

Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Sistemas de Bases de Datos 1 “A”



Proyecto 1

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:

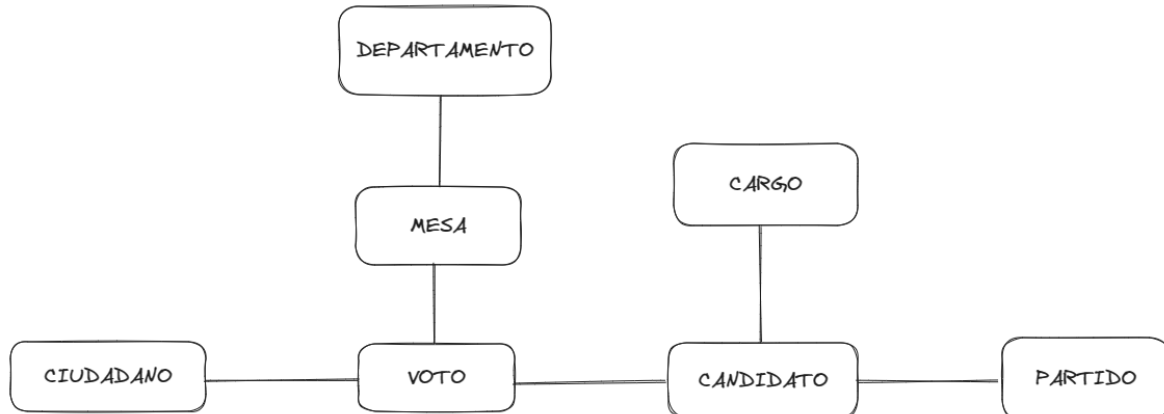
5	1069	8.3771E+12	66	25/06/2023 11:38		
ciudadanos	votaciones	partidos	mesas	departamentos	cargos	candidatos

Ahora bien, para las relaciones que podrían estar sucediendo entre las entidades en un principio solo reconoceremos dos: uno a muchos (una línea) y muchos a muchos (doble línea). 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:

id_mesa	id_departamento
1	15
2	7

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:

DPI	Nombre	Apellido	Direccion	Telefono	Edad	Genero
1979280000000	Vivian	Schultz	1511 Pooh Bear Lane	4608499546	55	F
3144520000000	Natisha	Flores	3663 McDowell Street	5119315633	80	F

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:

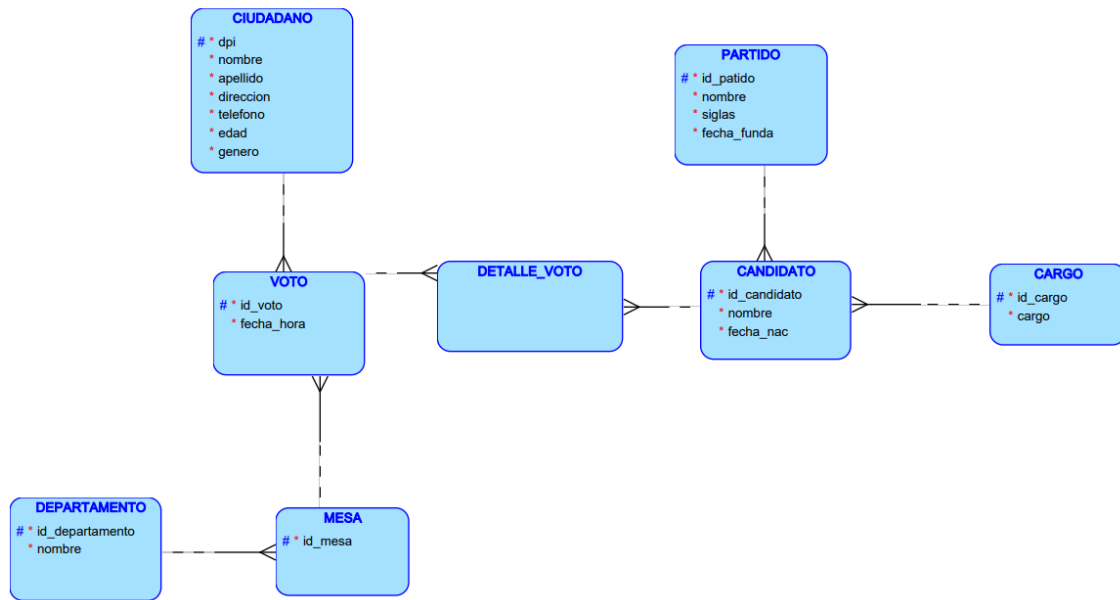
id_voto	id_candidato	dpi_ciudadano	mesa_id	fecha_hora
1	6	1.97928E+12	29	25/06/2023 09:54
1	1068	1.97928E+12	29	25/06/2023 09:54
1	253	1.97928E+12	29	25/06/2023 09:54
1	615	1.97928E+12	29	25/06/2023 09:54
1	784	1.97928E+12	29	25/06/2023 09:54
2	16	3.14452E+12	30	25/06/2023 13:08
2	1064	3.14452E+12	30	25/06/2023 13:08
2	254	3.14452E+12	30	25/06/2023 13:08
2	510	3.14452E+12	30	25/06/2023 13:08
2	825	3.14452E+12	30	25/06/2023 13:08
3	1	8.72809E+12	61	25/06/2023 18:45
3	1069	8.72809E+12	61	25/06/2023 18:45
3	65	8.72809E+12	61	25/06/2023 18:45

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 repetentes se agruparan y trasladan a otra tabla que denominamos detalle votos, junto con el id del voto al que pertenece.

