.devil\_text > ' ';

Class 3 Objectives

* solidify foundations of writing basic- to intermediate-level MySQL statements.
* understand how to perform subqueries, & difference between inner and outer joins.
* develop an introductory understanding of table design and database management.

3.1 Warmup - Joins

SELECT address, city

FROM address a

JOIN city c

ON (a.city\_id = c.city\_id);

* a & c shorten table names

SELECT address, city

FROM address

JOIN city

USING (city\_id);

* USING works if we are matching on same column names
* JOIN and INNER JOIN are synonymous.

3.2 GregariousAggregates

<https://dev.mysql.com/doc/refman/5.7/en/group-by-functions.html>

 to get a bird's-eye view of the table, we can use SELECT \* FROM film;, as we have done, or  DESCRIBE film;, which will list columns, as well as their types, default values, whether NULL values are allowed, and other details:

How many films exist in our database ?

SELECT COUNT(film\_id) AS 'Total films' ( "aliasing" in SQL, create alias name for column.)

FROM film;

break down the number of movies by its rating? How many G-rated movies are there?

SELECT rating, COUNT(film\_id) AS 'Total films' FROM film GROUP BY rating;

-- 1. What is the average cost to rent a film in the Sakila stores?

SELECT AVG(rental\_rate) AS 'Average rental rate'

FROM film;

-- 2. What is the average rental cost of films by rating?

-- On average, what is the cheapest rating of films to rent? Most expensive?

SELECT rating, AVG(rental\_rate) AS 'Average rental rate'

FROM film

GROUP BY rating;

-- 3. How much would it cost to replace all the films in the database?

SELECT SUM(replacement\_cost) AS 'Total replacement cost'

FROM film;

-- 4. How much would it cost to replace all the films in each ratings category?

SELECT rating, SUM(replacement\_cost) AS 'Replacement cost'

FROM film

GROUP BY rating;

-- 5. How long is the longest movie in the database? The shortest?

SELECT MAX(length)

FROM film;

-- 6. For customers with id numbers 1 through 4, display the total amount they have paid.

SELECT customer\_id, SUM(amount)

FROM payment

GROUP BY customer\_id

HAVING customer\_id < 5;

for aggregate functions, we use the keyword HAVING instead of WHERE

3.3 Sub queries

-- list the names and ID numbers of cities that are in the following list:

-- 'Qalyub', 'Qinhuangdao', 'Qomsheh', 'Quilmes'

SELECT city, city\_id

FROM city

WHERE city IN ('Qalyub', 'Qinhuangdao', 'Qomsheh', 'Quilmes');

-- Display the districts in the above list of cities:

SELECT district

FROM address

WHERE city\_id IN

(

SELECT city\_id

FROM city

WHERE city IN ('Qalyub', 'Qinhuangdao', 'Qomsheh', 'Quilmes'));

-- Using subqueries, find the first and last names of customers who reside in cities that begin with the letter \*\*Q\*\*.

SELECT first\_name, last\_name

FROM customer cus

WHERE address\_id IN

(

SELECT address\_id

FROM address a

WHERE city\_id IN

(

SELECT city\_id

FROM city

WHERE city LIKE 'Q%'));

* We identify the use of a **wildcard** with the keyword LIKE
* **%** tells us that our query can substitute zero, one, or multiple characters in its place.

 the single character wildcard: \_

With the **underscore**, one, and only one, character must take its place in the query.

SELECT \* FROM actor WHERE first\_name LIKE '**\_**AN'; (Dan…Ian)

In HW, options…:

-- 6d. How many copies of the film “Hunchback Impossible” exist in the inventory system? 6

-- get it from film\_id = 439?

**In 2 steps:**

SELECT title, film\_id FROM film WHERE title = 'Hunchback Impossible';

SELECT \* FROM inventory WHERE film\_id = "439";

**With a join:**

SELECT i.inventory\_id, i.film\_id, i.store\_id

FROM inventory i

JOIN film f

ON (i.film\_id = f.film\_id)

WHERE f.title = 'Hunchback Impossible';

**Sub-query:**

SELECT \* FROM inventory

WHERE film\_id IN

(SELECT film\_id

FROM film

WHERE title = 'Hunchback Impossible');

SELECT title, (SELECT COUNT(\*) FROM inventory WHERE film.film\_id = inventory.film\_id ) AS 'Number of Copies'

FROM film;

-- these 3 above do the same thing; **below = direct answer**.

