UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN
INGENIERÍA EN COMPUTACIÓN

Proyecto de Base de Datos.

Base de Datos relacional de candidatos.

Profesor:

Dr. Omar Mendoza Gonzalez .

Alumnos:

Jimenez Jimenez Roberto Salvador

Lara Martinez Christian Gael.

Ramírez Cortés Saúl Isaac.

Santos Sanchez Roberto Carlos.

Grupo:

2410

Semestre:

2204-2

b) Modelo Conceptual.

1. Partido Político.
Descripción:
Esta tabla almacena información sobre los partidos políticos, incluyendo sus nombres, abreviaturas, y detalles de contacto como teléfono y correo electrónico.

Atributos:
id_partido: INT, clave primaria, auto-incremental.
nombre: VARCHAR(200), no nulo, único.
abreviatura: VARCHAR(10).
telefono: VARCHAR(15), no nulo, único.
email: VARCHAR(100), no nulo, único.

Relaciones:
Sin relaciones directas a otras tablas.

2. Coalición
Descripción:
Una coalición es una agrupación de partidos políticos que se unen para trabajar juntos en una elección.

Atributos:
id_coalicion: INT, clave primaria, auto-incremental.
id_partido: INT.
id_partido2: INT.
id_partido3: INT.
nombre: VARCHAR(30), no nulo, único.

Relaciones:
Una coalición puede incluir entre 1 y 3 partidos políticos.
id_partido referencia a partido_politico(id_partido).
id_partido2 referencia a partido_politico(id_partido).
id_partido3 referencia a partido_politico(id_partido).

3. Militancia
Descripción:
Esta tabla define los diferentes tipos de militancia, es decir, los grados de afiliación o membresía en un partido político.

Atributos:
id_militancia: INT, clave primaria, auto-incremental.
nombre: VARCHAR(30), no nulo, único.

Relaciones:
Sin relaciones directas a otras tablas.

4. Cargo
Descripción:
Aquí se almacenan los diferentes cargos públicos que los candidatos pueden ocupar.

Atributos:
id_cargo: INT, clave primaria, auto-incremental.
nombre_cargo: VARCHAR(50).

Relaciones:
Un cargo puede ser ocupado por 0 o muchos candidatos.
Relación de uno a muchos con la tabla candidato.

5. Historial
Descripción:
Registra los intentos anteriores de los candidatos para ocupar cargos públicos, incluyendo los resultados y fechas de las elecciones.

Atributos:
id_historial: INT, clave primaria, auto-incremental.
cargo_solicitado: VARCHAR(70), no nulo.
resultado: VARCHAR(30), no nulo.
fecha_eleccion: DATE, no nulo.

Relaciones:
Un historial puede estar asociado a 0 o muchos candidatos.
Relación de uno a muchos con la tabla candidato.

6. Documentación
Descripción:
Contiene la documentación requerida para que un candidato se postule a un cargo público, incluyendo verificaciones y declaraciones de interés.

Atributos:
id_documentacion: INT, clave primaria, auto-incremental.
documento: TEXT, no nulo.
verificacion_eligibilidad: VARCHAR(50), no nulo.
declaracion_interes: TEXT, no nulo.

Relaciones:
Un documento puede estar asociado a 0 o muchos candidatos.
Relación de uno a muchos con la tabla candidato.

7. Estados
Descripción:
Esta tabla guarda información sobre los estados de un país, incluyendo el nombre y estadísticas de población y electores.

Atributos:
id_estado: INT, clave primaria, auto-incremental.
nombre: VARCHAR(30), no nulo, único.
numero_habitantes: INT.
numero_electores: INT.

Relaciones:
Un estado puede tener muchos distritos y municipios.
Relación de uno a muchos con las tablas distritos, municipios y demarcacion.

8. Distritos
Descripción:
Define las divisiones electorales dentro de un estado, incluyendo información demográfica y su asociación a un estado específico.

Atributos:
id_distrito: INT, clave primaria, auto-incremental.
nombre: VARCHAR(70), no nulo, único.
num_habitantes: INT.
num_electores: INT.
id_estado: INT.

Relaciones:
Un distrito pertenece a un estado.
id_estado referencia a estados(id_estado).

9. Municipios
Descripción:
Describe los municipios dentro de los estados, con detalles similares a los distritos, pero a un nivel administrativo diferente.

Atributos:
id_municipio: INT, clave primaria, auto-incremental.
nombre: VARCHAR(70), no nulo, único.
num_habitantes: INT.
num_electores: INT.
id_estado: INT.

Relaciones:
Un municipio pertenece a un estado.
id_estado referencia a estados(id_estado).

10. Demarcación

Descripción:
Define las áreas geográficas específicas para propósitos electorales, asociadas a un estado o municipio.

Atributos:
id_demarcacion: INT, clave primaria, auto-incremental.
id_estado: INT.
id_municipio_o_estado: INT.

Relaciones:
Una demarcación pertenece a un estado o a un municipio.
id_estado referencia a estados(id_estado).

11. candidato

Descripción:
Almacena la información personal y profesional de los candidatos que se postulan para cargos públicos.

Atributos:
id_candidato: INT, clave primaria, auto-incremental.
rfc: VARCHAR(13), no nulo, único.
nombre: VARCHAR(50), no nulo.
ap_paterno: VARCHAR(50), no nulo.
ap_materno: VARCHAR(50), no nulo.
fnac: DATE, no nulo.
direccion: VARCHAR(150), no nulo.
telefono: VARCHAR(15), no nulo, único.
email: VARCHAR(100), no nulo, único.
propuestas: TEXT, no nulo.
finscrip: DATE, no nulo.
id_partido: INT.
id_coalicion: INT.
id_militancia: INT.
id_cargo: INT.
id_demarcacion: INT.
id_historial: INT.
id_documentacion: INT.

Relaciones:
Un candidato puede pertenecer a un partido político (id_partido referencia a partido_politico(id_partido)).
Un candidato puede pertenecer a una coalición (id_coalicion referencia a coalicion(id_coalicion)).
Un candidato puede tener una militancia (id_militancia referencia a militancia(id_militancia)).
Un candidato puede tener un cargo (id_cargo referencia a cargo(id_cargo)).

Un candidato puede estar en una demarcación (id_demarcacion referencia a demarcacion(id_demarcacion)).
Un candidato puede tener un historial (id_historial referencia a historial(id_historial)).
Un candidato puede tener documentación (id_documentacion referencia a documentacion(id_documentacion)).

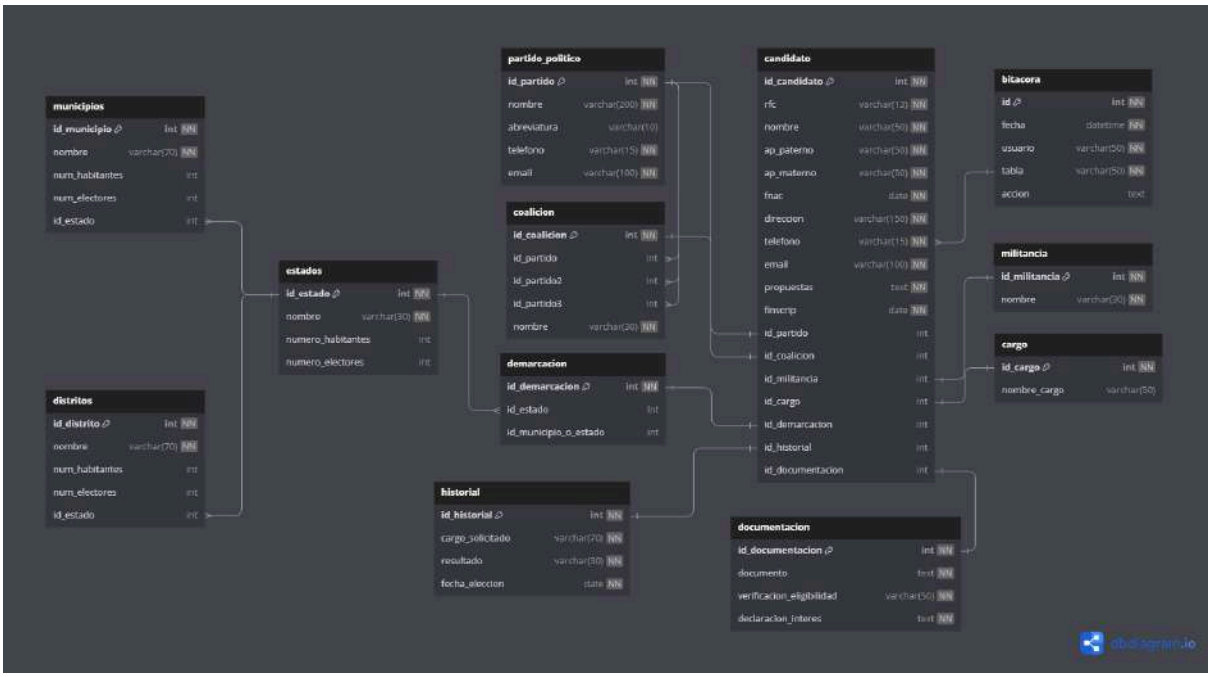
12. Bitácora

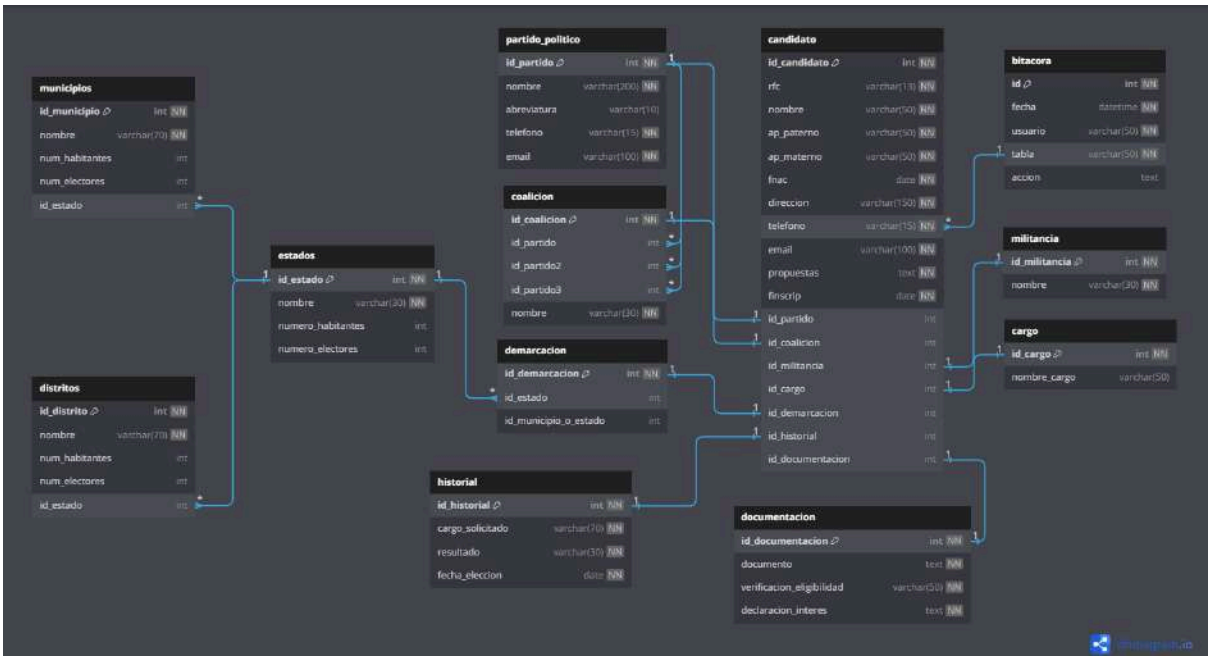
Descripción:
Registra las acciones realizadas en la base de datos para auditoría, como quién hizo qué cambios y cuándo.

