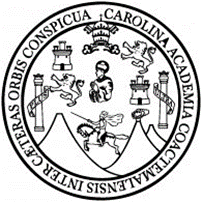
Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ciencias y Sistemas

Sistemas de Bases de Datos 1 “A”

**Proyecto 1**

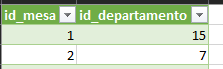
Estuardo Gabriel Son Mux – 202003894

# Modelo Conceptual

El modelo conceptual de la base de datos fue realizado por mediante el análisis de los requerimientos del problema, en este caso se tenía también el apoyo de los archivos csv que nos daban una idea de las entidades que conformarían la base de datos y pudiéndolos unir en un solo archivo de Excel nos damos una idea más clara:

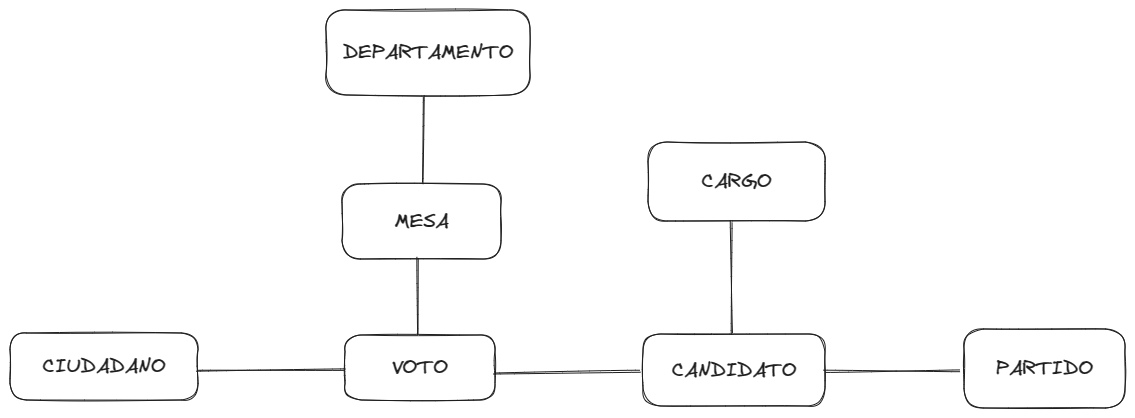


Ahora bien, para las relaciones que podrían estar sucediendo entre las entidades en un principio solo reconoceremos dos: uno a muchos (una linea) y muchos a muchos (doble linea). Si vemos más detenidamente los campos de las tablas de los csv veremos que hay referencias a campos de otras tablas como por ejemplo el archivo mesa en donde se hace referencia al id de departamento:



Esto nos indica que hay una relación de entre ambas entidades y podemos hacer una relación.

Una vez realizado el análisis mediante el uso de la herramienta excalidrab se realiza un diagrama con la información.

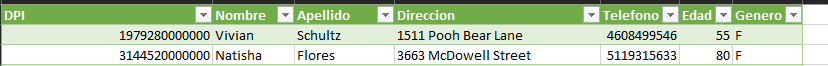


Inicialmente se reconocieron siete tablas con las cuales se trabajarán.

# Modelo Lógico

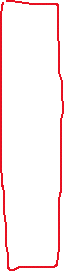
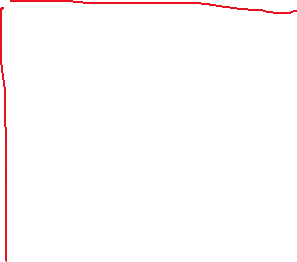
Una vez completado el modelo lógico se procede a la elaboración del modelo lógico con el cual se profundizan las relaciones entre las entidades de la base de datos, sin embargo, para ello se debe hacer un análisis más profundo para identificar atributos que puedan ser divididos en atributos más simples con el fin de hacer más evidentes las relaciones.

Si vemos los archivos csv estos también nos dan una idea de los campos que deben llevar las tablas como por ejemplo el archivo ciudadano:

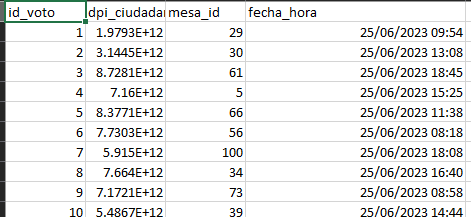
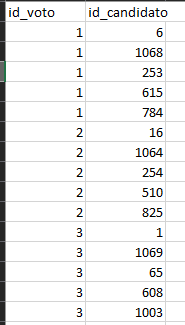


Sin embargo, no podemos simplemente cargar los datos así a nuestro modelo ya que debemos aplicar casos de normalización para mantener la integridad de los datos y evitar datos repetidos. En este caso la única tabla que podemos normalizar es la tabla que se encuentra en el archivo Voto.csv, en ella podemos notar que hay varios campos que se repiten y podrían agruparse para formar otra tabla:

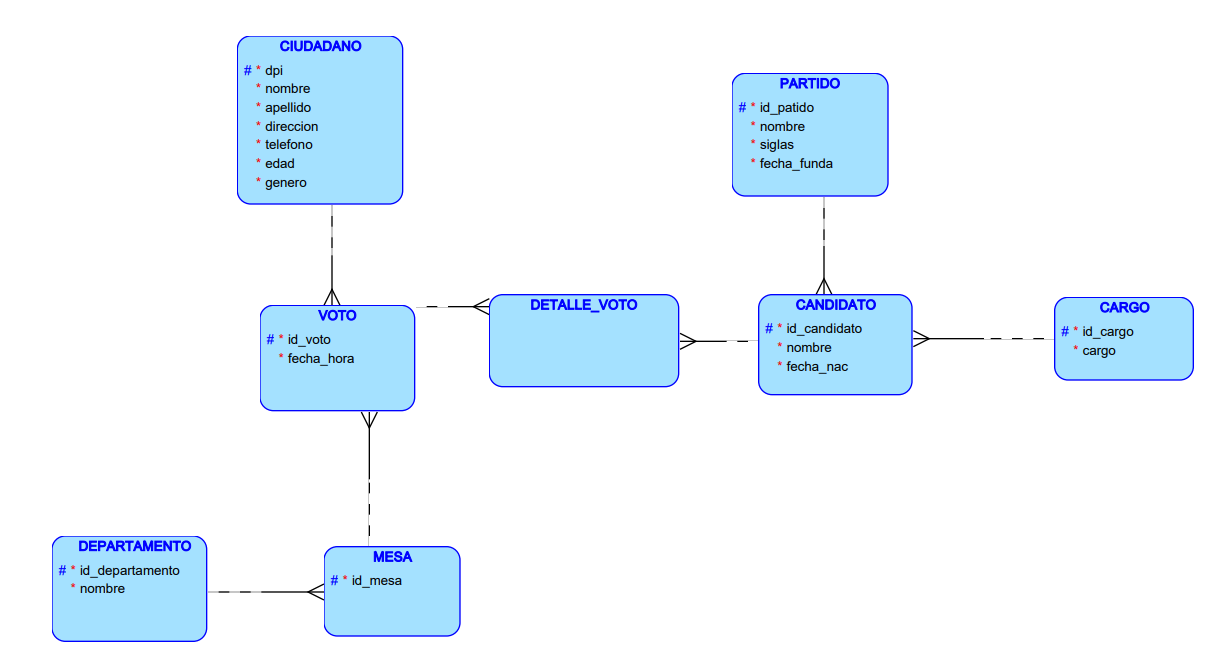




Como se puede apreciar los valores de las columnas id\_voto, dpi\_ciudadano, mesa\_id y fecha\_hora por lo que se agrupan en una tabla que denominaremos como la nueva tabla voto y los datos no repitentes se agruparan y trasladan a otra tabla que denominamos detalle votos, junto con el id del voto al que pertenece.

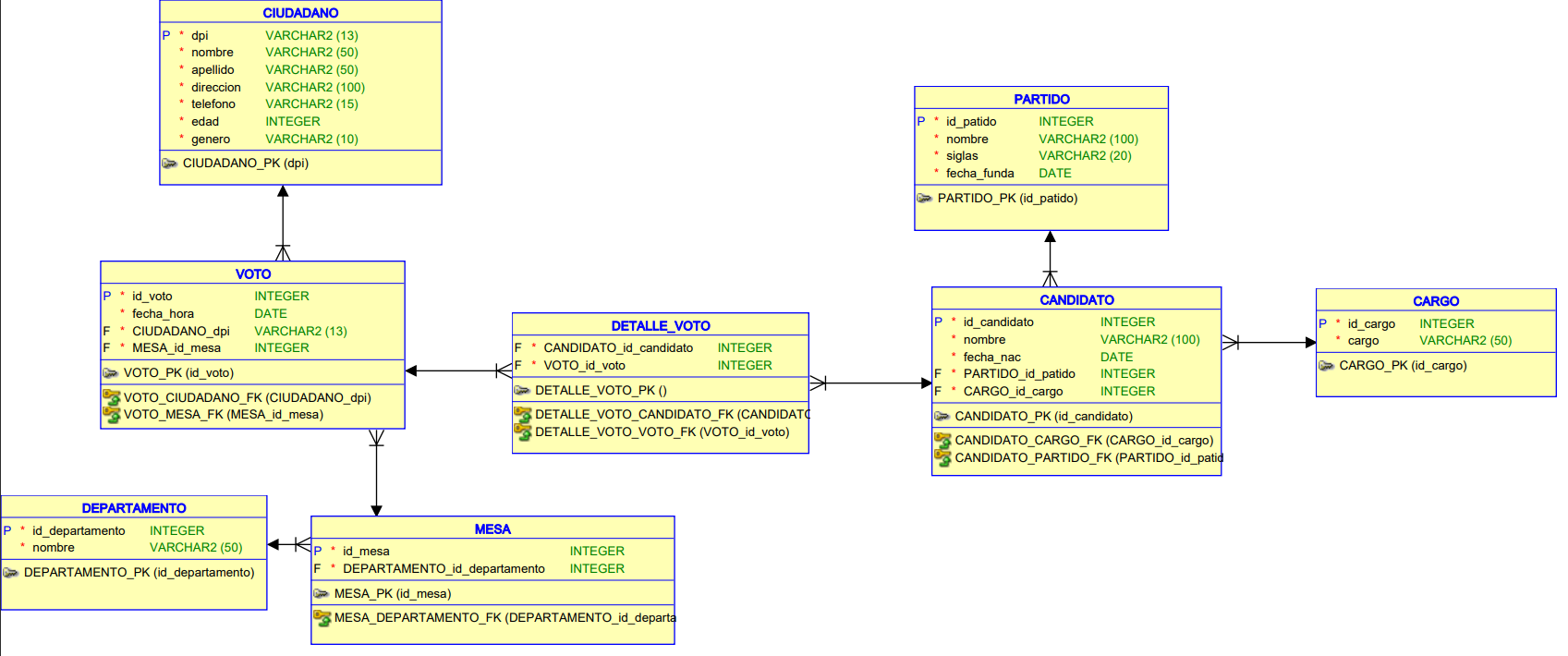
 

Una vez normalizados los votos procedemos a la elaboración del modelo lógico con la ayuda de DataModeler, obteniendo el siguiente resultado:

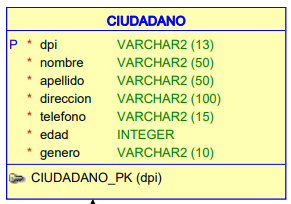


# Modelo Físico

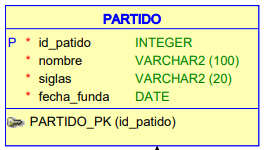
Una vez completado el modelo físico y abstraídos todas las características se añade al modelo físico los atributos necesarios para identificar con más precisión las relaciones entre las entidades identificando llaves primarias y foráneas, así como el tipo de valores que aceptara cada tipo de dato siendo varchar, int o datetime dependiendo del caso.



Todas las relaciones que se entran en el diagrama son de uno a muchos siendo la tabla que posee la llave foránea la que tiene la denominada “pata de gallo”.