SELECT film.title, COUNT(inventory.film\_id) AS "Number of Copies"

FROM inventory

INNER JOIN film ON

inventory.film\_id=film.film\_id

WHERE film.title = "Hunchback Impossible"

GROUP BY film.film\_id;

<https://sqlbolt.com/lesson/select_queries_order_of_execution>

3.4 View\_Room\_Queries

Write a MySQL statement to **sub-query** a film title, and the numbers of copies that exist in the inventory for that title.

SELECT title, (SELECT COUNT(\*) FROM inventory WHERE film.film\_id = inventory.film\_id ) AS 'Number of Copies' FROM film;

Database…reverse engineer…to get **ERD** = entity relationship diagram

to generate a diagram of the schema in MySQL Workbench

<https://stackoverflow.com/questions/2488/auto-generate-database-diagram-mysql>

The schema makes it easier to identify which tables we need, and which keys we will use to link our subqueries. The schema is an SVG file and can be opened with a web browser

3.5 Revisit Sub-queries

Q: How many people have rented the film *\_Blanket Beverly\_*?

SELECT COUNT(\*)

FROM customer

WHERE customer\_id IN

(SELECT customer\_id

FROM payment

WHERE rental\_id IN

(SELECT rental\_id

FROM rental

WHERE inventory\_id IN

(SELECT inventory\_id

FROM inventory

WHERE film\_id IN

(SELECT film\_id

FROM film

WHERE title = 'Blanket Beverly'

) ) ) );

3.6 Mine\_the\_Subquery

-- 1. Using subqueries, identify all actors who appear in the film \_Alter Victory\_ in the `sakila` database.

SELECT first\_name, last\_name

FROM actor

WHERE actor\_id IN

( SELECT actor\_id

FROM film\_actor

WHERE film\_id IN

(SELECT film\_id

FROM film

WHERE title = 'ALTER VICTORY'));

-- 2. Using subqueries, display the titles of films that were rented out by an employee named Jon Stephens.

SELECT title

    FROM film

    WHERE film\_id

    IN (SELECT film\_id

        FROM inventory

        WHERE inventory\_id

        IN (SELECT inventory\_id

            FROM rental

            WHERE staff\_id

            IN (SELECT staff\_id

                FROM staff

                WHERE last\_name = "Stephens" AND first\_name = "Jon")));

-- Bonus

-- For problem No. 2, refine the query to select for films that were rented on May 31st,

-- and whose titles begin with the letter \*\*C\*\*. Hint: use \*\*wildcards\*\* in MySQL.

SELECT title

    FROM film

WHERE film\_id

IN (SELECT film\_id

FROM inventory

WHERE inventory\_id

IN (SELECT inventory\_id

FROM rental

WHERE staff\_id

IN (SELECT staff\_id

FROM staff

WHERE first\_name = "Jon" AND last\_name = "Stephens")

                AND rental\_date LIKE '%05-31%'))

                    AND title LIKE 'C%';

3.7 Foreign Keys

we've been dealing with multiple tables that share columns with matching data. They are the basis of joins that we have been performing.

\* Suppose that we have two tables that share the same column.

What would happen if we updated the data in that column in one of the two tables?

What would happen to the data in the same column in the other table?

They would no longer match!

This would be a problem, one that is solved by using a constraint called foreign keys.

\* A foreign key is a link between tables. The foreign key in a first table "points" to, or is linked to, the primary key in a second table.

\* A foreign key also prevents invalid data from being entered into a column.

The data being entered **\*\*has\*\*** to be a value from the referenced column.

think back to the sakila database, where street addresses, cities, and countries were stored in different tables. So if we were to change the address of a customer who moved to a different country, for example, all the information across the tables would need to change.

This is called maintaining the referential integrity.

In a database where the ID number of an employee is used in multiple tables, what happens if the employee's ID number should change? It would need to be changed across all the tables that contain it.

using foreign keys to build relationships between data is a strong suit of relational databases, hence the name.

SEE LESSON PLAN FOR HOW ACTIVITY WAS DONE.

3.8

<https://stackoverflow.com/questions/193780/how-to-find-all-the-tables-in-mysql-with-specific-column-names-in-them>

<https://www.lucidchart.com/pages/er-diagrams>

The official way to pronounce “**MySQL**” is **“My Ess Que Ell”** (not “my sequel”), but we do not mind if you pronounce it as “my sequel” or in some other localized way.