Atributos:
id: INT, clave primaria, auto-incremental.
fecha: DATETIME, no nulo.
usuario: VARCHAR(50), no nulo.
tabla: VARCHAR(50), no nulo.
accion: TEXT.

Relaciones:
Sin relaciones directas a otras tablas.

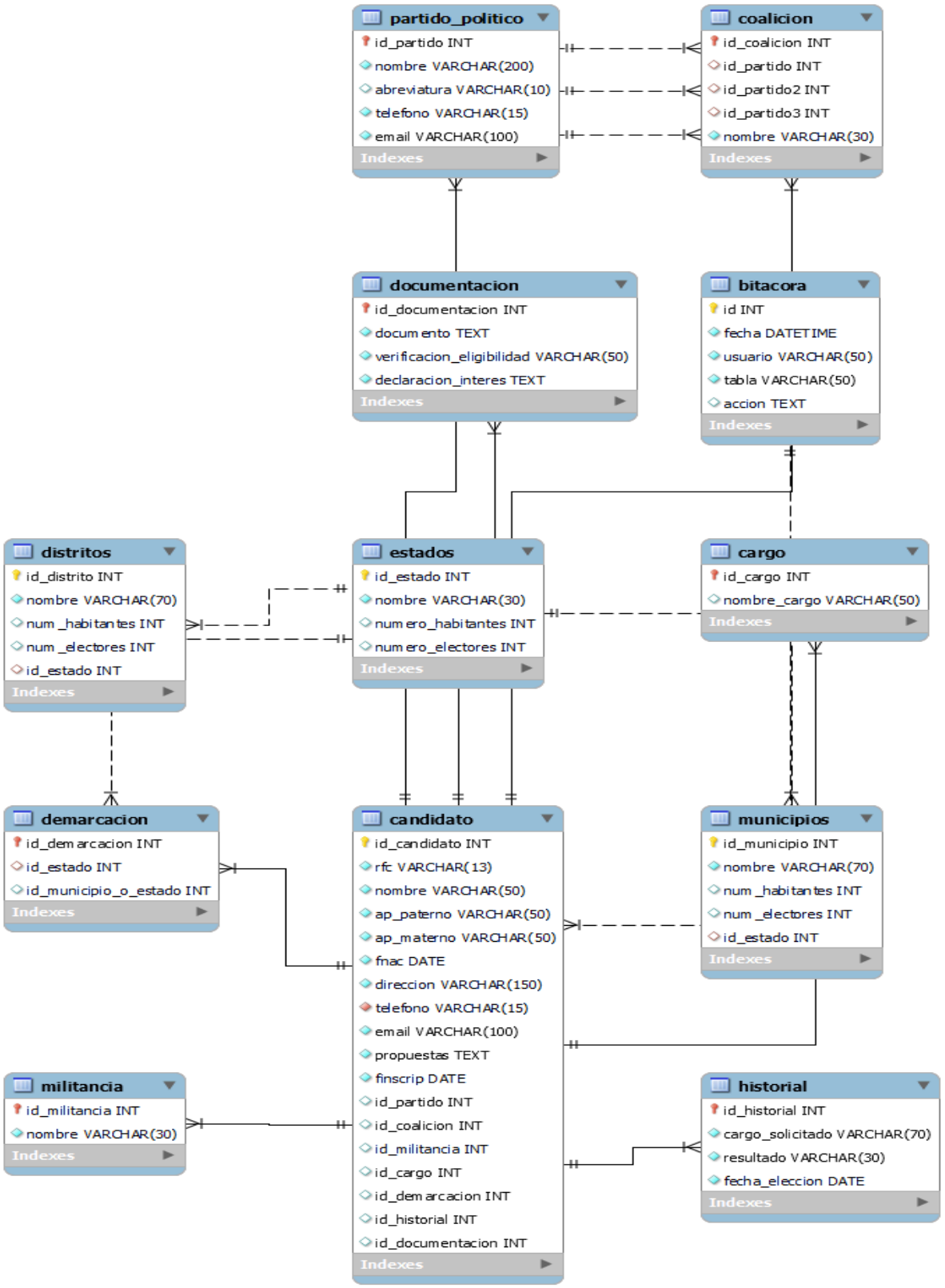
i) Modelo relacional.





Enlace a dbdiagram: https://dbdiagram.io/d/BD-project-664e80acf84ecd1d22e75874

d) Diagrama Entidad Relación.



e) Documentación:

```
1  create database Candidatos;
2  use Candidatos;
```

CREATE DATABASE Candidatos: Esta instrucción crea una nueva base de datos llamada "Candidatos". La base de datos es donde se almacenarán todas las tablas y datos relacionados con los candidatos y las elecciones.

USE Candidatos: Esta instrucción selecciona la base de datos "Candidatos" como la base de datos activa. Esto significa que todas las consultas y operaciones posteriores se aplicarán a esta base de datos específica hasta que se cambie con una instrucción USE diferente.

```
4  create table partido_politico(
5  id_partido int not null primary key auto_increment,
6  nombre varchar(200) not null unique,
7  abreviatura varchar(10),
8  telefono varchar(15) not null unique,
9  email varchar(100) not null unique
10 );
```

Crea una tabla llamada partido_politico con las siguientes columnas:
id_partido: Identificador único del partido (entero, autoincremental).
nombre: Nombre del partido (cadena de caracteres, máximo 200 caracteres, único, no nulo).
abreviatura: Abreviatura del partido (cadena de caracteres, máximo 10 caracteres).
telefono: Número de teléfono del partido (cadena de caracteres, máximo 15 caracteres, único, no nulo).
email: Correo electrónico del partido (cadena de caracteres, máximo 100 caracteres, único, no nulo).

```
12 create table coalicion(
13 id_coalicion int not null primary key auto_increment,
14 id_partido int ,
15 id_partido2 int,
16 id_partido3 int,
17 nombre varchar(30) not null unique,
18
19 foreign key (id_partido) references partido_politico(id_partido)  on delete cascade on update cascade,
20 FOREIGN KEY (id_partido2) REFERENCES partido_politico(id_partido) ON DELETE CASCADE ON UPDATE CASCADE,
21 FOREIGN KEY (id_partido3) REFERENCES partido_politico(id_partido) ON DELETE CASCADE ON UPDATE CASCADE
22 );
```

Crea una tabla llamada coalicion con las siguientes columnas:
id_coalicion: Identificador único de la coalición (entero, autoincremental).
id_partido, id_partido2, id_partido3: Identificadores de los partidos políticos que forman parte de la coalición (enteros).
nombre: Nombre de la coalición (cadena de caracteres, máximo 30 caracteres, único, no nulo).
Además, establece las siguientes restricciones de clave externa (foreign key constraints):
id_partido: Se referencia a la columna id_partido de la tabla partido_politico.
id_partido2: Se referencia a la columna id_partido de la tabla partido_politico.
id_partido3: Se referencia a la columna id_partido de la tabla partido_politico.
Estas restricciones aseguran que los valores en las columnas id_partido, id_partido2 y id_partido3 de la tabla coalicion existan en la columna id_partido de la tabla partido_politico, y especifican que si se actualizan o eliminan los valores en la tabla partido_politico, también se actualicen o eliminen en la tabla coalicion (acción en cascada).

```
24  create table militancia(
25      id_militancia int not null primary key auto_increment,
26      nombre varchar(30) not null unique
27  );
```

Crea una tabla llamada militancia con las siguientes columnas:
id_militancia: Identificador único de la militancia (entero, autoincremental).
nombre: Nombre de la militancia (cadena de caracteres, máximo 30 caracteres, único, no nulo).

```
29  create table cargo(
30      id_cargo int not null primary key auto_increment,
31      nombre_cargo varchar(50));
```

Crea una tabla llamada cargo con las siguientes columnas:
id_cargo: Identificador único del cargo (entero, autoincremental).
nombre_cargo: Nombre del cargo (cadena de caracteres, máximo 50 caracteres).

```sql
create table historial(
id_historial int not null primary key auto_increment,
cargo_solicitado varchar(70) not null,
resultado varchar(30) not null,
fecha_eleccion date not null
);
```

Esta sentencia crea una tabla llamada historial.
La tabla tiene columnas para el ID del historial (autonumérico y clave primaria), el cargo solicitado en la elección, el resultado de la elección y la fecha de la elección.
Los campos cargo_solicitado, resultado y fecha_eleccion son obligatorios (no nulos).
El id_historial se genera automáticamente y actúa como clave primaria única para cada registro en la tabla.

```sql
create table documentacion(
id_documentacion int not null primary key auto_increment,
documento text not null,
verificacion_eligibilidad varchar(50) not null,
declaracion_interes text not null
);
```

Crea la tabla documentacion, la tabla documentacion tiene columnas para el ID de la documentación (autonumérico y clave primaria), el tipo de documento, la verificación de elegibilidad y la declaración de intereses del candidato.
El id_documentacion se genera automáticamente y actúa como clave primaria única para cada registro en la tabla.
Los campos documento, verificacion_eligibilidad y declaracion_interes son obligatorios (no nulos).

```sql
create table distritos(
id_distrito int not null primary key auto_increment,
nombre varchar(70) not null unique,
num_habitantes int ,
num_electores int,
id_estado int,

FOREIGN KEY (id_estado) REFERENCES estados(id_estado) ON DELETE CASCADE ON UPDATE CASCADE
);
```

Crea la tabla ‘distritos’, la tabla distritos tiene columnas para el ID del distrito (autonumérico y clave primaria), el nombre del distrito (único), el número de habitantes, el número de electores y el ID del estado al que pertenece.