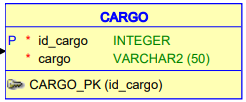
La tabla ciudadana se compone de los atributos dpi, nombre, apellido dirección y genero de tipo varchar, mientras que el atributo edad es de tipo integer ya que este solo contiene datos numéricos, en teléfono se decidió utilizar varchar ya que algunos números pueden contener caracteres como + o paréntesis, y el atributo dpi funciona como llave primaria al ser un identificador único de cada registro.



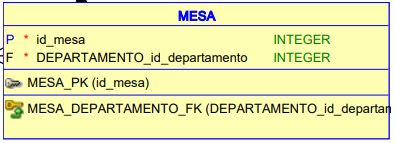
La tabla partido se conforma de los atributos id que funciona como llave primaria de tipo entero, los atributos nombre y siglas de tipo varchar y el atributo fecha\_funda de tipo date.



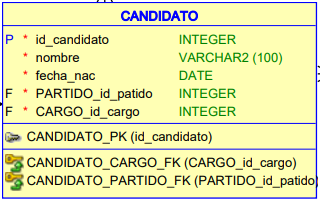
La tabla partido posee únicamente dos atributos. El primero es id\_departamento que es utilizado como llave primaria y es de tipo integer, y el atributo nombre de tipo varchar.



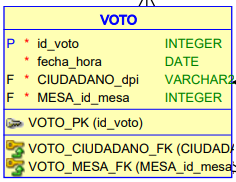
La tabla partido posee únicamente dos atributos. El primero es id\_departamento que es utilizado como llave primaria y es de tipo integer, y el atributo cargo de tipo varchar.



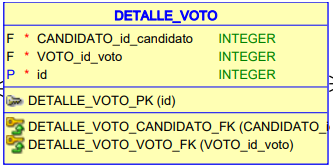
La tabla partido posee únicamente dos atributos. El primero es id\_mesa que es utilizado como llave primaria y es de tipo integer, y el atributo departamento que funciona como referencia al id de un registro en la tabla departamento al ser una llave foránea de tipo integer.



La tabla candidato posee el atributo id\_candidato de tipo integer como llave primaria, el atributo nombre de tipo varchar, el atributo fecha\_nac de tipo date y los atributos partido y cargo que pese a ser integer son también llaves foráneas que referencias id de registros en las tablas partido y cargo respectivamente.



La tabla voto se conforma de los atributos id\_voto que es una llave primaria de tipo integer, fecha\_hora de tipo datetime y las llaves foráneas de tipo integer, ciudadano y mesa.



La tabla detalle\_voto se conforma de tres atributos de los cuales los atributos candidato y voto son de tipo integer y son llaves foráneas mientras que el atributo id funciona como llave primaria.

Una vez definido completamente el modelo se puede pasar a la codificación en el DBMS MySQL para hacer pruebas.

# MySQL Script

Antes de empezar cabe aclarar que en el script no se muestran las tablas temporales elaboradas ya que estas se crean hasta ejecutar la Api, pero son en esencia las mismas tablas que se utilizan en para el modelo final con excepción de la tabla TEMP que contiene la información de las votaciones sin normalizar.

*-- Creacion de la base de datos*

DROP DATABASE IF EXISTS PRACTICA1BASES1;

CREATE DATABASE PRACTICA1BASES1;

USE PRACTICA1BASES1;

ALTER SCHEMA PRACTICA1BASES1  DEFAULT CHARACTER SET utf8  DEFAULT COLLATE utf8\_spanish\_ci ;

*-- Creación de la tabla ciudadano*

CREATE TABLE CIUDADANO(

dpi VARCHAR(13) PRIMARY KEY,

nombre VARCHAR(50),

apellido VARCHAR(50),

direccion VARCHAR(100),

telefono VARCHAR(15),

edad INT,

genero VARCHAR(10)

);

*-- Creacion de la tabla departamento*

CREATE TABLE DEPARTAMENTO(

id INT PRIMARY KEY,

nombre VARCHAR(50)

);

*-- Creacion de la tabla mesa*

CREATE TABLE MESA(

id INT PRIMARY KEY,

departamento INT,

FOREIGN KEY (departamento) REFERENCES DEPARTAMENTO(id)

);

*--Creacion de la tabla voto*

CREATE TABLE VOTO(

id INT PRIMARY KEY,

fecha\_hora DATETIME,

dpi VARCHAR(13),

mesa INT,

FOREIGN KEY (dpi) REFERENCES CIUDADANO(dpi),

FOREIGN KEY (mesa) REFERENCES MESA(id)

);

*--Creacion de la tabla partido*

CREATE TABLE PARTIDO(

id INT PRIMARY KEY,

nombre VARCHAR(100),

siglas VARCHAR(20),

fecha\_funda DATE

);

*-- Creacion de la tabla cargo*

CREATE TABLE CARGO(

id INT PRIMARY KEY,

cargo VARCHAR(50)

);

*-- Creacion de la tabla candidato*

CREATE TABLE CANDIDATO(

id INT PRIMARY KEY,

nombre VARCHAR(100),

fecha\_nac DATE,

partido INT,

cargo INT,

FOREIGN KEY (partido) REFERENCES PARTIDO(id),

FOREIGN KEY (cargo) REFERENCES CARGO(id)

);

*-- Creacion de la tabla detalle\_voto*

CREATE TABLE DETALLE\_VOTO(

id int primary key auto\_increment,

candidato INT,

voto INT,

FOREIGN KEY(candidato) REFERENCES CANDIDATO(id),

FOREIGN KEY(voto) REFERENCES VOTO(id)

);

SELECT count(\*) as ciudadano FROM CIUDADANO;

SELECT count(\*) FROM CANDIDATO;

SELECT count(\*) as partido FROM PARTIDO;

SELECT count(\*) as cargo FROM CARGO;

SELECT count(\*) as mesa FROM MESA;

SELECT count(\*) as voto FROM VOTO;

SELECT count(\*) as departamento FROM DEPARTAMENTO;

SELECT count(\*) as detalle FROM DETALLE\_VOTO;

SELECT \* FROM TEMP;

## Consultas

*-- Extraer votos*

Como bien mencionábamos en un inicio la tabla TEMP contiene la información de todos los votos sin normalizar por lo que para extraer únicamente los datos que queremos para la tabla Voto del modelo utilizamos la siguiente consulta:

SELECT id\_voto,dpi,mesa,fecha\_hora FROM TEMP GROUP BY id\_voto,dpi,mesa,fecha\_hora;

*-- Extraer detalle voto*

La consulta siguiente es para obtener todos los registros en la tabla TEMP con los datos que hicieron falta de la consulta anterior y colocarlos en la tabla DETALLE\_VOTO:

SELECT id\_voto,id\_candidato FROM TEMP;

*-- Presidente y vice por partido*

Para esta consulta fue necesaria la utilización de dos inner join las cuales obtenían los candidatos con el mismo partido y enlazarlos con la tabla partido, y en base a eso devuelva el los registros siempre y cuando el cargo asignado a los candidatos sea el de presidente y viceprecidente.

