**MYSQL Download link : https://dev.mysql.com/downloads/mysql/**

**To log in MYSQL :**

1. **Window key + R** then type : **cmd** to go to command line

Then :  **mysql -u root -p**

Enter password – **R11@**

This will log you in MYSQL Shell

Type quit; to exit the MySQL Shell.

To begin, first confirm that the MySQL server is running. Type the following command into the terminal:

mysql --version

Command To create database:

CREATE DATABASE election;

USE election;

To create table :

CREATE TABLE candidates (

id INTEGER AUTO\_INCREMENT PRIMARY KEY,

first\_name VARCHAR(30) NOT NULL,

last\_name VARCHAR(30) NOT NULL,

industry\_connected BOOLEAN NOT NULL

);

To verify that the table was created successfully, we can use the following MySQL CLI command:

DESCRIBE candidates;

This command describes the table's fields and field attributes but doesn't display any of the table's data or records.

SQL statement that will load data into the table

INSERT INTO candidates (first\_name, last\_name, industry\_connected)

VALUES ('Ronald', 'Firbank', 1);

Verify that the data has been inserted into the table by using a query that will return all rows of data in the table.

SELECT \* FROM candidates;

To add more data user INSERT statement to add multiple rows

INSERT INTO candidates (first\_name, last\_name, industry\_connected)

VALUES

('Virginia', 'Woolf', 1),

('Piers', 'Gaveston', 0),

('Charles', 'LeRoi', 1),

('Katherine', 'Mansfield', 1),

('Dora', 'Carrington', 0),

('Edward', 'Bellamy', 0),

('Montague', 'Summers', 1),

('Octavia', 'Butler', 1),

('Unica', 'Zurn', 1);

Now suppose we wish to retrieve only specific column values from the table, like just the first\_name and last\_name.

SELECT first\_name, last\_name FROM candidates;

we can retrieve specific information by naming the columns after the SELECT statement.

Suppose next that we want to find, by first name, all the candidates who are industry\_connected. The WHERE clause enables us to conditionally filter through the data in the table.

SELECT first\_name, industry\_connected

FROM candidates

WHERE industry\_connected = 1;

The WHERE clause is a powerful filtering tool that can be used with equality operators like less than (<) or not equal to (!=). We can also use the OR and AND logical operators to evaluate multiple conditions. If the expression evaluates to true, the row is returned.

<https://dev.mysql.com/doc/refman/8.0/en/comparison-operators.html>

We can also select a specific row using the primary key with the id. In the following SQL statement, we'll select the row with the id of 5:

SELECT first\_name, last\_name, industry\_connected

FROM candidates

WHERE id = 5;

To check all databases, type :

show databases;

Delete, or DROP, the database with the following SQL statement in the MySQL Shell:

DROP DATABASE election;

The MySQL Shell command source executes script files. With the preceding code, we're executing the db.sql file located in the db folder,

source db/db.sql

source db/schema.sql

DESCRIBE candidates; // to check candidate table created

source db/seeds.sql

To change the data for a particular row. Perhaps a name was misspelled or a candidate who was previously lacking an industry connection has one now. To handle this type of change, we can use an UPDATE statement.

UPDATE candidates

SET industry\_connected = 1

WHERE id = 3;

To delete candidate entry :

DELETE FROM candidates

WHERE first\_name = "Montague";

To check table, Use :

SELECT \* FROM candidates;

CRUD = CREATE, READ, UPDATE, DELETE

**Setup Node.js application with MySQL**

initialize the Node.js application, and install the npm modules that we'll need.

1 ) npm init --y

create the .gitingore file with the node\_modules/ from command line using :

2 ) echo "node\_modules/" > .gitignore

**or use :** “ touch .gitignore “ in command line and then add “ node\_modules/ “ inside it

Install npm packages **express** and **mysql2** both :

3 ) npm install express mysql2

Or

npm install express

npm install --save mysql2

Install & setup testing using Jest

npm install jest --save-dev

Change the current "test" script to the package.json file to the following:

"scripts": {

"test": "jest",

},

Run the test suite in the terminal and confirm whether the function passes the tests,

npm test

Add one more script to start the Express.js server. Add the following code to the package.json file, under the test script:

"start": "node server.js"

Check that your MySQL Server is running—initiate the MySQL command line by typing the following in your command line:

mysql -u root -p

check whether the election database exists by typing the following command:

SHOW DATABASES;

If you see the election database listed, switch to that database by typing the following command:

USE election;

Next, check whether the candidates table is set up correctly by typing the following command:

SHOW TABLES;

Finally, to check whether the database is seeded correctly, type the following SQL query:

SELECT \* FROM candidates;

**Create Connection to Express.js Server**

Create the server.js file in the root directory, using the following command at the command line:

touch server.js

 Open the server.js file and import express at the top of the file, by adding the following code:

const express = require('express');

 Add the PORT designation and the app expression, by adding the following code:

const PORT = process.env.PORT || 3001;

const app = express();

 Add the Express.js middleware, by adding the following code:

// Express middleware

app.use(express.urlencoded({ extended: false }));

app.use(express.json());

 Now let's add the function that will start the Express.js server on port 3001. Place the following code at the bottom of the server.js file:

app.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});

create a GET test route and place it above the connection function that we just created, as shown in the following example:

app.get('/', (req, res) => {

res.json({

message: 'Hello World'

});

});

// Default response for any other request (Not Found)

app.use((req, res) => {

res.status(404).end();

});

At the terminal, execute the script we added to start the Express.js server, as follows:

npm start

**to end server, type :**  ctrl + c

open a browser and type the local host port connection into the address bar:

<http://localhost:3001>

## Connect to the MySQL Database

In this step, we'll connect to the MySQL database. To begin, we'll need to import the mysql2 package that we installed previously into the node\_modules.