Se establece una restricción de clave externa (FOREIGN KEY) en la columna id_estado, que referencia la tabla estados y se vincula con la columna id_estado de esa tabla.
Se especifica que si se elimina o actualiza un estado en la tabla estados, estas acciones se propagarán a la tabla distritos debido a la cláusula ON DELETE CASCADE ON UPDATE CASCADE.

```
57  create table municipios(
58  id_municipio int not null primary key auto_increment,
59  nombre varchar(70) not null unique,
60  num_habitantes int,
61  num_electores int,
62  id_estado int,
63
64  FOREIGN KEY (id_estado) REFERENCES estados(id_estado) ON DELETE CASCADE ON UPDATE CASCADE
65  );
```

Crea la tabla municipios, la tabla municipios tiene columnas para el ID del municipio (autonumérico y clave primaria), el nombre del municipio (único), el número de habitantes, el número de electores y el ID del estado al que pertenece.
Se establece una restricción de clave externa (FOREIGN KEY) en la columna id_estado, que referencia la tabla estados y se vincula con la columna id_estado de esa tabla.
Se especifica que si se elimina o actualiza un estado en la tabla estados, estas acciones se propagarán a la tabla municipios debido a la cláusula ON DELETE CASCADE ON UPDATE CASCADE.

```
67  create table estados(
68  id_estado int not null primary key auto_increment,
69  nombre varchar(30) not null unique,
70  numero_habitantes int,
71  numero_electores int
72  );
```

crea la tabla estados, La tabla estados tiene columnas para el ID del estado (autonumérico y clave primaria), el nombre del estado (único), el número de habitantes y el número de electores.
La columna id_estado se define como clave primaria y se autonumera automáticamente.
La restricción UNIQUE en la columna nombre asegura que no haya dos estados con el mismo nombre en la tabla.
No se establecen restricciones de clave externa en esta tabla, ya que no parece haber relaciones de este tipo especificadas en la definición.

```sql
create table demarcacion(
id_demarcacion int not null primary key auto_increment,
id_estado int,
id_municipio_o_estado int,

FOREIGN KEY (id_estado) REFERENCES estados(id_estado) ON DELETE CASCADE ON UPDATE CASCADE
);
```

Crea la tabla demarcacion, la tabla demarcacion tiene columnas para el ID de la demarcación (autonumérico y clave primaria), el id del estado al que pertenece y el id del municipio o estado que constituye la demarcación.
Se establece una restricción de clave externa en la columna id_estado que hace referencia al id del estado en la tabla estados. Esta restricción especifica que si se elimina o actualiza un estado en la tabla estados, también se eliminarán o actualizarán automáticamente las filas correspondientes en la tabla demarcacion.

```sql
create table candidato(
id_candidato int not null primary key auto_increment,
rfc varchar(13) not null unique,
nombre varchar(50) not null,
ap_paterno varchar(50) not null,
ap_materno varchar(50) not null,
fnac date not null,
direccion varchar(150) not null,
telefono varchar(15)  not null unique,
email varchar(100) not null unique,
propuestas text not null,
finscrip date not null,
id_partido int,
id_coalicion int,
id_militancia int,
id_cargo int,
id_demarcacion int,
id_historial int,
id_documentacion int,

UNIQUE KEY uk_nombre (nombre, ap_paterno, ap_materno),
FOREIGN KEY (id_partido) REFERENCES partido_politico(id_partido) ON DELETE SET NULL ON UPDATE CASCADE,
FOREIGN KEY (id_coalicion) REFERENCES coalicion(id_coalicion) ON DELETE SET NULL ON UPDATE CASCADE,
FOREIGN KEY (id_militancia) REFERENCES militancia(id_militancia) ON DELETE SET NULL ON UPDATE CASCADE,
FOREIGN KEY (id_cargo) REFERENCES cargo(id_cargo) ON DELETE SET NULL ON UPDATE CASCADE,
FOREIGN KEY (id_demarcacion) REFERENCES demarcacion(id_demarcacion) ON DELETE SET NULL ON UPDATE CASCADE,
FOREIGN KEY (id_historial) REFERENCES historial(id_historial) ON DELETE SET NULL ON UPDATE CASCADE,
FOREIGN KEY (id_documentacion) REFERENCES documentacion(id_documentacion) ON DELETE CASCADE ON UPDATE CASCADE
);
```

Crea la tabla candidato, la tabla candidato tiene columnas para el ID del candidato (autonumérico y clave primaria), RFC, nombre, apellidos, fecha de nacimiento, dirección, teléfono, correo electrónico, propuestas, fecha de inscripción y varias claves externas que hacen referencia a otras tablas.
Se establecen restricciones de clave externa para garantizar la integridad referencial entre las tablas relacionadas, especificando las acciones a realizar en caso de eliminación o actualización (como SET NULL, CASCADE, etc.).

Se define una clave única compuesta por el nombre y los apellidos para evitar la duplicación de registros de candidatos con el mismo nombre completo.

```sql
create table bitacora(
id int not null auto_increment primary key,
fecha datetime not null,
usuario varchar(50) not null,
tabla varchar(50) not null,
accion text null
);
```

Crea la tabla bitacora, la tabla bitacora tiene las siguientes columnas:
id: Un identificador único para cada registro en la tabla, que se incrementa automáticamente.
fecha: La fecha y hora en que se realizó la acción, almacenada en el formato de fecha y hora (DATETIME).
usuario: El nombre del usuario que realizó la acción.
tabla: El nombre de la tabla en la que se realizó la acción.
accion: Una descripción de la acción realizada, que puede ser nula (NULL).
Esta tabla sirve para mantener un registro de las operaciones realizadas en la base de datos, lo que puede ser útil para auditar cambios, rastrear actividades y solucionar problemas. Cada vez que se realiza una acción importante en la base de datos, como agregar, modificar o eliminar registros, se registra en esta tabla junto con información relevante como la fecha, el usuario y la tabla afectada.

```sql
-- TRIGGERS

-- verifica que el no haya un duplicado (before insert)
DELIMITER //
drop trigger if exists bi_candidato //
CREATE TRIGGER bi_candidato
BEFORE INSERT ON candidato
FOR EACH ROW
BEGIN
    DECLARE candidato_existente INT;
    SELECT COUNT(*) INTO candidato_existente
    FROM candidato
    WHERE nombre = NEW.nombre
      AND ap_paterno = NEW.ap_paterno
      AND ap_materno = NEW.ap_materno;
    IF candidato_existente > 0 THEN
        INSERT INTO bitacora VALUES (null, sysdate(), user(),'CANDIDATO', CONCAT('Intento de inserción de candidato duplicado: ',
        NEW.nombre, ' ', NEW.ap_paterno, ' ', NEW.ap_materno));
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error: El candidato ya existe.';
    END IF;
END//
DELIMITER ;
```

Se define un trigger bi_candidato que se activa antes de insertar un nuevo registro en la tabla candidato.
Su propósito es verificar si hay un candidato duplicado antes de realizar la inserción.
DROP TRIGGER IF EXISTS bi_candidato: Elimina el trigger bi_candidato si ya existe para evitar conflictos al crear uno nuevo.
CREATE TRIGGER bi_candidato: Define un nuevo trigger llamado bi_candidato.
BEFORE INSERT ON candidato: Especifica que el trigger se activa antes de insertar un nuevo registro en la tabla candidato.
FOR EACH ROW: Indica que el trigger se ejecutará una vez por cada fila afectada por la operación de inserción.
El bloque BEGIN END contiene la lógica del trigger.
DECLARE candidato_existente INT: Declara una variable llamada candidato_existente que se utilizará para almacenar el número de candidatos duplicados encontrados.
SELECT COUNT(*) INTO candidato_existente: Realiza una consulta para contar el número de registros en la tabla candidato que coinciden con los valores del nuevo candidato (NEW.nombre, NEW.ap_paterno, NEW.ap_materno).
IF candidato_existente > 0 THEN ... END IF;: Verifica si se encontraron candidatos duplicados.
INSERT INTO bitacora: Registra en la tabla bitacora un mensaje indicando el intento de inserción de un candidato duplicado.
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error: El candidato ya existe.' : Lanza una señal de error para detener la operación de inserción si se encuentra un candidato duplicado.

```
146   -- muestra que candidato se agrego (after insert)
147   DELIMITER //
148   DROP TRIGGER IF EXISTS ai_candidato //
149   CREATE TRIGGER ai_candidato
150   AFTER INSERT ON candidato
151   FOR EACH ROW
152   BEGIN
153       INSERT INTO bitacora VALUES (null, sysdate(), user(),'CANDIDATO', CONCAT('Se agregó un nuevo candidato: ',
154       NEW.nombre, ' ', NEW.ap_paterno, ' ', NEW.ap_materno));
155   END//
156   DELIMITER ;
```

Se define un trigger ai_candidato que se activa después de insertar un nuevo registro en la tabla candidato.
Su propósito es registrar en la bitácora la información sobre el candidato que se acaba de agregar.
DROP TRIGGER IF EXISTS ai_candidato: Elimina el trigger ai_candidato si ya existe para evitar conflictos al crear uno nuevo.
CREATE TRIGGER ai_candidato: Define un nuevo trigger llamado ai_candidato.