SELECT

    P.nombre AS presidente,

    V.nombre AS vicepresidente,

    PARTIDO.nombre AS partido

FROM

    CANDIDATO AS P

INNER JOIN

    CANDIDATO AS V ON P.partido = V.partido

INNER JOIN

    PARTIDO ON P.partido = PARTIDO.id

WHERE

    P.cargo = 1

    AND V.cargo = 2;

*-- Cantidad diputados por partido*

Para esta consulta fue vital el uso de la función SUM, los comando CASE WHEN que funcionan como un if else, en este caso hace una consulta a la tabla Candidato y la une con la tabla partido para obtener los candidatos de y su respectivo partido, la utilidad de sum es de sumar un 1 o 0 cada vez que un registro cumpla con la condición ingresada en el CASE WHEN.

SELECT

    Partido.nombre AS partido,

    SUM(CASE WHEN C.cargo = 3 THEN 1 ELSE 0 END) AS "Diputados congreso lista nacional",

    SUM(CASE WHEN C.cargo = 4 THEN 1 ELSE 0 END) AS "Diputados congreso distrito electoral",

    SUM(CASE WHEN C.cargo = 5 THEN 1 ELSE 0 END) AS "Diputados parlamento centroamericano",

    COUNT(C.id) AS "Cantidad de Candidatos"

FROM

    Candidato C

INNER JOIN

    Partido ON C.partido = Partido.id

WHERE

    C.cargo IN (3, 4, 5)

GROUP BY

    Partido.id, Partido.nombre;

*-- Alcande por partido*

Esta consulta une la tabla Candidato con la tabla partido y devuelve los registros cuyo partido sea igual y el cargo del candidato sea alcalde.

SELECT Partido.nombre as partido, C.nombre as alcalde

FROM Candidato C

INNER JOIN

    PARTIDO Partido ON C.partido = Partido.id

WHERE

    C.cargo=6

;

*-- Cantidad candidatos por partido*

Para esta consulta fue vital el uso de la función SUM, los comando CASE WHEN que funcionan como un if else, en este caso la instrucción SUM suma un 1 o 0 en caso de que la condición en su interior se cumpla.

SELECT P.nombre as partido,

SUM(CASE WHEN C.cargo = 1 THEN 1 ELSE 0 END) as presidente,

SUM(CASE WHEN C.cargo = 2 THEN 1 ELSE 0 END) as vicepresidente,

SUM(CASE WHEN C.cargo = 3 OR C.cargo = 4 OR C.cargo = 5 THEN 1 ELSE 0 END) as diputados,

SUM(CASE WHEN C.cargo = 6 THEN 1 ELSE 0 END) as alcaldes

FROM

    PARTIDO P

INNER JOIN

    CANDIDATO C ON P.id = C.partido

GROUP BY

    P.nombre, P.id;

*-- Votos por departamento*

Para esta consulta se unen las tablas Departamento, Mesa, Voto para obtener los registros cuyo respectivos id concuerden para agruparlos por nombre de departamento e id del departamento.

SELECT D.nombre AS DEPARTAMENTO, COUNT(V.id) AS "CANTIDAD DE VOTOS" FROM DEPARTAMENTO D

INNER JOIN MESA M ON M.departamento = D.id

INNER JOIN VOTO V ON V.mesa = M.id

GROUP BY D.nombre, D.id

;

*-- Votos nulos*

Para esta consulta se unen la tablas Detalle\_Voto y Voto para obtener los registros que tienen como candidato al candidato -1 lo que indica que es un voto nulo.

SELECT COUNT(DISTINCT DV.voto) as "VOTO NULOS" FROM DETALLE\_VOTO DV

INNER JOIN VOTO V ON V.id = DV.voto

WHERE DV.candidato = -1;

*-- Top edades*

Para esta consulta se une la tabla Voto con ciudadano y se agrupan en base a las edades de los ciudadanos para obtener a así el contador de veces en que ciudadanos de la misma edad emitieron su voto.

SELECT C.edad, COUNT(C.dpi) as Cantidad FROM VOTO V

INNER JOIN

CIUDADANO C ON C.dpi = V.dpi

GROUP BY

C.edad

ORDER BY

COUNT(C.dpi)

DESC LIMIT 10;

*-- Top candidatos presidente y vicepresidente*

En este caso se unen las tablas candidato dos veces y con detalle\_voto para obtener los registros cuyo partido sea el mismo y su cargo sea el de presidente y vicepresidente, añadiendo el plus de agrupar los registros en base a la cantidad de veces en las que se voto por un presidente.

SELECT

    P.nombre AS presidente,

    V.nombre AS vicepresidente,

    COUNT(P.id) AS "Cantidad Votos"

FROM

    CANDIDATO P

INNER JOIN

    CANDIDATO V ON P.partido = V.partido

INNER JOIN

    DETALLE\_VOTO DV ON DV.candidato = P.id

WHERE

    P.cargo = 1 AND V.cargo = 2

GROUP BY

    P.id, V.id, P.nombre, V.nombre

ORDER BY COUNT(P.id) DESC LIMIT 10;

*-- Mesas más frecuentadas*

Para esta consulta se unieron las tablas Voto, Mesa y Departamento para poder agrupar las mesas por id y obtener así la cantidad de votos por mesa en cada departamento agrupando los registros por el id de la Mesa.

