MySql Workbench

* Open MySQL Workbench
* Take the contents of `animalsDB.sql` and paste it into your MySQL Workbench
* Using MySQL Workbench examine the newly created `animals_db`.

```
-- 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 (
  -- Makes a string column called "name" which cannot contain null --
  name VARCHAR(30) NOT NULL,
  -- Makes a boolean column called "has_pet" which cannot contain null --
  has_pet BOOLEAN NOT NULL,
  -- Makes a sting column called "pet_name" --
  pet_name VARCHAR(30),
  -- Makes an numeric column called "pet_age" --
  pet_age INTEGER(10)
);

-- 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, has_pet)
VALUES ("Peter", false);

-- Updates the row where the column name is peter --
UPDATE people
SET has_pet = true, pet_name = "Franklin", pet_age = 2
WHERE name = "Peter";
```

```
-- 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 an numeric column called "score" --
  score INTEGER(10)
);

CREATE TABLE favorite_songs (
  song VARCHAR(100) NOT NULL,
  artist VARCHAR(50),
  score INTEGER(10)
);

CREATE TABLE favorite_movies (
  id INTEGER NOT NULL AUTO_INCREMENT,
  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 INTEGER(10),
  PRIMARY KEY (id)
);
```

```
-- Drops the favorite_db if it exists currently --
DROP DATABASE IF EXISTS favorite_db;

-- Creates the "favorite_db" database --
CREATE DATABASE favorite_db;

-- Make 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 (
  -- Make a string column called "food" which cannot contain null --
  food VARCHAR(50) NOT Null,
  -- Make an numeric column called "score" --
  score INTEGER(10)
);

CREATE TABLE favorite_songs (
  -- Make a string column called "song" which cannot contain null --
  song VARCHAR(100) NOT NULL,
  -- Make a string column called "artist" --
  artist VARCHAR(50) NOT NULL,
  -- Make an integer column called "score" --
  score INTEGER(0)
);
CREATE TABLE favorite_movies (
  -- Create a numeric column called "id" which automatically increments
            and cannot be null --
  id INTEGER(0) NOT NULL AUTO_INCREMENT,
  -- Create a string column called "movie" which cannot be null --
  movie VARCHAR() NOT NULL,
  -- Create a boolean column called "five_times" that sets the default value
            to false if nothing is entered --
  five_times BOOLEAN DEFAULT false,
  -- Make an integer column called "score" --
  score INT,
  -- Set the primary key of the table to id --
  PRIMARY KEY (id)
);

INSERT INTO favorite_movies(movie) VALUES ("avengers");

SELECT * FROM favorite_movies;

INSERT INTO favorite_movies(movie, five_times) VALUES ("Tron", true);

INSERT INTO favorite_movies(movie) VALUES ("Fatal Fury");

-- error since 2 is already set
-- INSERT INTO favorite_movies(id, movie) VALUES(2, "Time Cop");
```

IDs

```
-- 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 INTEGER(11) AUTO_INCREMENT NOT NULL,
  language VARCHAR(20),
  rating INTEGER(11),
  -- 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);
```

Join

```
DROP DATABASE IF EXISTS books_db;

CREATE DATABASE books_db;

USE books_db;

CREATE TABLE books(
  id INTEGER(11) AUTO_INCREMENT NOT NULL,
  authorId INTEGER(11),
  title VARCHAR(100),
  PRIMARY KEY (id)
);

CREATE TABLE authors(
  id INTEGER(11) 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;
```