AFTER INSERT ON candidato: Especifica que el trigger se activa después de insertar un nuevo registro en la tabla candidato.
FOR EACH ROW: Indica que el trigger se ejecutará una vez por cada fila afectada por la operación de inserción.
El bloque BEGIN ... END contiene la lógica del trigger.
INSERT INTO bitacora ...: Registra en la tabla bitacora un mensaje indicando que se agregó un nuevo candidato, junto con su nombre completo (NEW.nombre, NEW.ap_paterno, NEW.ap_materno).

```
158   -- muestra despues de borrar (after delete)
159   DELIMITER //
160   DROP trigger IF exists ad_candidato //
161    create trigger ad_candidato
162    after delete on candidato
163    for each row
164   begin
165      insert into bitacora values(null, sysdate(), user(), 'CANDIDATOS',
166      json_object('accion', 'ELIMINACION', 'id_candidato', old.id_candidato,'rfc', old.rfc,
167          'nombre', old.nombre, 'ap_paterno', old.ap_paterno,'ap_materno', old.ap_materno, 'fecha de nacimiento',
168          old.fenac, 'telefono', old.telefono,
169          'propuestas', old.propuestas, 'fecha de inscripción', old.finscrip,
170          'direccion', old.direccion, 'email', old.email, 'id_partido', old.id_partido,
171          'id_coalicion', old.id_coalicion, 'id_militancia', old.id_militancia,
172          'id_cargo', old.id_cargo, 'id_demarcacion', old.id_demarcacion,
173          'id_historial', old.id_historial, 'id_documentacion', old.id_documentacion
174      ));
175   END //
176   DELIMITER ;
```

define un trigger ad_candidato que se activa después de eliminar un registro de la tabla candidato. Su función es registrar en la bitácora la información del candidato que ha sido eliminado.
DROP TRIGGER IF EXISTS ad_candidato: Elimina el trigger ad_candidato si ya existe para evitar conflictos al crear uno nuevo.
CREATE TRIGGER ad_candidato: Define un nuevo trigger llamado ad_candidato.
AFTER DELETE ON candidato: Especifica que el trigger se activa después de eliminar un registro de la tabla candidato.
FOR EACH ROW: Indica que el trigger se ejecutará una vez por cada fila afectada por la operación de eliminación.
El bloque BEGIN ... END contiene la lógica del trigger.
INSERT INTO bitacora ...: Registra en la tabla bitacora un mensaje que contiene la información del candidato que ha sido eliminado, utilizando la función JSON_OBJECT para estructurar los datos en formato JSON.

```
178     -- verifica que no tenga mas de un partido
179    DELIMITER //
180    DROP TRIGGER IF EXISTS bi_maspartidos //
181    CREATE TRIGGER bi_partidos
182    BEFORE INSERT ON candidato
183    FOR EACH ROW
184    BEGIN
185        DECLARE partido_existente INT;
186        SELECT COUNT(*) INTO partido_existente
187        FROM candidato
188        WHERE id_partido IS NOT NULL
189        AND id_candidato = NEW.id_candidato;
190        IF partido_existente > 0 THEN
191            INSERT INTO bitacora VALUES (null, sysdate(), user(), 'CANDIDATOS',
192            CONCAT('Intento de asignar más de un partido al candidato ', NEW.id_candidato));
193            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error: El candidato ya pertenece a un partido político.';
194        END IF;
195    END //
196    DELIMITER ;
```

define un trigger bi_partidos que se activa antes de insertar un nuevo registro en la tabla candidato. Su objetivo es verificar que un candidato no esté asociado a más de un partido político.

DROP TRIGGER IF EXISTS bi_partidos: Elimina el trigger bi_partidos si ya existe para evitar conflictos al crear uno nuevo.

CREATE TRIGGER bi_partidos: Define un nuevo trigger llamado bi_partidos.

BEFORE INSERT ON candidato: Especifica que el trigger se activa antes de insertar un registro en la tabla candidato.

FOR EACH ROW: Indica que el trigger se ejecutará una vez por cada fila afectada por la operación de inserción.

El bloque BEGIN ... END contiene la lógica del trigger.

DECLARE partido_existente INT;: Declara una variable partido_existente que se utilizará para almacenar el número de partidos asociados al candidato.

SELECT COUNT(*) INTO partido_existente ...: Realiza una consulta para contar el número de registros en la tabla candidato que tienen un partido asociado y coinciden con el id_candidato del nuevo candidato que se está insertando.

IF partido_existente > 0 THEN ...: Verifica si el candidato ya está asociado a un partido político. Si es así, registra un mensaje en la bitácora y genera un error utilizando SIGNAL para indicar que el candidato no puede pertenecer a más de un partido político.

```
198   -- verifica que no tenga mas de una coalicion
199   DELIMITER //
200   DROP TRIGGER IF EXISTS bi_mascoalicion //
201   CREATE TRIGGER bi_mascoalicion
202   BEFORE INSERT ON candidato
203   FOR EACH ROW
204   BEGIN
205       DECLARE coalicion_existente INT;
206       SELECT COUNT(*) INTO coalicion_existente
207       FROM candidato
208       WHERE id_coalicion IS NOT NULL
209       AND id_candidato = NEW.id_candidato;
210       IF coalicion_existente > 0 THEN
211           INSERT INTO bitacora VALUES (null, sysdate(), user(), 'CANDIDATOS',
212           CONCAT('Intento de asignar más de una coalición al candidato ', NEW.id_candidato));
213           SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error: El candidato ya pertenece a una coalición.';
214       END IF;
215   END //
216   DELIMITER ;
```

define un trigger llamado bi_mascoalicion que se activa antes de insertar un nuevo registro en la tabla candidato. Su propósito es verificar que un candidato no esté asociado a más de una coalición.

DROP TRIGGER IF EXISTS bi_mascoalicion: Elimina el trigger bi_mascoalicion si ya existe para evitar conflictos al crear uno nuevo.

CREATE TRIGGER bi_mascoalicion: Define un nuevo trigger llamado bi_mascoalicion.

BEFORE INSERT ON candidato: Especifica que el trigger se activa antes de insertar un registro en la tabla candidato.

FOR EACH ROW: Indica que el trigger se ejecutará una vez por cada fila afectada por la operación de inserción.

El bloque BEGIN ... END contiene la lógica del trigger.

DECLARE coalicion_existente INT;: Declara una variable coalicion_existente que se utilizará para almacenar el número de coaliciones asociadas al candidato.

SELECT COUNT(*) INTO coalicion_existente ...: Realiza una consulta para contar el número de registros en la tabla candidato que tienen una coalición asociada y coinciden con el id_candidato del nuevo candidato que se está insertando.

IF coalicion_existente > 0 THEN ...: Verifica si el candidato ya está asociado a una coalición. Si es así, registra un mensaje en la bitácora y genera un error utilizando SIGNAL para indicar que el candidato no puede pertenecer a más de una coalición.

```
218   -- verifica que el candidato no tenga mas de un cargo
219   DELIMITER //
220   DROP TRIGGER IF EXISTS bi_mascargo //
221   CREATE TRIGGER bi_mascargo
222   BEFORE INSERT ON candidato
223   FOR EACH ROW
224   BEGIN
225       DECLARE cargo_existente INT;
226       SELECT COUNT(*) INTO cargo_existente
227       FROM candidato
228       WHERE id_cargo IS NOT NULL
229       AND id_candidato = NEW.id_candidato;
230       IF cargo_existente > 0 THEN
231           INSERT INTO bitacora VALUES (null, sysdate(), user(), 'CANDIDATOS',
232           CONCAT('Intento de asignar más de un cargo al candidato ', NEW.id_candidato));
233           SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error: El candidato ya tiene asignado un cargo.';
234       END IF;
235   END //
236   DELIMITER ;
```

Se define un trigger llamado bi_mascargo que se activa antes de insertar un nuevo registro en la tabla candidato.
Su propósito es verificar que un candidato no esté asociado a más de un cargo.
DROP TRIGGER IF EXISTS bi_mascargo: Elimina el trigger bi_mascargo si ya existe, evitando conflictos al crear uno nuevo.
CREATE TRIGGER bi_mascargo: Define un nuevo trigger llamado bi_mascargo.
BEFORE INSERT ON candidato: Especifica que el trigger se activa antes de insertar un registro en la tabla candidato.
FOR EACH ROW: Indica que el trigger se ejecutará una vez por cada fila afectada por la operación de inserción.
DECLARE cargo_existente INT: Declara una variable cargo_existente que se utilizará para almacenar el número de cargos asociados al candidato.
SELECT COUNT(*) INTO cargo_existente: Realiza una consulta para contar el número de registros en la tabla candidato que tienen un cargo asociado y coinciden con el id_candidato del nuevo candidato que se está insertando.
IF cargo_existente > 0 THEN: Verifica si el candidato ya está asociado a un cargo. Si es así, registra un mensaje en la bitácora y genera un error utilizando SIGNAL para indicar que el candidato no puede tener más de un cargo.

```sql
select *
from INFORMATION_SCHEMA.TRIGGERS
WHERE event_object_schema = 'candidatos';
```

Busca en el esquema de información del sistema (INFORMATION_SCHEMA) todos los desencadenadores (triggers) que estén asociados al esquema de la base de datos llamado 'candidatos'.