SELECT V.mesa, D.nombre, COUNT(V.mesa) AS "Cantidad votantes"

FROM VOTO V

INNER JOIN

    MESA M ON V.mesa = M.id

INNER JOIN

    DEPARTAMENTO D ON M.departamento = D.id

group by

    M.id

ORDER BY COUNT(V.mesa) Desc;

*-- Horas mas frecuentadas*

SELECT TIME(V.fecha\_hora) AS HORA, COUNT(V.fecha\_hora) AS CANTIDAD FROM VOTO V *GROUP BY V.fecha\_hora ORDER BY COUNT(V.fecha\_hora) DESC LIMIT 5;*

*-- Votos por genero*

Esta consulta devuelve el total de votos que realizo cada genero mediante la unión de las tablas Ciudadano y Voto, y agrupando los datos por genero.

SELECT C.genero, COUNT(C.genero) AS "Cantidad de votos" FROM CIUDADANO C

INNER JOIN

    VOTO V ON C.dpi = V.dpi

GROUP BY

    C.genero;

# Creación de la API

Para la creación del api se utilizó un servidor en Nodejs con conexión a base de datos MySQL. Así mismo para carga a la base de datos se utilizaron diferentes archivos .csv que contienen la información.

## Archivos CSV

Los archivos csv se almacenan en una carpeta llamada “API” dentro del proyecto.



## Conexión a la base de datos

Importación de las llibrerías

const mysql = require(“mysql2”);

const dotenv = require(“dotenv”).config();

Establecimiento de la conexión

const connection = mysql.createConnection({

  host: process.env.HOST,

  user: process.env.USER\_NAME,

  password: process.env.PASSWORD,

  dateStrings: true,

});

Creación de la base de datos

const getConnection = () => {

  try {

    connection.promise().query(`CREATE DATABASE IF NOT EXISTS ${process.env.DATABASE};`);

    connection.promise().query(

        `ALTER DATABASE ${process.env.DATABASE} CHARACTER SET utf8 COLLATE utf8\_spanish\_ci;`

      );

    connection.changeUser({ database: process.env.DATABASE });

    return connection.promise();

  } catch (error) {

    console.error("Error al establecer la conexión:", error);

    throw conexion;

  }

};

Exportación de la información

module.exports = {

  getConnection,

};

## Index.js

Importación de librerías

const express = require('express');

const cors = require('cors');

const app = express();

const morgan = require('morgan');

const router = require('./router/router');

Middleware

app.use(morgan(‘dev’));

app.use(express.json());

app.use(cors());

Uso del router para acceder a endpoints

app.use(‘/’, router);

const port = 5000;

Ejecución del servidor

app.listen(port, () => {

  console.log(`Information: Server running on http://localhost:${port}`);

});

## Router.js

Importar librerías

const express = require('express');

const router = express.Router();

const queryController = require('../controller/queryController');

const fillController = require('../controller/fillController');

*Establecer las rutas para cargar los datos al modelo*

router.get('/crearmodelo', fillController.crearmodelo);

router.get('/eliminarmodelo', fillController.eliminarmodelo);

router.get('/cargartabtemp', fillController.cargartabtemp);

*Establecer las rutas para realizar las consultas*

router.get('/consulta1', queryController.consulta1);

router.get('/consulta2', queryController.consulta2);

router.get('/consulta3', queryController.consulta3);

router.get('/consulta4', queryController.consulta4);

router.get('/consulta5', queryController.consulta5);

router.get('/consulta6', queryController.consulta6);

router.get('/consulta7', queryController.consulta7);

router.get('/consulta8', queryController.consulta8);

router.get('/consulta9', queryController.consulta9);

router.get('/consulta10', queryController.consulta10);

router.get('/consulta11', queryController.consulta11);

module.exports = router;

## Fillcontroller.js

const fs = require("fs");

const { getConnection } = require("../database/database");

const { error } = require("console");

const connection = getConnection();

exports.crearmodelo = async (req, res) => {

*// Cración de las tablas del modelo*

  const result = await connection.query(

    `CREATE TABLE IF NOT EXISTS CIUDADANO(dpi VARCHAR(13) PRIMARY KEY,nombre VARCHAR(50),apellido VARCHAR(50),direccion VARCHAR(100),telefono VARCHAR(15),edad INT,genero VARCHAR(10));`

  );

  const result1 = await connection.query(

    `CREATE TABLE IF NOT EXISTS DEPARTAMENTO(id INT PRIMARY KEY,nombre VARCHAR(50));`

  );

  const result6 = await connection.query(

    `CREATE TABLE IF NOT EXISTS MESA(id INT PRIMARY KEY,departamento INT,FOREIGN KEY (departamento) REFERENCES DEPARTAMENTO(id));`

  );

  const result7 = await connection.query(

    `CREATE TABLE IF NOT EXISTS PARTIDO(id INT PRIMARY KEY,nombre VARCHAR(100),siglas VARCHAR(20),fecha\_funda DATE);`

  );

  const result2 = await connection.query(

    `CREATE TABLE IF NOT EXISTS VOTO(id INT PRIMARY KEY,fecha\_hora DATETIME,dpi VARCHAR(13),mesa INT,FOREIGN KEY (dpi) REFERENCES CIUDADANO(dpi),FOREIGN KEY (mesa) REFERENCES MESA(id));`

  );

  const result3 = await connection.query(

    `CREATE TABLE IF NOT EXISTS CARGO(id INT PRIMARY KEY,cargo VARCHAR(50));`

  );

  const result4 = await connection.query(

    `CREATE TABLE IF NOT EXISTS CANDIDATO(id INT PRIMARY KEY,nombre VARCHAR(100),fecha\_nac DATE,partido INT, cargo INT,FOREIGN KEY (partido) REFERENCES PARTIDO(id),FOREIGN KEY (cargo) REFERENCES CARGO(id));`

  );

  const result5 = await connection.query(

    `CREATE TABLE IF NOT EXISTS DETALLE\_VOTO(candidato INT,voto INT,FOREIGN KEY(candidato) REFERENCES CANDIDATO(id),FOREIGN KEY(voto) REFERENCES VOTO(id));`

  );

  res.send("Modelo creado");