Connection

**Remember,
you must create a database before attempting to connect to it.
Doing otherwise will return an error.**

```
const mysql = require('mysql');
const connection = mysql.createConnection({
  host: 'localhost',

  // Your port, if not 3306
  port: 3306,

  // Your username
  user: 'root',

  // Be sure to update with your own MySQL password!
  password: 'rootpass',
  database: 'ice_creamDB',
});

connection.connect((err) => {
  if (err) throw err;
  console.log(`connected as id ${connection.threadId}`);
  connection.end();
});
```

```
  DROP DATABASE IF EXISTS ice_creamDB;

  CREATE DATABASE ice_creamDB;

  USE ice_creamDB;

  CREATE TABLE products (
    id INT NOT NULL AUTO_INCREMENT,
    flavor VARCHAR(45) NULL,
    price DECIMAL(10,2) NULL,
    quantity INT NULL,
    PRIMARY KEY (id)
  );

  INSERT INTO products (flavor, price, quantity)
  VALUES ("vanilla", 2.50, 100);

  INSERT INTO products (flavor, price, quantity)
  VALUES ("chocolate", 3.10, 120);

  INSERT INTO products (flavor, price, quantity)
  VALUES ("strawberry", 3.25, 75);

  -- ### Alternative way to insert more than one row
  -- INSERT INTO products (flavor, price, quantity)
  -- VALUES ("vanilla", 2.50, 100), ("chocolate", 3.10, 120), ("strawberry", 3.25, 75);
```

Read

```javascript
const mysql = require('mysql');

const connection = mysql.createConnection({
  host: 'localhost',

  // Your port; if not 3306
  port: 3306,

  // Your username
  user: 'root',

  // Be sure to update with your own MySQL password!
  password: '',
  database: 'playlistDB',
});

const queryAllSongs = () => {
  connection.query('SELECT * FROM songs', (err, res) => {
    if (err) throw err;
    res.forEach(({ id, title, artist, genre }) => {
      console.log(`${id} | ${title} | ${artist} | ${genre}`);
    });
    console.log('-----------------------------------');
  });
};

const queryDanceSongs = () => {
  const query = connection.query(
    'SELECT * FROM songs WHERE genre=?',
    ['Dance'],
    (err, res) => {
      if (err) throw err;
      res.forEach(({ id, title, artist, genre }) => {
        console.log(`${id} | ${title} | ${artist} | ${genre}`);
      });
    }
  );

  // logs the actual query being run
  console.log(query.sql);
  connection.end();
};

connection.connect((err) => {
  if (err) throw err;
  console.log(`connected as id ${connection.threadId}`);
  queryAllSongs();
  queryDanceSongs();
});
```

```sql
DROP DATABASE IF EXISTS playlistDB;

CREATE DATABASE playlistDB;

USE playlistDB;

CREATE TABLE songs(
  id INT NOT NULL AUTO_INCREMENT,
  title VARCHAR(45) NULL,
  artist VARCHAR(45) NULL,
  genre VARCHAR(45) NULL,
  PRIMARY KEY (id)
);

INSERT INTO songs (title, artist, genre)
VALUES ("Human", "Krewella", "Dance");

INSERT INTO songs (title, artist, genre)
VALUES ("TRNDSTTR","Black Coast", "Dance");

INSERT INTO songs (title, artist, genre)
VALUES ("Who's Next", "The Who", "Classic Rock");

INSERT INTO songs (title, artist, genre)
VALUES ("Yellow Submarine", "The Beatles", "Classic Rock");
```

CRUD

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

const inquirer = require('inquirer');

// create the connection information for the sql database
const connection = mysql.createConnection({
  host: 'localhost',
  // Your port; if not 3306
  port: 3306,
  // Your username
  user: 'root',
  // Your password
  password: '',
  database: 'greatBay_DB',
});

// function which prompts the user for what action they should take
const start = () => {
  inquirer
    .prompt({
      name: 'postOrBid',
      type: 'list',
      message: 'Would you like to [POST] an auction or [BID] on an auction?',
      choices: ['POST', 'BID', 'EXIT'],
    })
    .then((answer) => {
      // based on their answer, either call the bid or the post functions
      if (answer.postOrBid === 'POST') {
        postAuction();
      } else if (answer.postOrBid === 'BID') {
        bidAuction();
      } else {
        connection.end();
      }
    });
};
```

```
DROP DATABASE IF EXISTS greatBay_DB;

CREATE DATABASE greatBay_DB;

USE greatBay_DB;

CREATE TABLE auctions(
  id INT NOT NULL AUTO_INCREMENT,
  item_name VARCHAR(100) NOT NULL,
  category VARCHAR(45) NOT NULL,
  starting_bid INT default 0,
  highest_bid INT default 0,
  PRIMARY KEY (id)
);
```