INSERCIÓN DE DATOS

```sql
use candidatos;
```

1. Selecciona la base de datos candidatos para que todas las operaciones se ejecuten en esta base de datos.

```sql
-- insercion candidatos
INSERT INTO candidato (rfc,nombre,ap_paterno,ap_materno,fnac,direccion,telefono,email,propuestas,finscrip)
VALUES
('KNE22RTR8EC', 'Bruce', 'Reyes','Gomez','1977-05-28','Mexico City', 8587855767, 'sed.sem.egestas@protonmail.org','sostenibilidad, tecnologia, infraestructura, incl
('PXF11VUR1VA','Kyle','Perez','Santana','1978-08-31','San Juan del Rio','6511713168','volutpat.nulla.facilisis@aol.net','transparencia, igualdad, desarrollo, sosten
('JER25GTR0VW','Connor','Caballero','Cortes','1992-05-09','Torreón','7175113948','vel@hotmail.org','salud, economia, empleo, seguridad, justicia, transparencia, igu
('ZVL68MBY8KS','Bell','Maria','Hidalgo','1955-03-19','Hermosillo','6347785918','arcu@google.ca','salud, economía, empleo, seguridad, justicia, transparencia, igualdad
('MNR16OID7TU','Paloma','Suarez','Santos','1986-05-24','Tehuacán','4882058272','aliquet@icloud.net','cambio, innovación, educación, salud, economía, empleo, seguridad
('WVK88LTL7DH','Harper','Reyes','Castillo','1982-08-30','Juárez','6269399561','nec.luctus@protonmail.net','igualdad, desarrollo, sostenibilidad, tecnología, infraes
('VQL51MJQ5SZ','Brock','Diego','Mendez','1987-12-21','Matamoros','8259883618','vivamus@protonmail.ca','justicia, transparencia, igualdad, desarrollo, sostenibilidad
('UHP61XJR5ON','Quemby','Vera','Pardo','1963-06-29','Soledad de Graciano Sánchez','5297053407','cursus.diam@icloud.ca','innovación, educación, salud, economía, empl
('JVM54RSZ6ZD','Fredericka','Gonzalez','Castro','1954-08-16','San Juan del Río','3188819221','phasellus@icloud.org','economía, empleo, seguridad, justicia, transpar
('QMT96BKE5XQ','Kimberly','Isabella','Diez','1978-05-11','Navojoa','8616308657','nunc.nulla@yahoo.net','desarrollo, sostenibilidad, tecnología, infraestructura, inc
('FHU64KFF4KO','Judah','Castillo','Munoz','1959-12-11','Acuña','6754671638','egestas@protonmail.net','cambio, innovación, educación, salud, economía, empleo, seguri
('UFQ00OQI4TA','Maya','Caballero','Augustin','1965-06-29','Guadalajara','8402870466','risus.at@icloud.couk','Liderazgo, cambio, innovación, educación, salud, econom
('GIN27BUZ5JS','Madison','Mendez','Lorenzo','1979-12-29','Irapuato','0753320118','donec.nibh.enim@hotmail.ca','seguridad, justicia, transparencia, igualdad, desarro
('ZRB59PGE1MP','Debra','Flores','Cano','1967-01-28','San Juan del Río','4916573223','est.ac.mattis@outlook.org','desarrollo, sostenibilidad, tecnología, infraestruc
('YOY10GYD3EF','Boris','Pardo','Sanchez','1979-09-15','Monterrey','4731556210','nec.cursus@protonmail.ca','economía, empleo, seguridad, justicia, transparencia, igu
('WRB45PRQ5YW','Quamar','Testudines','Áez','1974-06-02','Chilpancingo','2242812343','tortor.integer@yahoo.com','seguridad, justicia, transparencia, igualdad, desarr
('NKM38NST9HV','Ahmed','Florencia','Vega','1977-11-12','Villahermosa','2719234338','phasellus.at.augue@icloud.com','transparencia, igualdad, desarrollo, sostenibili
('ALC87TWM4DI','Emmanuel','Sebastian','Augustin','1968-05-24','Irapuato','1992325207','montes.nascetur@hotmail.ca','igualdad, desarrollo, sostenibilidad, tecnología
('DCR99DMK6DS','Andrew','Hernandez','Castro','1984-12-21','Tuxtla Gutiérrez','8662111321','mauris.ut@google.edu','economía, empleo, seguridad, justicia, transparenc
('RDT45BIH4TJ','Eve','Ramos','Vargas','1995-04-22','Guadalupe','6815285502','quisque@protonmail.net','cambio, innovación, educación, salud, economía, empleo, seguri
('DZV20LQR1VH','Robert','Trinidad','Martin','1968-03-11','Saltillo','1137597755','quisque@protonmail.couk','seguridad, justicia, transparencia, igualdad, desarrollo
```

Esta instrucción prepara la inserción de datos en la tabla candidato, especificando las columnas en las que se insertarán los valores.
Columnas Especificadas:
rfc: Clave única para identificar a cada candidato.
nombre: Nombre del candidato.
ap_paterno: Apellido paterno del candidato.
ap_materno: Apellido materno del candidato.
fnac: Fecha de nacimiento del candidato.

direccion: Dirección de residencia del candidato.
telefono: Número de teléfono del candidato.
email: Dirección de correo electrónico del candidato.
propuestas: Lista de propuestas o temas de campaña del candidato.
finscrip: Fecha de inscripción del candidato.

Funcionalidad general de esta parte del código:
Propósito: Cada conjunto de valores corresponde a un candidato y se inserta en la tabla candidato en las columnas especificadas.
Función: Estas filas representan datos individuales para cada candidato, incluyendo detalles personales y propuestas de campaña.
Inserción de Datos: El código inserta múltiples registros en la tabla candidato en una sola operación INSERT. Cada fila de datos contiene información relevante sobre un candidato.
Mantenimiento de la Base de Datos: Al insertar estos datos, se está llenando la tabla candidato con información necesaria para el sistema electoral o de gestión de candidatos.
Integridad de Datos: Asume que las restricciones y claves están correctamente definidas en la tabla candidato para mantener la integridad de los datos, como asegurar que rfc sea único para cada candidato.

```sql
-- insercion de partidos politicos
INSERT INTO partido_politico (nombre, abreviatura, telefono, email) VALUES
('Partido Revolucionario Institucional', 'PRI', 5512345678, 'contacto@pri.org.mx'),
('Partido Acción Nacional', 'PAN', 5512345679, 'contacto@pan.org.mx'),
('Partido de la Revolución Democrática', 'PRD', 5512345680, 'contacto@prd.org.mx'),
('Partido Verde Ecologista de México', 'PVEM', 5512345681, 'contacto@pvem.org.mx'),
('Movimiento Regeneración Nacional', 'MORENA', 5512345682, 'contacto@morena.org.mx'),
('Partido del Trabajo', 'PT', 5512345683, 'contacto@pt.org.mx'),
('Movimiento Ciudadano', 'MC', 5512345684, 'contacto@mc.org.mx'),
('Encuentro Social', 'ES', 5512345685, 'contacto@es.org.mx');
```

Esta instrucción prepara la inserción de datos en la tabla partido_politico, especificando las columnas en las que se insertarán los valores.
Columnas Especificadas:
nombre: El nombre completo del partido político.
abreviatura: La abreviatura o sigla del partido político.
telefono: El número de teléfono de contacto del partido político.
email: La dirección de correo electrónico de contacto del partido político.
Inserción de Datos: El código inserta múltiples registros en la tabla partido_politico en una sola operación INSERT. Cada fila de datos contiene información relevante sobre un partido político.

Mantenimiento de la Base de Datos: Al insertar estos datos, se está llenando la tabla partido_politico con información necesaria para el sistema de gestión de partidos políticos.
Integridad de Datos: Asume que las restricciones y claves están correctamente definidas en la tabla partido_politico para mantener la integridad de los datos, como asegurar que los nombres y abreviaturas de los partidos sean únicos.

```sql
-- insercion coalicion
INSERT INTO coalicion (id_partido, id_partido2 ,nombre) VALUES
(2,3,'Unidos por México'),
(5,6,'Alianza por el Futuro'),
(1,4,'Juntos Somos Más');
```

Esta instrucción prepara la inserción de datos en la tabla coalicion, especificando las columnas en las que se insertarán los valores.
Columnas Especificadas:
id_partido: Identificador del primer partido político en la coalición.
id_partido2: Identificador del segundo partido político en la coalición.
nombre: El nombre de la coalición.
Estas filas representan datos individuales para cada coalición, incluyendo los identificadores de los partidos que la conforman y el nombre de la coalición.

```sql
47    -- insercion militancia
48 •  insert into militancia (nombre) value('independiente');
49
```