};

exports.eliminarmodelo = async (req, res) => {

*// Eliminar las tablas del modelo*

  const result5 = await connection.query(`DROP TABLE IF EXISTS DETALLE\_VOTO;`);

  const result4 = await connection.query(`DROP TABLE IF EXISTS CANDIDATO;`);

  const result3 = await connection.query(`DROP TABLE IF EXISTS CARGO;`);

  const result2 = await connection.query(`DROP TABLE IF EXISTS VOTO;`);

  const result6 = await connection.query(`DROP TABLE IF EXISTS MESA;`);

  const result = await connection.query(`DROP TABLE IF EXISTS CIUDADANO;`);

  const result1 = await connection.query(`DROP TABLE IF EXISTS DEPARTAMENTO;`);

  const result7 = await connection.query(`DROP TABLE IF EXISTS PARTIDO;`);

  res.send("Modelo eliminado");

};

exports.cargartabtemp = async (req, res) => {

*//////////////////////////////// CARGAR DATOS EN TABLAS TEMPORALES ////////////////////////////////*

*// Creación de las tablas temporales*

  await connection.query(

    `CREATE TEMPORARY TABLE IF NOT EXISTS TEMP(id\_voto int,id\_candidato int,dpi varchar(13),mesa int,fecha\_hora datetime);`

  );

  await connection.query(

    `CREATE TEMPORARY TABLE IF NOT EXISTS TEMP\_CIUDADANO(dpi varchar(13),nombre varchar(50),apellido varchar(50),direccion varchar(100),telefono varchar(15), edad int, genero varchar(10));`

  );

  await connection.query(

    `CREATE TEMPORARY TABLE IF NOT EXISTS TEMP\_DEPARTAMENTO(id int, nombre varchar(50));`

  );

  await connection.query(

    `CREATE TEMPORARY TABLE IF NOT EXISTS TEMP\_MESA(id int, departamento int);`

  );

  await connection.query(

    `CREATE TEMPORARY TABLE IF NOT EXISTS TEMP\_PARTIDO(id int, nombre varchar(100), siglas varchar(20), fecha\_funda date);`

  );

  await connection.query(

    `CREATE TEMPORARY TABLE IF NOT EXISTS TEMP\_CARGO(id int, cargo varchar(50));`

  );

  await connection.query(

    `CREATE TEMPORARY TABLE IF NOT EXISTS TEMP\_CANDIDATO(id int, nombre varchar(100), fecha\_nac date, partido int, cargo int);`

  );

  console.log("tablas temporales creadas");

*// Lectura del archivo votaciones.csv*

  let data = fs.readFileSync("src/csv/votaciones.csv", "utf8");

  let rows = data.split("\r\n").slice(1, data.length - 1);

*//Cargar datos en la tabla temp*

  for (const element of rows) {

    const row = element.split(";");

    try {

      await connection.query(`INSERT INTO TEMP VALUES(?,?,?,?,?);`, [

        row[0],

        row[1],

        row[2],

        row[3],

        row[4],

      ]);

    } catch (error) {

      console.log("Error: " + error.sql);

      console.log("Error: " + error.sqlMessage);

    }

  }

  console.log("votos cargados");

*// Lectura de los archivos ciudadanos.csv*

  data = fs.readFileSync("src/csv/ciudadanos.csv", "utf8");

  rows = data.split("\r\n").slice(1, data.length - 1);

*//Cargar datos en la tabla temp\_ciudadano*

  for (const element of rows) {

    const row = element.split(";");

    try {

      await connection.query(

        `INSERT INTO TEMP\_CIUDADANO VALUES(?,?,?,?,?,?,?);`,

        [row[0], row[1], row[2], row[3], row[4], row[5], row[6]]

      );

    } catch (error) {

      console.log("Error: " + error.sql);

      console.log("Error: " + error.sqlMessage);

    }

  }

  console.log("ciudadanos cargados");

*// Lectura de los archivos departamentos.csv*

  data = fs.readFileSync("src/csv/departamentos.csv", "utf8");

  rows = data.split("\r\n").slice(1, data.length - 1);

*//Cargar datos en la tabla temp\_departamento*

  for (const element of rows) {

    const row = element.split(";");

    try {

      await connection.query(`INSERT INTO TEMP\_DEPARTAMENTO VALUES(?,?);`, [

        row[0],

        row[1],

      ]);

    } catch (error) {

      console.log("Error: " + error.sql);

      console.log("Error: " + error.sqlMessage);

    }

  }

  console.log("departamentos cargados");

*// Lectura de los archivos mesas.csv*

  data = fs.readFileSync("src/csv/mesas.csv", "utf8");

  rows = data.split("\r\n").slice(1, data.length - 1);

*//Cargar datos en la tabla temp\_mesa*

  for (const element of rows) {

    const row = element.split(";");

    try {

      await connection.query(`INSERT INTO TEMP\_MESA VALUES(?,?);`, [

        row[0],

        row[1],

      ]);

    } catch (error) {

      console.log("Error: " + error.sql);

      console.log("Error: " + error.sqlMessage);

    }

  }

  console.log("mesas cargadas");

*// Lectura de los archivos partidos.csv*

  data = fs.readFileSync("src/csv/partidos.csv", "utf8");

  rows = data.split("\r\n").slice(1, data.length - 1);

*//Cargar datos en la tabla temp\_partido*

  for (const element of rows) {

    const row = element.split(";");

    try {

      await connection.query(`INSERT INTO TEMP\_PARTIDO VALUES(?,?,?,?);`, [

        row[0],

        row[1],

        row[2],

        row[3],

      ]);

    } catch (error) {

      console.log("Error: " + error.sql);

      console.log("Error: " + error.sqlMessage);

    }

  }

  console.log("partidos cargados");

*// Lectura de los archivos cargos.csv*

  data = fs.readFileSync("src/csv/cargos.csv", "utf8");

  rows = data.split("\r\n").slice(1, data.length - 1);

*//Cargar datos en la tabla temp\_cargo*

  for (const element of rows) {

    const row = element.split(";");

    try {

      await connection.query(`INSERT INTO TEMP\_CARGO VALUES(?,?);`, [

        row[0],

        row[1],

      ]);