```
// function to handle posting new items up for auction
const postAuction = () => {
  // prompt for info about the item being put up for auction
  inquirer
    .prompt([
      {
        name: 'item',
        type: 'input',
        message: 'What is the item you would like to submit?',
      },
      {
        name: 'category',
        type: 'input',
        message: 'What category would you like to place your auction in?',
      },
      {
        name: 'startingBid',
        type: 'input',
        message: 'What would you like your starting bid to be?',
        validate(value) {
          if (isNaN(value) === false) {
            return true;
          }
          return false;
        },
      },
    ])
    .then((answer) => {
      // when finished prompting, insert a new item into the db with that info
      connection.query(
        'INSERT INTO auctions SET ?',
        // QUESTION: What does the || 0 do?
        {
          item_name: answer.item,
          category: answer.category,
          starting_bid: answer.startingBid || 0,
          highest_bid: answer.startingBid || 0,
        },
        (err) => {
          if (err) throw err;
          console.log('Your auction was created successfully!');
          // re-prompt the user for if they want to bid or post
          start();
        }
      );
    });
};
```

```
const bidAuction = () => {
  // query the database for all items being auctioned
  connection.query('SELECT * FROM auctions', (err, results) => {
    if (err) throw err;
    // once you have the items, prompt the user for which they'd like to bid on
    inquirer
      .prompt([
        {
          name: 'choice',
          type: 'rawlist',
          choices() {
            const choiceArray = [];
            results.forEach(({ item_name }) => {
              choiceArray.push(item_name);
            });
            return choiceArray;
          },
          message: 'What auction would you like to place a bid in?',
        },
        {
          name: 'bid',
          type: 'input',
          message: 'How much would you like to bid?',
        },
      ])
      .then((answer) => {
        // get the information of the chosen item
        let chosenItem;
        results.forEach((item) => {
          if (item.item_name === answer.choice) {
            chosenItem = item;
          }
        });
        // determine if bid was high enough
        if (chosenItem.highest_bid < parseInt(answer.bid)) {
          // bid was high enough, so update db, let the user know, and start over
          connection.query(
            'UPDATE auctions SET ? WHERE ?',
            [
              {
                highest_bid: answer.bid,
              },
              {
                id: chosenItem.id,
              },
            ],
            (error) => {
              if (error) throw err;
              console.log('Bid placed successfully!');
              start();
            }
          );
        } else {
          // bid wasn't high enough, so apologize and start over
          console.log('Your bid was too low. Try again...');
          start();
        }
      });
  });
};

// connect to the mysql server and sql database
connection.connect((err) => {
  if (err) throw err;
  // run the start function after the connection is made to prompt the user
  start();
});
```

CSV

```javascript
const mysql = require('mysql');
const inquirer = require('inquirer');

const connection = mysql.createConnection({
  host: 'localhost',
  port: 3306,
  user: 'root',
  password: '',
  database: 'top_songsDB',
});

connection.connect((err) => {
  if (err) throw err;
  runSearch();
});

const runSearch = () => {
  inquirer
    .prompt({
      name: 'action',
      type: 'list',
      message: 'What would you like to do?',
      choices: [
        'Find songs by artist',
        'Find all artists who appear more than once',
        'Find data within a specific range',
        'Search for a specific song',
        'exit',
      ],
    })
    .then((answer) => {
      switch (answer.action) {
        case 'Find songs by artist':
          artistSearch();
          break;
        case 'Find all artists who appear more than once':
          multiSearch();
          break;
        case 'Find data within a specific range':
          rangeSearch();
          break;
        case 'Search for a specific song':
          songSearch();
          break;
        case 'Exit':
          connection.end();
          break;
        default:
          console.log(`Invalid action: ${answer.action}`);
          break;
      }
    });
};

const artistSearch = () => {
  inquirer
    .prompt({
      name: 'artist',
      type: 'input',
      message: 'What artist would you like to search for?',
    })
    .then((answer) => {
      const query = 'SELECT position, song, year
                                FROM top5000 WHERE ?';
      connection.query(query, { artist: answer.artist }, (err, res) => {
        if (err) throw err;
        res.forEach(({ position, song, year }) => {
          console.log(
            `Position: ${position} || Song: ${song} || Year: ${year}`
          );
        });
        runSearch();
      });
    });
};