Esta instrucción inserta un nuevo registro en la tabla militancia
nombre: El nombre del partido político o la designación de "independiente" si el candidato no pertenece a ningún partido.

```sql
-- insercion municipios
INSERT INTO municipios (id_municipio, nombre, num_habitantes, num_electores) VALUES
-- Aguascalientes
(1, 'Aguascalientes', 934424, 543682),
(2, 'Jesús María', 118173, 75000),
(3, 'Rincón de Romos', 51087, 32000),
(4, 'Pabellón de Arteaga', 46736, 29000),
(5, 'San Francisco de los Romo', 50946, 33000),
(6, 'Calvillo', 56292, 34000),
(7, 'Asientos', 45292, 28000),
-- Baja California
(8, 'Tijuana', 1940922, 1200000),
(9, 'Mexicali', 1075636, 800000),
(10, 'Ensenada', 522768, 350000),
(11, 'Rosarito', 126890, 80000),
(12, 'Tecate', 108440, 70000),
(13, 'San Quintín', 104020, 68000),
(14, 'San Felipe', 16800, 10000),
-- Baja California Sur
(15, 'La Paz', 317764, 189762),
(16, 'Los Cabos', 351111, 200000),
(17, 'Comondú', 70448, 45000),
(18, 'Loreto', 18912, 12000),
(19, 'Mulegé', 63934, 40000),
(20, 'Todos Santos', 7545, 5000),
(21, 'Cabo San Lucas', 202694, 150000),
```

```sql
-- Campeche
(22, 'Campeche', 294077, 200000),
(23, 'Ciudad del Carmen', 248303, 180000),
(24, 'Champotón', 82513, 50000),
(25, 'Escárcega', 58380, 40000),
(26, 'Calkiní', 49850, 30000),
(27, 'Hecelchakán', 28427, 18000),
(28, 'Candelaria', 43200, 25000),

-- Coahuila
(29, 'Saltillo', 879958, 520000),
(30, 'Torreón', 679288, 450000),
(31, 'Monclova', 237151, 150000),
(32, 'Piedras Negras', 169771, 110000),
(33, 'Acuña', 181426, 120000),
(34, 'San Pedro', 100879, 65000),
(35, 'Frontera', 83577, 55000),

-- Colima
(36, 'Colima', 150673, 100000),
(37, 'Manzanillo', 191031, 130000),
(38, 'Tecomán', 130690, 85000),
(39, 'Villa de Álvarez', 152571, 105000),
(40, 'Comala', 20573, 13000),
(41, 'Coquimatlán', 21188, 14000),
(42, 'Armería', 29062, 18000),

-- Chiapas
(43, 'Tuxtla Gutiérrez', 598710, 400000),
(44, 'Tapachula', 353706, 250000),
(45, 'San Cristóbal de las Casas', 215874, 150000),
(46, 'Comitán', 141013, 90000),
(47, 'Palenque', 108720, 70000),
(48, 'Ocosingo', 200446, 130000),
(49, 'Cintalapa', 88106, 49201),
```

```sql
-- Chihuahua
(50, 'Chihuahua', 925762, 650000),
(51, 'Juárez', 1491583, 980000),
(52, 'Delicias', 148808, 95000),
(53, 'Cuauhtémoc', 183307, 110000),
(54, 'Parral', 123794, 80000),
(55, 'Nuevo Casas Grandes', 60619, 40000),
(56, 'Camargo', 57382, 35000),

-- Durango
(64, 'Durango', 654876, 400000),
(65, 'Gómez Palacio', 372344, 250000),
(66, 'Lerdo', 141201, 90000),
(67, 'Pueblo Nuevo', 49263, 30000),
(68, 'Santiago Papasquiaro', 48877, 29000),
(69, 'Canatlán', 33196, 21000),
(70, 'Nuevo Ideal', 28375, 18000),

-- Guanajuato
(71, 'León', 1721215, 1200000),
(72, 'Irapuato', 651844, 400000),
(73, 'Celaya', 521169, 350000),
(74, 'Salamanca', 273271, 180000),
(75, 'Silao', 202672, 130000),
(76, 'Guanajuato', 194500, 120000),
(77, 'San Miguel de Allende', 174615, 110000),

-- Guerrero
(78, 'Acapulco', 779566, 500000),
(79, 'Chilpancingo', 214219, 150000),
(80, 'Iguala', 140363, 90000),
(81, 'Zihuatanejo', 125780, 80000),
(82, 'Taxco', 105944, 70000),
(83, 'Chilapa', 120790, 80000),
(84, 'Tlapa', 73781, 50000),
```

```
-- Hidalgo
(85, 'Pachuca', 314331, 250000),
(86, 'Tulancingo', 167759, 110000),
(87, 'Tizayuca', 153152, 100000),
(88, 'Tula de Allende', 111648, 70000),
(89, 'Tepeji del Río', 87478, 50000),
(90, 'Ixmiquilpan', 91296, 60000),
(91, 'Huejutla', 115786, 75000),

-- Jalisco
(92, 'Guadalajara', 1488282, 1200000),
(93, 'Zapopan', 1425876, 1100000),
(94, 'Tlaquepaque', 678823, 500000),
(95, 'Tonalá', 459974, 300000),
(96, 'Puerto Vallarta', 291839, 200000),
(97, 'Lagos de Moreno', 172483, 110000),
(98, 'Tepatitlán', 141322, 90000),

-- México
(99, 'Ecatepec', 1655015, 1200000),
(100, 'Nezahualcóyotl', 1206313, 800000),
(101, 'Naucalpan', 834434, 600000),
(102, 'Toluca', 910608, 600000),
(103, 'Tlalnepantla', 715267, 500000),
(104, 'Chimalhuacán', 702924, 500000),
(105, 'Atizapán de Zaragoza', 511711, 350000),

-- Michoacán
(106, 'Morelia', 784776, 540000),
(107, 'Uruapan', 368408, 200000),
(108, 'Zamora', 186102, 120000),
(109, 'Lázaro Cárdenas', 192715, 120000),
(110, 'Zitácuaro', 105534, 120000),
(111, 'Apatzingán', 132515, 80000),
(112, 'Pátzcuaro', 89011, 60000),

-- Morelos
(113, 'Cuernavaca', 366321, 250000),
(114, 'Jiutepec', 221713, 150000),
(115, 'Temixco', 141184, 90000),
(116, 'Cuautla', 187099, 120000),
(117, 'Yautepec', 110630, 70000),
(118, 'Emiliano Zapata', 107945, 70000),
(119, 'Jojutla', 61832, 40000),
```

```
-- Nayarit
(120, 'Tepic', 503338, 300000),
(121, 'Bahía de Banderas', 184225, 120000),
(122, 'Xalisco', 57396, 35000),
(123, 'Compostela', 65494, 40000),
(124, 'Santiago Ixcuintla', 87361, 55000),
(125, 'San Blas', 46589, 30000),
(126, 'Ixtlán del Río', 30611, 20000),

-- Nuevo León
(127, 'Monterrey', 1135512, 900000),
(128, 'Guadalupe', 673616, 450000),
(129, 'San Nicolás de los Garza', 443273, 300000),
(130, 'Apodaca', 652130, 480000),
(131, 'Santa Catarina', 304398, 200000),
(132, 'San Pedro Garza García', 132169, 90000),
(133, 'General Escobedo', 401214, 300000),

-- Oaxaca
(134, 'Oaxaca de Juárez', 300050, 200000),
(135, 'Salina Cruz', 83526, 50000),
(136, 'Juchitán', 91123, 60000),
(137, 'San Juan Bautista Tuxtepec', 103600, 70000),
(138, 'Heroica Ciudad de Huajuapan de León', 70684, 50000),
(139, 'Santa Cruz Xoxocotlán', 92485, 60000),
(140, 'Puerto Escondido', 45684, 30000),

-- Puebla
(141, 'Puebla', 1697960, 1200000),
(142, 'Tehuacán', 319375, 200000),
(143, 'San Martín Texmelucan', 155738, 100000),
(144, 'Atlixco', 117262, 85000),
(145, 'San Pedro Cholula', 140562, 90000),
(146, 'San Andrés Cholula', 137314, 85000),
(147, 'Huauchinango', 104213, 68000),

-- Querétaro
(148, 'Querétaro', 878931, 600000),
(149, 'San Juan del Río', 311047, 200000),
(150, 'El Marqués', 166497, 100000),
(151, 'Corregidora', 208667, 120000),
(152, 'Tequisquiapan', 72644, 45000),
(153, 'Pedro Escobedo', 65351, 40000),
(154, 'Cadereyta de Montes', 67315, 48000),
```

```
-- Quintana Roo
(155, 'Cancún', 888797, 600000),
(156, 'Chetumal', 151243, 100000),
(157, 'Playa del Carmen', 202702, 150000),
(158, 'Cozumel', 88087, 50000),
(159, 'Tulum', 46621, 30000),
(160, 'Felipe Carrillo Puerto', 81008, 50000),
(161, 'José María Morelos', 43836, 25000),

-- San Luis Potosí
(162, 'San Luis Potosí', 951908, 600000),
(163, 'Soledad de Graciano Sánchez', 300214, 200000),
(164, 'Ciudad Valles', 176935, 110000),
(165, 'Matehuala', 101615, 65000),
(166, 'Rioverde', 98338, 60000),
(167, 'Tamazunchale', 96836, 60000),
(168, 'Ébano', 38828, 25000),

-- Sinaloa
(169, 'Culiacán', 905265, 650000),
(170, 'Mazatlán', 502547, 350000),
(171, 'Los Mochis', 365967, 240000),
(172, 'Guasave', 300115, 190000),
(173, 'Navolato', 135603, 85000),
(174, 'Escuinapa', 62463, 40000),
(175, 'El Fuerte', 125674, 75000),

-- Sonora
(176, 'Hermosillo', 936263, 650000),
(177, 'Ciudad Obregón', 436484, 300000),
(178, 'Nogales', 264702, 180000),
(179, 'San Luis Río Colorado', 210060, 150000),
(180, 'Navojoa', 164450, 100000),
(181, 'Guaymas', 150063, 90000),
(182, 'Agua Prieta', 79420, 50000),

-- Tabasco
(183, 'Villahermosa', 684847, 450000),
(184, 'Cárdenas', 282353, 180000),
(185, 'Comalcalco', 210231, 140000),
(186, 'Macuspana', 165729, 100000),
(187, 'Huimanguillo', 145285, 90000),
(188, 'Jalpa de Méndez', 84720, 50000),
(189, 'Teapa', 55380, 30000),
```

```
-- Tamaulipas
(190, 'Reynosa', 957327, 700000),
(191, 'Matamoros', 544606, 400000),
(192, 'Nuevo Laredo', 407436, 350000),
(193, 'Ciudad Victoria', 349688, 250000),
(194, 'Tampico', 314418, 220000),
(195, 'Madero', 209175, 150000),
(196, 'Altamira', 210177, 140000),

-- Tlaxcala
(197, 'Tlaxcala', 124018, 80000),
(198, 'Apizaco', 81320, 55000),
(199, 'Huamantla', 107143, 70000),
(200, 'Chiautempan', 71964, 45000),
(201, 'San Pablo del Monte', 85743, 55000),
(202, 'Zacatelco', 54314, 35000),
(203, 'Contla', 44512, 30000),

-- Veracruz
(204, 'Xalapa', 488531, 300000),
(205, 'Veracruz', 609964, 400000),
(206, 'Coatzacoalcos', 319187, 200000),
(207, 'Córdoba', 204748, 120000),
(208, 'Poza Rica', 200110, 120000),
(209, 'Minatitlán', 157840, 100000),
(210, 'Tuxpan', 154470, 90000),

-- Yucatán
(211, 'Mérida', 892363, 600000),
(212, 'Valladolid', 85913, 50000),
(213, 'Tizimín', 76812, 45000),
(214, 'Progreso', 58668, 35000),
(215, 'Kanasín', 141554, 90000),
(216, 'Motul', 42335, 25000),
(217, 'Umán', 56045, 35000),
-- Zacatecas
(218, 'Zacatecas', 149607, 100000),
(219, 'Fresnillo', 240532, 150000),
(220, 'Villanueva',29527,23000),
(221, 'Jerez', 62219, 40000),
(222, 'Sombrerete', 70818, 45000),
(223, 'Río Grande', 62220, 35000),
(224, 'Calera', 41906, 33000);
```

Esta instrucción específica que se van a insertar múltiples filas de datos en la tabla de municipios. Los valores se proporcionarán para las columnas id_municipio, nombre, num_habitantes y num_electores
Estos valores representan los datos específicos que se van a insertar en la tabla de municipios. Cada fila incluye:
id_municipio: Un identificador único para el municipio.
nombre: El nombre del municipio.
num_habitantes: El número total de habitantes en el municipio.
num_electores: El número total de electores registrados en el municipio.

```
328     -- insercion distrito
329 •   INSERT INTO distritos (nombre, num_habitantes, num_electores) VALUES
330     ('Álvaro Obregón', 759137, 550000),
331     ('Coyoacán', 620416, 450000),
332     ('Cuauhtémoc', 531831, 380000),
333     ('Gustavo A. Madero', 1160537, 800000),
334     ('Iztapalapa', 1820885, 1200000),
335     ('Miguel Hidalgo', 414470, 300000),
336     ('Tlalpan', 677104, 450000);
```

Este bloque de código inserta registros en la tabla distritos. Cada registro representa un distrito con tres columnas:
nombre: Nombre del distrito.
num_habitantes: Número de habitantes del distrito.
num_electores: Número de electores en el distrito.

```
338     -- insercio historial de elecciones pasadas
339 •   INSERT INTO historial (cargo_solicitado, resultado, fecha_eleccion) VALUES
340     ('Presidente Municipal', 'Electo', '2018-07-01'),
341     ('Diputado Federal', 'No Electo', '2016-06-05'),
342     ('Gobernador', 'Electo', '2014-07-07'),
343     ('Senador', 'No Electo', '2012-07-01'),
344     ('Regidor', 'Electo', '2010-07-04'),
345     ('Síndico', 'No Electo', '2008-07-06'),
346     ('Jefe de Gobierno', 'Electo', '2006-07-02'),
347     ('Diputado Local', 'No Electo', '2004-07-04'),
348     ('Presidente Municipal', 'No Electo', '2018-07-01'),
349     ('Diputado Federal', 'Electo', '2016-06-05'),
350     ('Gobernador', 'No Electo', '2014-07-07'),
351     ('Senador', 'Electo', '2012-07-01'),
352     ('Regidor', 'No Electo', '2010-07-04'),
353     ('Síndico', 'Electo', '2008-07-06'),
354     ('Jefe de Gobierno', 'No Electo', '2006-07-02'),
355     ('Diputado Local', 'Electo', '2004-07-04'),
356     ('Presidente Municipal', 'Electo', '2002-07-07'),
357     ('Diputado Federal', 'No Electo', '2000-07-02'),
358     ('Gobernador', 'Electo', '1998-07-05'),
359     ('Senador', 'No Electo', '1996-07-07'),
360     ('Regidor', 'Electo', '1994-07-03');
```