    } catch (error) {

      console.log("Error: " + error.sql);

      console.log("Error: " + error.sqlMessage);

    }

  }

  console.log("cargos cargados");

*// Lectura de los archivos candidatos.csv*

  data = fs.readFileSync("src/csv/candidatos.csv", "utf8");

  rows = data.split("\r\n").slice(1, data.length - 1);

*//Cargar datos en la tabla temp\_candidato*

  for (const element of rows) {

    const row = element.split(",");

    try {

      await connection.query(`INSERT INTO TEMP\_CANDIDATO VALUES(?,?,?,?,?);`, [

        row[0],

        row[1],

        row[2],

        row[3],

        row[4],

      ]);

    } catch (error) {

      console.log("Error: " + error.sql);

      console.log("Error: " + error.sqlMessage);

    }

  }

  console.log("candidatos cargados");

*//////////////////////////////// FIN CARGAR DATOS EN TABLAS TEMPORALES ////////////////////////////////*

*//////////////////////////////// CARGAR DATOS MODELO ////////////////////////////////*

  let [result, error] = await connection.query(`SELECT \* FROM TEMP\_CIUDADANO;`);

*//Cargar datos en la tabla ciudadano*

  for (const element of result) {

    try {

      await connection.query(

        `INSERT INTO CIUDADANO VALUES("${element.dpi}","${element.nombre}","${element.apellido}","${element.direccion}","${element.telefono}",${element.edad},"${element.genero}");`

      );

    } catch (error) {

      console.log("Error: " + error.sql);

      console.log("Error: " + error.sqlMessage);

    }

  }

  console.log("ciudadanos cargados");

  [result, error] = await connection.query(`SELECT \* FROM TEMP\_DEPARTAMENTO;`);

*//Cargar datos en la tabla departamento*

  for (const element of result) {

    try {

      await connection.query(`INSERT INTO DEPARTAMENTO VALUES(?,?);`, [

        element.id,

        element.nombre,

      ]);

    } catch (error) {

      console.log("Error: " + error.sql);

      console.log("Error: " + error.sqlMessage);

    }

  }

  console.log("departamentos cargados");

  [result, error] = await connection.query(`SELECT \* FROM TEMP\_MESA;`);

*//Cargar datos en la tabla mesa*

  for (const element of result) {

    try {

      await connection.query(`INSERT INTO MESA VALUES(?,?);`, [

        element.id,

        element.departamento,

      ]);

    } catch (error) {

      console.log("Error: " + error.sql);

      console.log("Error: " + error.sqlMessage);

    }

  }

  console.log("mesas cargadas");

  [result, error] = await connection.query(

    `SELECT id\_voto,dpi,mesa,fecha\_hora FROM TEMP GROUP BY id\_voto,dpi,mesa,fecha\_hora;`

  );

*//Cargar datos en la tabla voto*

  for (const element of result) {

    try {

      await connection.query(`INSERT INTO VOTO VALUES(?,?,?,?);`, [

        element.id\_voto,

        element.fecha\_hora,

        element.dpi,

        element.mesa,

      ]);

    } catch (error) {

      console.log("Error: " + error.sql);

      console.log("Error: " + error.sqlMessage);

    }

  }

  console.log("votos cargados");

  [result, error] = await connection.query(`SELECT \* FROM TEMP\_PARTIDO;`);

*//Cargar datos en la tabla partido*

  for (const element of result) {

    try {

      await connection.query(`INSERT INTO PARTIDO VALUES(?,?,?,?);`, [

        element.id,

        element.nombre,

        element.siglas,

        element.fecha\_funda,

      ]);

    } catch (error) {

      console.log("Error: " + error.sql);

      console.log("Error: " + error.sqlMessage);

    }

  }

  console.log("partidos cargados");

  [result, error] = await connection.query(`SELECT \* FROM TEMP\_CARGO;`);

*//Cargar datos en la tabla cargo*

  for (const element of result) {

    try {

      await connection.query(`INSERT INTO CARGO VALUES(?,?);`, [

        element.id,

        element.cargo,

      ]);

    } catch (error) {

      console.log("Error: " + error.sql);

      console.log("Error: " + error.sqlMessage);

    }

  }

  console.log("cargos cargados");

  [result, error] = await connection.query(`SELECT \* FROM TEMP\_CANDIDATO;`);

*//Cargar datos en la tabla candidato*

  for (const element of result) {

    try {

      await connection.query(`INSERT INTO CANDIDATO VALUES(?,?,?,?,?);`, [

        element.id,

        element.nombre,

        element.fecha\_nac,

        element.partido,

        element.cargo,

      ]);

    } catch (error) {

      console.log("Error: " + error.sql);

      console.log("Error: " + error.sqlMessage);

    }

  }

  console.log("candidatos cargados");

  [result, error] = await connection.query(

    `SELECT DISTINCT id\_voto,id\_candidato FROM TEMP;`

  );

*//Cargar datos en la tabla detalle\_voto*

  for (const element of result) {

    try {

      await connection.query(`INSERT INTO DETALLE\_VOTO VALUES(?,?);`, [

        element.id\_candidato,

        element.id\_voto,

      ]);

    } catch (error) {

      console.log("Error: " + error.sql);

      console.log("Error: " + error.sqlMessage);

    }

  }

  console.log("detalle de votos cargados");

*//////////////////////////////// FIN CARGAR DATOS MODELO ////////////////////////////////*

*//////////////////////////////// ELIMINAR DATOS TABLAS TEMPORALES ////////////////////////////////*

  await connection.query(`DELETE FROM TEMP;`);

  await connection.query(`DELETE FROM TEMP\_CANDIDATO;`);

  await connection.query(`DELETE FROM TEMP\_CARGO;`);

  await connection.query(`DELETE FROM TEMP\_PARTIDO;`);

  await connection.query(`DELETE FROM TEMP\_MESA;`);

  await connection.query(`DELETE FROM TEMP\_DEPARTAMENTO;`);

  await connection.query(`DELETE FROM TEMP\_CIUDADANO;`);

*//////////////////////////////// FIN ELIMINAR DATOS TABLAS TEMPORALES ////////////////////////////////*

  res.send("MODELO CARGADO");

};