At the top of the file, insert the following statement:

const mysql = require('mysql2');

Now let's add the code that will connect the application to the MySQL database. Add the following expression under the Express.js middleware section in server.js, right after app.use(express.json());:

// Connect to database

const db = mysql.createConnection(

{

host: 'localhost',

// Your MySQL username,

user: 'root',

// Your MySQL password

password: '',

database: 'election'

},

console.log('Connected to the election database.')

);

Replace the values of user and password with your MySQL username and password.

Build database Calls Examples : -

// Get all candidates

app.get('/api/candidates', (req, res) => {

    const sql = `SELECT \* FROM candidates`;

    db.query(sql, (err, rows) => {

      if (err) {

        res.status(500).json({ error: err.message });

        return;

      }

      res.json({

        message: 'success',

        data: rows

      });

    });

  });

  // Get a single candidate

app.get('/api/candidate/:id', (req, res) => {

    const sql = `SELECT \* FROM candidates WHERE id = ?`;

    const params = [req.params.id];

    db.query(sql, params, (err, row) => {

      if (err) {

        res.status(400).json({ error: err.message });

        return;

      }

      res.json({

        message: 'success',

        data: row

      });

    });

  });

  // Delete a candidate

app.delete('/api/candidate/:id', (req, res) => {

    const sql = `DELETE FROM candidates WHERE id = ?`;

    const params = [req.params.id];

    db.query(sql, params, (err, result) => {

      if (err) {

        res.statusMessage(400).json({ error: res.message });

      } else if (!result.affectedRows) {

        res.json({

          message: 'Candidate not found'

        });

      } else {

        res.json({

          message: 'deleted',

          changes: result.affectedRows,

          id: req.params.id

        });

      }

    });

  });

  // Create a candidate

app.post('/api/candidate', ({ body }, res) => {

    const errors = inputCheck(body, 'first\_name', 'last\_name', 'industry\_connected');

    if (errors) {

      res.status(400).json({ error: errors });

      return;

    }

    const sql = `INSERT INTO candidates (first\_name, last\_name, industry\_connected)

    VALUES (?,?,?)`;

  const params = [body.first\_name, body.last\_name, body.industry\_connected];

  db.query(sql, params, (err, result) => {

    if (err) {

      res.status(400).json({ error: err.message });

      return;

    }

    res.json({

      message: 'success',

      data: body

    });

   });

  });

A foreign key is a field in one table that references the primary key of another table.

To add a new field without losing any of the existing data

ALTER TABLE candidates ADD COLUMN party\_id INTEGER;

Diagram

Description automatically generated with medium confidence

To alter Candidates : --

SHOW databases;

USE election;

Show tables;

SELECT \* FROM candidates;

**ALTER TABLE candidates ADD COLUMN party\_id INTEGER;**

DESCRIBE candidates;

To delete a tablet, use :

DROP TABLE

As noted previously, the order of table creation is vital due to the dependency of the candidates table on the existence of a parties.id. In the same regard, the candidates table must be dropped before the parties table due to the foreign key constraint that requires the parties table to exist.

There are a few different ways JOIN statements can be used.

A picture containing text, athletic game, sport

Description automatically generated

Restore the database by running the db.sql, schema.sql and seeds.sql files

source db/db.sql

source db/schema.sql

source db/seeds.sql

**SQL's JOIN (Query) statement :**

SELECT \* FROM candidates

LEFT JOIN parties ON candidates.party\_id = parties.id;

This will still return all of the candidate's data but only the party's name.

SELECT candidates.\*, parties.name

FROM candidates

LEFT JOIN parties ON candidates.party\_id = parties.id;

The **AS** keyword lets you define an **alias** for your data, which is particularly useful when joining tables that might have overlapping field names.

SELECT candidates.\*, parties.name AS party\_name

FROM candidates

LEFT JOIN parties ON candidates.party\_id = parties.id;

Text

Description automatically generated

Start your server by running “ npm start “ or “ node server.js “

Sort Options in SQL Examples :

-- get all voters who do not have a last name of Cooper or Jarman

SELECT \* FROM voters WHERE last\_name != 'Cooper' AND last\_name != 'Jarman';

-- get all voters who have a .edu email address

SELECT \* FROM voters WHERE email LIKE '%.edu';

-- get only the last created voter

SELECT \* FROM voters ORDER BY created\_at DESC LIMIT 1;

-- get sorted by last name in descending order

`SELECT \* FROM voters ORDER BY last\_name DESC`;

In the case of COUNT(), it will count how many times a certain field value appears.

SELECT COUNT(candidate\_id) FROM votes;

Other useful aggregate functions in SQL include:

* AVG() to return the average value within a group
* SUM() to add up all of the values in a group
* MIN() to return the minimum value of a group

GROUP BY can consolidate several rows of data, grouping by a shared value (e.g., candidate\_id). The nice thing about GROUP BY is that you can then run an aggregate function to retrieve an average, sum, or minimum value from the group.

SELECT COUNT(candidate\_id) FROM votes GROUP BY candidate\_id;