El código inserta múltiples registros en la tabla historial. Cada registro representa un evento electoral con tres campos:
cargo_solicitado: El cargo por el cual se postuló el candidato (por ejemplo, Presidente Municipal, Diputado Federal, Gobernador, etc.).
resultado: El resultado de la elección (por ejemplo, Electo o No Electo).
fecha_eleccion: La fecha en la que se realizó la elección.
El propósito de este código es agregar registros históricos sobre elecciones pasadas en la base de datos. Estos datos pueden ser utilizados para varios fines, como:
Análisis histórico de los resultados electorales.
Investigación y estudios sobre tendencias electorales.
Consultas administrativas y planeación de futuras elecciones.

```sql
-- insercion documentacion
INSERT INTO documentacion (documento, verificacion_eligibilidad, declaracion_interes) VALUES
('Acta de nacimiento', 'Verificado', 'Declaro no tener conflictos de interés en ninguna empresa privada.'),
('CURP', 'Verificado', 'Declaro tener participación en una ONG sin fines de lucro.'),
('INE', 'Verificado', 'Declaro ser miembro del Colegio de Abogados de México.'),
('Comprobante de domicilio', 'Verificado', 'Declaro no tener bienes inmuebles más allá de mi residencia.'),
('Título profesional', 'Verificado', 'Declaro no tener relaciones contractuales con el gobierno.'),
('Cédula profesional', 'Verificado', 'Declaro ser socio de una firma consultora, pero sin contratos gubernamentales.'),
('Carta de no antecedentes penales', 'Verificado', 'Declaro haber participado en varias conferencias de políticas públicas.'),
('Carta de residencia', 'Verificado', 'Declaro no tener inversiones en el extranjero.'),
('Certificado de salud', 'Verificado', 'Declaro no tener deudas significativas que puedan afectar mi juicio.'),
('Fotografía reciente', 'Verificado', 'Declaro haber recibido donaciones para actividades comunitarias.'),
('Declaración de impuestos', 'Verificado', 'Declaro haber cumplido con todas mis obligaciones fiscales.'),
('Estado de cuenta bancario', 'Verificado', 'Declaro tener una cuenta de ahorros sin movimientos significativos.'),
('Certificado de no inhabilitación', 'Verificado', 'Declaro no haber sido sancionado administrativamente.'),
('Carta de no conflicto de intereses', 'Verificado', 'Declaro que mis familiares no tienen contratos con el gobierno.'),
('Carta de aceptación de candidatura', 'Verificado', 'Declaro haber aceptado la candidatura sin presiones externas.'),
('Plan de gobierno', 'Verificado', 'Declaro haber elaborado mi plan de gobierno basándome en consultas ciudadanas.'),
('Propuesta económica', 'Verificado', 'Declaro no tener compromisos financieros con entidades privadas.'),
('Propuesta de seguridad', 'Verificado', 'Declaro haber consultado con expertos en seguridad para mi propuesta.'),
('Propuesta de educación', 'Verificado', 'Declaro mi compromiso con la transparencia en el sector educativo.'),
('Propuesta de salud', 'Verificado', 'Declaro no tener relaciones con empresas farmacéuticas.'),
('Propuesta de infraestructura', 'Verificado', 'Declaro haber considerado el impacto ambiental en mis propuestas de infraestructura.');
```

El propósito de este código es ingresar los tipos de documentos requeridos para la verificación de elegibilidad y declaración de intereses en la base de datos. Estos datos son esenciales para garantizar la transparencia y la integridad en los procesos electorales y administrativos.
Cómo Funciona:
INSERT INTO documentacion: Especifica la tabla documentacion en la que se van a insertar los datos.
(documento, verificacion_eligibilidad, declaracion_interes): Especifica las columnas en las que se van a insertar los datos.
VALUES: Proporciona los valores a insertar en las columnas correspondientes, agrupados en filas.

```sql
-- insercion cargos
insert into cargo (nombre_cargo) values
('Presidente'),('Gobernador'),('Diputado'),('Senador'),('Presidente Municipal');
```

insert into cargo (nombre_cargo) values: Esta es la declaración de inserción. Indica que se van a insertar datos en la tabla cargo en la columna nombre_cargo. ('Presidente'),('Gobernador'),('Diputado'),('Senador'),('Presidente Municipal'): Estos son los valores que se están insertando en la columna nombre_cargo. Cada valor está entre paréntesis y separado por comas.
El propósito de este código es simplemente agregar los nombres de diferentes cargos políticos o gubernamentales a la tabla cargo, probablemente para su uso posterior en el sistema de gestión de elecciones o administración gubernamental.

```
-- insercion municipios y dsitritos a estados
-- Ciudad de Mexico
UPDATE distritos SET id_estado = 9 WHERE nombre IN ('Álvaro Obregón','Coyoacán', 'Cuauhtémoc','Gustavo A. Madero','Iztapalapa','Miguel Hidalgo','Tlalpan');
-- Aguascalientes
UPDATE municipios SET id_estado = 1 WHERE nombre IN ( 'Aguascalientes','Jesús María', 'Rincón de Romos' ,'Pabellón de Arteaga' ,'San Francisco de los Romo' ,'Calvill
-- Baja California
UPDATE municipios SET id_estado = 2 WHERE nombre IN ('Tijuana', 'Mexicali','Ensenada', 'Rosarito' ,'Tecate', 'San Quintín', 'San Felipe');
-- Baja California Sur
UPDATE municipios SET id_estado = 3 WHERE nombre IN ('La Paz','Los Cabos','Comondú', 'Loreto','Mulegé','Todos Santos','Cabo San Lucas');
-- Campeche
UPDATE municipios SET id_estado = 4 WHERE nombre IN ('Campeche','Ciudad del Carmen','Champotón','Escárcega','Calkiní', 'Hecelchakán','Candelaria');
-- Coahuila
UPDATE municipios SET id_estado = 5 WHERE nombre IN ('Saltillo', 'Torreón' ,'Monclova','Piedras Negras','Acuña' ,'San Pedro','Frontera');
-- Colima
UPDATE municipios SET id_estado = 6 WHERE nombre IN ('Colima','Manzanillo','Tecomán','Villa de Álvarez', 'Comala','Coquimatlán','Armería');
-- Chiapas
UPDATE municipios SET id_estado = 7 WHERE nombre IN ('Tuxtla Gutiérrez','Tapachula','San Cristóbal de las Casas','Comitán','Palenque','Ocosingo','Cintalapa');
-- Chihuahua
UPDATE municipios SET id_estado = 8 WHERE nombre IN ('Chihuahua','Juárez','Delicias','Cuauhtémoc','Parral','Nuevo Casas Grandes','Camargo');
-- Durango
UPDATE municipios SET id_estado = 10 WHERE nombre IN ('Durango','Gómez Palacio','Lerdo','Pueblo Nuevo','Santiago Papasquiaro','Canatlán','Nuevo Ideal');
-- Guanajuato
UPDATE municipios SET id_estado = 11 WHERE nombre IN ('León','Irapuato','Celaya','Salamanca','Silao','Guanajuato','San Miguel de Allende');
-- Guerrero
UPDATE municipios SET id_estado = 12 WHERE nombre IN ('Acapulco','Chilpancingo','Iguala','Zihuatanejo','Taxco','Chilapa','Tlapa');
-- Hidalgo
UPDATE municipios SET id_estado = 13 WHERE nombre IN ('Pachuca','Tulancingo','Tizayuca' ,'Tula de Allende','Tepeji del Río' ,'Ixmiquilpan','Huejutla');
```

```
-- Jalisco
UPDATE municipios SET id_estado = 14 WHERE nombre IN ('Guadalajara','Zapopan','Tlaquepaque','Tonalá','Puerto Vallarta','Lagos de Moreno','Tepatitlán');
-- México
UPDATE municipios SET id_estado = 15 WHERE nombre IN ('Ecatepec','Nezahualcóyotl','Naucalpan','Toluca','Tlalnepantla','Chimalhuacán','Atizapán de Zaragoza');
-- Michoacán
UPDATE municipios SET id_estado = 16 WHERE nombre IN ('Morelia','Uruapan','Zamora','Lázaro Cárdenas','Zitácuaro','Apatzingán','Pátzcuaro');
-- Morelos
UPDATE municipios SET id_estado = 17 WHERE nombre IN ('Cuernavaca','Jiutepec','Temixco','Cuautla','Yautepec','Emiliano Zapata','Jojutla');
-- Nuevo León
UPDATE municipios SET id_estado = 18 WHERE nombre IN ('Monterrey','Guadalupe','San Nicolás de los Garza','Apodaca','Santa Catarina', 'San Pedro Garza Garcia' 'Genera
-- Nayarit
UPDATE municipios SET id_estado = 19 WHERE nombre IN ('Tepic','Bahía de Banderas','Xalisco','Compostela', 'Santiago Ixcuintla','San Blas','Ixtlán del Río' );
-- Oaxaca
UPDATE municipios SET id_estado = 20 WHERE nombre IN ('Oaxaca de Juárez', 'Salina Cruz', 'Juchitán', 'San Juan Bautista Tuxtepec', 'Heroica Ciudad de Huajuapan de Le
-- Puebla
UPDATE municipios SET id_estado = 21 WHERE nombre IN ('Puebla', 'Tehuacán', 'San Martín Texmelucan', 'Atlixco', 'San Pedro Cholula', 'San Andrés Cholula', 'Huauchina
-- Querétaro
UPDATE municipios SET id_estado = 22 WHERE nombre IN ('Querétaro', 'San Juan del Río', 'El Marqués', 'Corregidora', 'Tequisquiapan', 'Pedro Escobedo', 'Cadereyta de
-- Quintana Roo
UPDATE municipios SET id_estado = 23 WHERE nombre IN ('Cancún', 'Chetumal', 'Playa del Carmen', 'Cozumel', 'Tulum', 'Felipe Carrillo Puerto', 'José María Morelos');
-- San Luis Potosí
UPDATE municipios SET id_estado = 24 WHERE nombre IN ('San Luis Potosí', 'Soledad de Graciano Sánchez', 'Ciudad Valles', 'Matehuala', 'Rioverde', 'Tamazunchale', 'Ebe
-- Sinaloa
UPDATE municipios SET id_estado = 25 WHERE nombre IN ('Culiacán', 'Mazatlán', 'Los Mochis', 'Guasave', 'Navolato', 'Escuinapa', 'El Fuerte');
-- Sonora
UPDATE municipios SET id_estado = 26 WHERE nombre IN ('Hermosillo', 'Ciudad Obregón', 'Nogales', 'San Luis Río Colorado', 'Navojoa', 'Guaymas', 'Agua Prieta');
-- Tabasco
UPDATE municipios SET id_estado = 27 WHERE nombre IN ('Villahermosa', 'Cárdenas', 'Comalcalco', 'Macuspana', 'Huimanguillo', 'Jalpa de Méndez', 'Teapa');
```

Para los distritos de la Ciudad de México, el código actualiza la columna id_estado en la tabla distritos para asignarles el identificador del estado correspondiente a la Ciudad de México.
Para los municipios de cada estado, el código actualiza la columna id_estado en la tabla municipios para asignarles el identificador del estado al que pertenecen.