## QueryController.js

const { getConnection } = require("../database/database");

const { error } = require("console");

const connection = getConnection();

exports.consulta1 = async (req, res) => {

*// Obtener los presidentes y vicepresidentes por partido*

  const result = await connection.query(`SELECT

  P.nombre AS presidente,

  V.nombre AS vicepresidente,

  PARTIDO.nombre AS partido

FROM

  CANDIDATO AS P

INNER JOIN

  CANDIDATO AS V ON P.partido = V.partido

INNER JOIN

  PARTIDO ON P.partido = PARTIDO.id

WHERE

  P.cargo = 1

  AND V.cargo = 2;`);

  res.send(result[0]);

};

exports.consulta2 = async (req, res) => {

*// Obtener la cantidad de diputados por partido*

  const result = await connection.query(`SELECT

    Partido.nombre AS partido,

    SUM(CASE WHEN C.cargo = 3 THEN 1 ELSE 0 END) AS "Diputados congreso lista nacional",

    SUM(CASE WHEN C.cargo = 4 THEN 1 ELSE 0 END) AS "Diputados congreso distrito electoral",

    SUM(CASE WHEN C.cargo = 5 THEN 1 ELSE 0 END) AS "Diputados parlamento centroamericano",

    COUNT(C.id) AS "Cantidad de Candidatos"

FROM

    Candidato C

INNER JOIN

    Partido ON C.partido = Partido.id

WHERE

    C.cargo IN (3, 4, 5)

GROUP BY

    Partido.id, Partido.nombre;`);

  res.send(result[0]);

};

exports.consulta3 = async (req, res) => {

*// Obtener los alcaldes por partido*

  const result =

    await connection.query(`SELECT Partido.nombre as partido, C.nombre as alcalde

    FROM Candidato C

    INNER JOIN

        PARTIDO Partido ON C.partido = Partido.id

    WHERE

        C.cargo=6

    ;`);

  res.send(result[0]);

};

exports.consulta4 = async (req, res) => {

*// Obtener la cantidad de candidatos por partido*

  const result = await connection.query(`SELECT P.nombre as partido,

  SUM(CASE WHEN C.cargo = 1 THEN 1 ELSE 0 END) as presidente,

  SUM(CASE WHEN C.cargo = 2 THEN 1 ELSE 0 END) as vicepresidente,

  SUM(CASE WHEN C.cargo = 3 OR C.cargo = 4 OR C.cargo = 5 THEN 1 ELSE 0 END) as diputados,

  SUM(CASE WHEN C.cargo = 6 THEN 1 ELSE 0 END) as alcaldes

  FROM

    PARTIDO P

  INNER JOIN

    CANDIDATO C ON P.id = C.partido

  GROUP BY

    P.nombre, P.id;`);

  res.send(result[0]);

};

exports.consulta5 = async (req, res) => {

*// Obtener la cantidad de votos por departamento*

  const result =

    await connection.query(`SELECT D.nombre AS DEPARTAMENTO, COUNT(V.id) AS "CANTIDAD DE VOTOS" FROM DEPARTAMENTO D

    INNER JOIN MESA M ON M.departamento = D.id

    INNER JOIN VOTO V ON V.mesa = M.id

    GROUP BY D.nombre, D.id

    ;`);

  res.send(result[0]);

};

exports.consulta6 = async (req, res) => {

*// Obtener la cantidad de votos nulos*

  const result =

    await connection.query(`SELECT COUNT(DISTINCT DV.voto) as "VOTO NULOS" FROM DETALLE\_VOTO DV

    INNER JOIN VOTO V ON V.id = DV.voto

    WHERE DV.candidato = -1;`);

  res.send(result[0]);

};

exports.consulta7 = async (req, res) => {

*// Obtener los 10 rangos de edad con mas votos*

  const result =

    await connection.query(`SELECT C.edad, COUNT(C.dpi) as Cantidad FROM VOTO V

    INNER JOIN

    CIUDADANO C ON C.dpi = V.dpi

    GROUP BY

    C.edad

    ORDER BY

    COUNT(C.dpi)

    DESC LIMIT 10;`);

  res.send(result[0]);

};

exports.consulta8 = async (req, res) => {

*// Obtener los 10 presidentes y vicepresidentes con mas votos*

  const result = await connection.query(`SELECT

    P.nombre AS presidente,

    V.nombre AS vicepresidente,

  COUNT(P.id) AS "Cantidad Votos"

FROM

    CANDIDATO P

INNER JOIN

    CANDIDATO V ON P.partido = V.partido

INNER JOIN

  DETALLE\_VOTO DV ON DV.candidato = P.id

WHERE

    P.cargo = 1 AND V.cargo = 2

GROUP BY

  P.id, V.id, P.nombre, V.nombre

ORDER BY COUNT(P.id) DESC LIMIT 10;`);

  res.send(result[0]);

};

exports.consulta9 = async (req, res) => {

*// Obtener los 5 departamentos con mas votantes*

  const result =

    await connection.query(`SELECT V.mesa, D.nombre, COUNT(V.mesa) AS "Cantidad votantes"

  FROM VOTO V

  INNER JOIN

    MESA M ON V.mesa = M.id

  INNER JOIN

    DEPARTAMENTO D ON M.departamento = D.id

  group by

    M.id

  ORDER BY COUNT(V.mesa) Desc LIMIT 5;`);

  res.send(result[0]);

};

exports.consulta10 = async (req, res) => {

*// Obtener las 5 horas con mas votos*

  const result = await connection.query(

    `SELECT TIME(V.fecha\_hora) AS HORA, COUNT(V.fecha\_hora) AS CANTIDAD FROM VOTO V GROUP BY V.fecha\_hora ORDER BY COUNT(V.fecha\_hora) DESC LIMIT 5;`

  );

  res.send(result[0]);

};

exports.consulta11 = async (req, res) => {

*// Obtener la cantidad de votos por genero*

  const result = await connection.query(

    `SELECT C.genero, COUNT(C.genero) AS "Cantidad de votos" FROM CIUDADANO C

    INNER JOIN

      VOTO V ON C.dpi = V.dpi

    GROUP BY

      C.genero;`

  );

  res.send(result[0]);

};