The SQL part of “MySQL” stands for “**Structured Query Language**”.

**S**tructured **Q**uery **L**anguage is a programming language/dialect used to query and manipulate data stored within **relational databases.**

MySQL, the most popular Open Source SQL **database management system**, is developed, distributed, and supported by **Oracle** Corporation.

Others: Postgres, MariaDB, Oracle, SQL Server

The MySQL website (<http://www.mysql.com/>) provides the latest information about MySQL software.

**Notes 9/10/18 - AD**

GUI = The **graphical user interface** is a type of user interface that allows users to interact with

electronic devices through graphical icons and visual indicators such as secondary notation, instead of text-based user interfaces, typed command labels or text navigation.

DDL = A **data definition language** or data **description** language is a syntax similar to a computer

programming language for defining data structures, especially database schemas

DML = A **data manipulation language** is a computer programming language used for adding

(inserting), deleting, and modifying (updating) data in a database.

A **DML** is often a sublanguage of a broader database language such as SQL,

with the **DML** comprising some of the operators in the language.

Understand different kinds of joins & how to use them to create new tables in MySQL:

<http://www.codeproject.com/KB/database/Visual_SQL_Joins/Visual_SQL_JOINS_orig.jpg>

or

<http://www.sql-join.com/sql-join-types>

**Notes 9/8/18 - Manuel**

stongs2 = pw

CRUD = Create Read Update Delete

vs. MongoDB (later) which is NoSQL

Cassandra, Ignite, Dynamo,

Graph database = **Neo4J**

if you'd like additional practice check out this site. Also great to reference SQL commands

<https://www.w3schools.com/sql/sql_ref_sqlserver.asp>

For this homework, you’ll need to load data from a sample database provided by MySQL named [Sakila](https://dev.mysql.com/doc/sakila/en/sakila-installation.html).

All queries noted in the instructions for the homework are tied to this database, so you must complete this step before moving forward.

A database is a structured collection of data… a simple shopping list to a picture gallery or the vast amounts of information in a corporate network.

To add, access, and process data stored in a computer database, you need a database management system such as MySQL Server.

Since computers are very good at handling large amounts of data, database management systems play a central role in computing, as standalone utilities, or as parts of other applications.

**MySQL databases are relational.**

A relational database stores data in separate tables rather than putting all the data in one big storeroom. The database structures are organized into physical files optimized for speed. The logical model, with objects such as databases, tables, views, rows, and columns, offers a flexible programming environment. You set up rules governing the relationships between different data fields, such as one-to-one, one-to-many, unique, required or optional, and “pointers” between different tables. The database enforces these rules, so that with a well-designed database, your application never sees inconsistent, duplicate, orphan, out-of-date, or missing data.

SQL is the most common standardized language used to access databases.

Depending on your programming environment, you might enter SQL directly (for example, to generate reports), embed SQL statements into code written in another language, or use a language-specific API that hides the SQL syntax.

**MySQL software is Open Source.**

Objectives

* Create a localhost connection to a MySQL server and successfully connect to it.
* Create, use, and populate a MySQL database with data.
* Create, populate, and select data from a MySQL table.
* Import large CSV datasets into MySQL Workbench using the import wizard.
* Use MySQL to select specific rows/columns of data out from a table.
* Solidify the foundations of writing basic- to intermediate-level MySQL statements.
* Develop an introductory understanding of table design and database management.

Additional Resources

* [MySQL Documentation](https://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html)
* [MySQL Command Lines](https://gist.github.com/hofmannsven/9164408)
* [MySQL Tutorial](https://dev.mysql.com/doc/refman/5.6/en/tutorial.html)

installing mysqlworkbench:

Mac install: <https://github.com/the-Coding-Boot-Camp-at-UT/UTAUS201807DATA2/blob/master/class-content/09-Project-1/1/Activities/01-Installation/mac_installation.md>

Sakila Database

<https://github.com/the-Coding-Boot-Camp-at-UT/UTAUS201807DATA2/blob/master/homework-instructions/10-SQL/Instructions/Installation.md>

Find **MySQL** in system preferences… **MySQLWorkbench in dock**

Online resources:

• MySQL Reference Manual

• [www.Oracle.com](http://www.Oracle.com)