const multiSearch = () => {
  const query =
    'SELECT artist FROM top5000 GROUP BY artist
                                HAVING count(*) > 1';
  connection.query(query, (err, res) => {
    if (err) throw err;
    res.forEach(({ artist }) => console.log(artist));
    runSearch();
  });
};

const rangeSearch = () => {
  inquirer
    .prompt([
      {
        name: 'start',
        type: 'input',
        message: 'Enter starting position: ',
        validate(value) {
          if (isNaN(value) === false) {
            return true;
          }
          return false;
        },
      },
      {
        name: 'end',
        type: 'input',
        message: 'Enter ending position: ',
        validate(value) {
          if (isNaN(value) === false) {
            return true;
          }
          return false;
        },
      },
    ])
    .then((answer) => {
      const query =
        'SELECT position,song,artist,year FROM top5000
                WHERE position BETWEEN ? AND ?';
      connection.query(query, [answer.start, answer.end], (err, res) => {
        if (err) throw err;
        res.forEach(({ position, song, artist, year }) =>
          console.log(
            `Position: ${position} || Song: ${song} || Artist: ${artist}
                          || Year: ${year}`
          )
        );
        runSearch();
      });
    });
};

const songSearch = () => {
  inquirer
    .prompt({
      name: 'song',
      type: 'input',
      message: 'What song would you like to look for?',
    })
    .then((answer) => {
      console.log(`You searched for "${answer.song}"`);
      connection.query(
        'SELECT * FROM top5000 WHERE ?',
        { song: answer.song },
        (err, res) => {
          if (err) throw err;
          if (res[0]) {
            console.log(
              `Position: ${res[0].position} || Song: ${res[0].song}
                          || Artist: ${res[0].artist} || Year: ${res[0].year}`
            );
            runSearch();
          } else {
            console.error('Song not found :(\n');
            runSearch();
          }
        }
      );
    });
};
```

```sql
DROP DATABASE IF EXISTS top_songsDB;

CREATE database top_songsDB;

USE top_songsDB;

CREATE TABLE top5000 (
  position INT NOT NULL,
  artist VARCHAR(100) NULL,
  song VARCHAR(100) NULL,
  year INT NULL,
  raw_total DECIMAL(10,4) NULL,
  raw_usa DECIMAL(10,4) NULL,
  raw_uk DECIMAL(10,4) NULL,
  raw_eur DECIMAL(10,4) NULL,
  raw_row DECIMAL(10,4) NULL,
  PRIMARY KEY (position)
);

SELECT * FROM top5000;
```

Mini Project

```
const mysql = require('mysql');
const inquirer = require('inquirer');

const connection = mysql.createConnection({
  host: 'localhost',
  port: 3306,
  user: 'root',
  password: '',
  database: 'top_songsDB',
});

connection.connect((err) => {
  if (err) throw err;
  runSearch();
});

const runSearch = () => {
  inquirer
    .prompt({
      name: 'action',
      type: 'rawlist',
      message: 'What would you like to do?',
      choices: [
        'Find songs by artist',
        'Find all artists who appear more than once',
        'Find data within a specific range',
        'Search for a specific song',
        'Find artists with a top song and top album in the same year',
      ],
    })
    .then((answer) => {
      switch (answer.action) {
        case 'Find songs by artist':
          artistSearch();
          break;
        case 'Find all artists who appear more than once':
          multiSearch();
          break;
        case 'Find data within a specific range':
          rangeSearch();
          break;
        case 'Search for a specific song':
          songSearch();
          break;
        case 'Find artists with a top song and top album in the same year':
          songAndAlbumSearch();
          break;
        default:
          console.log(`Invalid action: ${answer.action}`);
          break;
      }
    });
};
```

```
DROP DATABASE IF EXISTS top_songsDB;

CREATE database top_songsDB;

USE top_songsDB;