id_voto	dpi_ciudadano	mesa_id	fecha_hora
1	1.9793E+12	29	25/06/2023 09:54
2	3.1445E+12	30	25/06/2023 13:08
3	8.7281E+12	61	25/06/2023 18:45
4	7.16E+12	5	25/06/2023 15:25
5	8.3771E+12	66	25/06/2023 11:38
6	7.7303E+12	56	25/06/2023 08:18
7	5.915E+12	100	25/06/2023 18:08
8	7.664E+12	34	25/06/2023 16:40
9	7.1721E+12	73	25/06/2023 08:58
10	5.4867E+12	39	25/06/2023 14:44

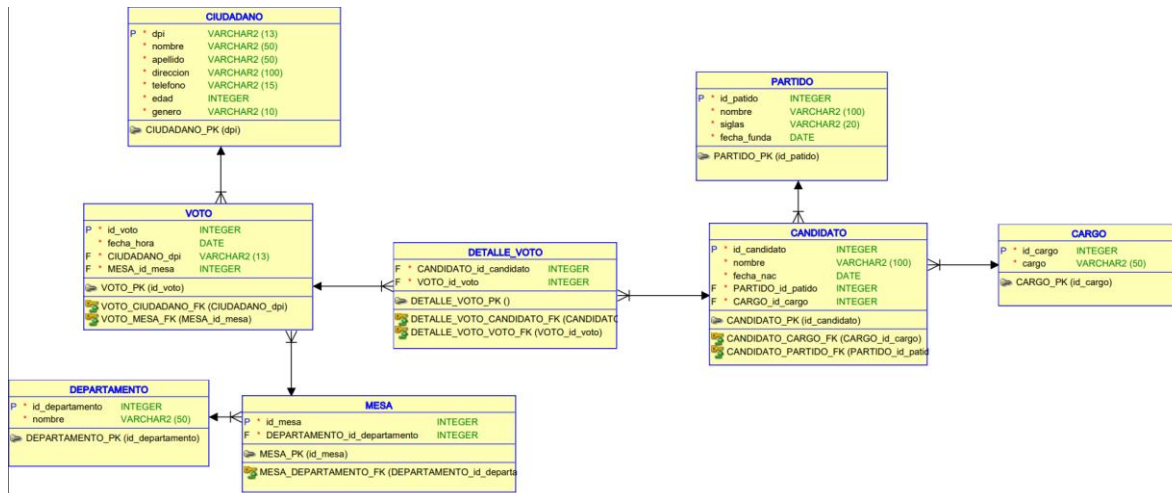
id_voto	id_candidato
1	6
1	1068
1	253
1	615
1	784
2	16
2	1064
2	254
2	510
2	825
3	1
3	1069
3	65
3	608
3	1003

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”.

CIUDADANO		
P *	dpi	VARCHAR2 (13)
*	nombre	VARCHAR2 (50)
*	apellido	VARCHAR2 (50)
*	direccion	VARCHAR2 (100)
*	telefono	VARCHAR2 (15)
*	edad	INTEGER
*	genero	VARCHAR2 (10)
CIUDADANO_PK (dpi)		

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.

PARTIDO		
P *	id_partido	INTEGER
*	nombre	VARCHAR2 (100)
*	siglas	VARCHAR2 (20)
*	fecha_funda	DATE
PARTIDO_PK (id_partido)		

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.

DEPARTAMENTO		
P *	id_departamento	INTEGER
*	nombre	VARCHAR2 (50)
DEPARTAMENTO_PK (id_departamen		

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.

CARGO		
P *	id_cargo	INTEGER
*	cargo	VARCHAR2 (50)
CARGO_PK (id_cargo)		

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.

MESA		
P *	id_mesa	INTEGER
F *	DEPARTAMENTO_id_departamento	INTEGER
MESA_PK (id_mesa)		
MESA_DEPARTAMENTO_FK (DEPARTAMENTO_id_departan		

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.

CANDIDATO		
P *	id_candidato	INTEGER
*	nombre	VARCHAR2 (100)
*	fecha_nac	DATE
F *	PARTIDO_id_partido	INTEGER
F *	CARGO_id_cargo	INTEGER
CANDIDATO_PK (id_candidato)		
CANDIDATO_CARGO_FK (CARGO_id_cargo)		
CANDIDATO_PARTIDO_FK (PARTIDO_id_partido)		

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.

VOTO		
P *	id_voto	INTEGER
*	fecha_hora	DATE
F *	CIUDADANO_dpi	VARCHAR2
F *	MESA_id_mesa	INTEGER
VOTO_PK (id_voto)		
VOTO_CIUDADANO_FK (CIUDADANO_dpi)		
VOTO_MESA_FK (MESA_id_mesa)		

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.

DETALLE_VOTO		
F *	CANDIDATO_id_candidato	INTEGER
F *	VOTO_id_voto	INTEGER
P *	id	INTEGER
DETALLE_VOTO_PK (id)		
DETALLE_VOTO_CANDIDATO_FK (CANDIDATO_id_candidato)		
DETALLE_VOTO_VOTO_FK (VOTO_id_voto)		

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 vicepresidente.

```
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.



- candidatos.csv
- cargos.csv
- ciudadanos.csv
- departamentos.csv
- detalle_normalizado.csv
- mesas.csv
- partidos.csv
- votaciones.csv
- votos_normalizado.csv

Conexión a la base de datos

Importación de las librerías

```
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.creamodelo);
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_CIUADADANO 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]);
};

```