CREATE TABLE top5000 (
  position INT NOT NULL,
  artist VARCHAR(100) NULL,
  song VARCHAR(100) NULL,
  year INT NULL,
  raw_total DECIMAL(10,4) NULL,
  raw_usa DECIMAL(10,4) NULL,
  raw_uk DECIMAL(10,4) NULL,
  raw_eur DECIMAL(10,4) NULL,
  raw_row DECIMAL(10,4) NULL,
  PRIMARY KEY (position)
);
```

```
CREATE TABLE top_albums (
  position INT NOT NULL,
  artist VARCHAR(100) NULL,
  album VARCHAR(100) NULL,
  year INT NULL,
  raw_total DECIMAL(10,4) NULL,
  raw_usa DECIMAL(10,4) NULL,
  raw_uk DECIMAL(10,4) NULL,
  raw_eur DECIMAL(10,4) NULL,
  raw_row DECIMAL(10,4) NULL,
  PRIMARY KEY (position)
);

SELECT * FROM top5000;

select * from top_albums;
```

```
const artistSearch = () => {
  inquirer
    .prompt({
      name: 'artist',
      type: 'input',
      message: 'What artist would you like to search for?',
    })
    .then((answer) => {
      const query = 'SELECT position, song, year FROM top5000 WHERE ?';
      connection.query(query, { artist: answer.artist }, (err, res) => {
        res.forEach(({ position, song, year }) => {
          console.log(
            `Position: ${position} || Song: ${song} || Year: ${year}`
          );
        });
        runSearch();
      });
    });
};

const multiSearch = () => {
  const query =
    'SELECT artist FROM top5000 GROUP BY artist HAVING count(*) > 1';
  connection.query(query, (err, res) => {
    res.forEach(({ artist }) => console.log(artist));
    runSearch();
  });
};

const rangeSearch = () => {
  inquirer
    .prompt([
      {
        name: 'start',
        type: 'input',
        message: 'Enter starting position: ',
        validate(value) {
          if (isNaN(value) === false) {
            return true;
          }
          return false;
        },
      },
      {
        name: 'end',
        type: 'input',
        message: 'Enter ending position: ',
        validate(value) {
          if (isNaN(value) === false) {
            return true;
          }
          return false;
        },
      },
    ])
    .then((answer) => {
      const query =
        'SELECT position,song,artist,year FROM top5000
              WHERE position BETWEEN ? AND ?';
      connection.query(query, [answer.start, answer.end], (err, res) => {
        res.forEach(({ position, song, artist, year }) => {
          console.log(
            `Position: ${position} || Song: ${song} || Artist: ${artist} || Year: ${year}`
          );
        });
        runSearch();
      });
    });
};
```

```
const songSearch = () => {
  inquirer
    .prompt({
      name: 'song',
      type: 'input',
      message: 'What song would you like to look for?',
    })
    .then((answer) => {
      console.log(answer.song);
      connection.query(
        'SELECT * FROM top5000 WHERE ?',
        { song: answer.song },
        (err, res) => {
          if (res[0]) {
            console.log(
              `Position: ${res[0].position} || Song: ${res[0].song}
                  || Artist: ${res[0].artist} || Year: ${res[0].year}`
            );
          } else {
            console.error(`No results for ${answer.song}`);
          }
          runSearch();
        }
      );
    });
};

const songAndAlbumSearch = () => {
  inquirer
    .prompt({
      name: 'artist',
      type: 'input',
      message: 'What artist would you like to search for?',
    })
    .then((answer) => {
      let query =
        'SELECT top_albums.year, top_albums.album, top_albums.position,
              top5000.song, top5000.artist ';
      query +=
        'FROM top_albums INNER JOIN top5000 ON
              (top_albums.artist = top5000.artist AND top_albums.year ';
      query +=
        '= top5000.year) WHERE (top_albums.artist = ? AND top5000.artist = ?)
              ORDER BY top_albums.year, top_albums.position';
      connection.query(query, [answer.artist, answer.artist], (err, res) => {
        console.log(`${res.length} matches found!`);
        res.forEach(({ year, position, artist, song, album }, i) => {
          const num = i + 1;
          console.log(
            `${num} Year: ${year} Position: ${position} || Artist: ${artist}
                || Song: ${song} || Album: ${album}`
          );
        });
        runSearch();
      });
    });
};
```