Esta asignación facilita la organización y consulta de la información en la base de datos, ya que permite identificar fácilmente a qué estado pertenece cada municipio y distrito

UPDATE: La palabra clave UPDATE se utiliza para modificar los registros existentes en una tabla de la base de datos.

SET: La cláusula SET especifica qué columnas de la tabla se van a actualizar y los valores que se les asignarán. En este caso, se utiliza para asignar valores nuevos a la columna id_estado.

WHERE: La cláusula WHERE se utiliza para especificar las condiciones que deben cumplir los registros que se van a actualizar. Solo se actualizarán los registros que cumplan estas condiciones.

IN: La función IN se utiliza para comparar un valor con una lista de posibles valores. En este contexto, se compara el valor de la columna nombre con una lista de nombres de municipios o distritos.

```
426    -- llenado de demarcaciones
427 •  INSERT INTO demarcacion (id_estado, id_municipio_o_estado) VALUES
428    (12,84),(28,188),(19,125),(3,17),(1,1),(23,155),(11,77),(9,6),(32,224),(21,145),(5,30),(20,138),(2,11),(29,201),(14,97),(22,154)
429    ,(30,209),(7,49),(10,70),(25,170),(17,113),(8,50);
```

INSERT INTO demarcacion: Indica que se van a insertar datos en la tabla llamada demarcacion.

(id_estado, id_municipio_o_estado) VALUES (...): Especifica los nombres de las columnas en las que se insertarán los datos, seguidos de la cláusula VALUES, que indica los valores que se van a insertar en esas columnas.

Los valores entre paréntesis (12, 84), (28, 188), ... son las filas de datos que se están insertando. Cada par de valores representa una fila, donde el primer valor es el id_estado y el segundo valor es el id_municipio_o_estado.

```
431 •  select * from candidato;
432 •  select * from documentacion;
433 •  select * from historial;
434 •  select * from demarcacion;
435 •  select * from cargo;
436 •  select * from partido_politico;
437 •  select * from coalicion;
438 •  select * from militancia;
```

Estas consultas te mostrarán los datos almacenados en cada una de las tablas que hemos indicado.

```
-- insercion de id foraneas en candidato
UPDATE candidato set id_partido =1, id_coalicion = 3, id_cargo = 5, id_demarcacion = 22, id_historial = 4, id_documentacion = 21 WHERE id_candidato =2;
UPDATE candidato set id_partido = 6, id_coalicion = 2 , id_cargo = 2, id_demarcacion = 1, id_historial = 7, id_documentacion = 12 WHERE id_candidato =23
UPDATE candidato set id_militancia = 1, id_cargo = 4, id_demarcacion = 6, id_historial = 14, id_documentacion = 4 WHERE id_candidato =24;
UPDATE candidato set id_partido =2, id_coalicion = 1, id_cargo = 5, id_demarcacion = 11, id_historial = 18, id_documentacion = 5 WHERE id_candidato =25;
UPDATE candidato set id_partido = 7, id_cargo = 2, id_demarcacion = 2, id_historial = 1, id_documentacion = 13 WHERE id_candidato =26;
UPDATE candidato set id_militancia = 1, id_cargo = 3, id_demarcacion = 18, id_historial = 8, id_documentacion = 3 WHERE id_candidato =27;
UPDATE candidato set id_partido = 7,id_cargo = 1, id_demarcacion = 13, id_historial = 6, id_documentacion = 20 WHERE id_candidato =28;
UPDATE candidato set id_partido = 7,id_cargo = 2, id_demarcacion = 19, id_historial = 9, id_documentacion = 19 WHERE id_candidato =29;
UPDATE candidato set id_militancia = 1, id_cargo = 5, id_demarcacion = 7, id_historial = 15, id_documentacion = 18 WHERE id_candidato =30;
UPDATE candidato set id_partido = 8, id_cargo = 2, id_demarcacion = 3, id_historial = 5, id_documentacion = 1 WHERE id_candidato =31;
UPDATE candidato set id_partido = 6, id_coalicion = 2, id_cargo = 5, id_demarcacion = 12, id_historial = 10, id_documentacion = 10 WHERE id_candidato =3
UPDATE candidato set id_partido = 7, id_cargo = 2, id_demarcacion = 14, id_historial = 19, id_documentacion = 17 WHERE id_candidato =33;
UPDATE candidato set id_partido = 8, id_cargo = 3, id_demarcacion = 8, id_historial = 2, id_documentacion = 2 WHERE id_candidato =34;
UPDATE candidato set id_partido = 8, id_cargo = 1, id_demarcacion = 4, id_historial = 16, id_documentacion = 9 WHERE id_candidato =35;
UPDATE candidato set id_partido = 8, id_cargo = 1, id_demarcacion = 17, id_historial = 11, id_documentacion = 7 WHERE id_candidato =36;
UPDATE candidato set id_militancia = 1, id_cargo = 5, id_demarcacion = 15, id_historial = 21, id_documentacion = 14 WHERE id_candidato =37;
UPDATE candidato set id_partido = 7, id_cargo = 4, id_demarcacion = 9, id_historial = 12, id_documentacion = 11 WHERE id_candidato =38;
UPDATE candidato set id_partido =3, id_coalicion = 1, id_cargo = 2, id_demarcacion = 21, id_historial = 17, id_documentacion = 16 WHERE id_candidato =39
UPDATE candidato set id_partido =4 , id_coalicion = 3, id_cargo = 5, id_demarcacion = 16, id_historial = 20, id_documentacion = 6 WHERE id_candidato =40
UPDATE candidato set id_militancia = 1, id_cargo = 1, id_demarcacion = 10, id_historial = 13, id_documentacion = 15 WHERE id_candidato =41;
UPDATE candidato set id_partido =2, id_coalicion = 1, id_cargo = 3, id_demarcacion = 5, id_historial = 3, id_documentacion = 8 WHERE id_candidato =42;
```

UPDATE candidato set ... WHERE id_candidato = Indica que se actualizará la tabla candidato, estableciendo ciertos valores para las columnas, y esto se aplicará a las filas donde el id_candidato coincida con el valor especificado.

id_partido = Asigna el identificador del partido político al que está afiliado el candidato.

id_coalicion = Asigna el identificador de la coalición a la que pertenece el partido político del candidato, en caso de que sea parte de una coalición.

id_cargo = Establece el cargo al que se está postulando el candidato.

id_demarcacion = Asigna el identificador de la demarcación electoral en la que el candidato está compitiendo.

id_historial = Establece el historial electoral del candidato, es decir, si ha sido electo o no en elecciones anteriores.

id_documentacion = Asigna el identificador de los documentos verificados que respaldan la candidatura del individuo.

id_militancia = En caso de ser relevante, asigna el identificador de la militancia del candidato en el partido político.

En resumen, estas instrucciones actualizan la información asociada con cada candidato en la tabla candidato, asegurando que tengan las claves foráneas adecuadas que los relacionen con partidos políticos, coaliciones, cargos, demarcaciones, historial electoral, documentación y, en algunos casos, militancia.

Consultas

```sql
-- queremos obtener información sobre los candidatos y sus afiliaciones partidarias
SELECT c.nombre, c.ap_paterno, c.ap_materno, p.nombre AS partido, co.nombre AS coalicion
FROM candidato c
LEFT JOIN partido_politico p ON c.id_partido = p.id_partido
LEFT JOIN coalicion co ON c.id_coalicion = co.id_coalicion;
```

SELECT c.nombre, c.ap_paterno, c.ap_materno, p.nombre AS partido, co.nombre AS coalicion: Selecciona las columnas nombre, ap_paterno y ap_materno de la tabla candidato, así como las columnas nombre de la tabla partido_politico y coalicion. El alias AS se utiliza para renombrar las columnas de salida como partido y coalicion respectivamente, para que sea más claro en el resultado qué representa cada columna.

FROM candidato c: Especifica que los datos se obtendrán de la tabla candidato y se le asigna un alias c.

LEFT JOIN partido_politico p ON c.id_partido = p.id_partido: Realiza una unión izquierda entre la tabla candidato y la tabla partido_politico utilizando la columna id_partido. Esto significa que se incluirán todos los candidatos, incluso aquellos que no estén afiliados a ningún partido político. Si un candidato está afiliado a un partido, se mostrará el nombre del partido; de lo contrario, la columna partido mostrará NULL.

LEFT JOIN coalicion co ON c.id_coalicion = co.id_coalicion: Realiza una unión izquierda entre la tabla candidato y la tabla coalicion utilizando la columna id_coalicion. Esto significa que se incluirán todos los candidatos, incluso aquellos que no estén afiliados a ninguna coalición. Si un candidato está afiliado a una coalición, se mostrará el nombre de la coalición; de lo contrario, la columna coalicion mostrará NULL.

```sql
-- muestra los candidatos que no tienen un partido político asignado.
SELECT nombre, ap_paterno, ap_materno
FROM candidato
WHERE id_partido IS NULL;
```

SELECT nombre, ap_paterno, ap_materno: Selecciona las columnas nombre, ap_paterno y ap_materno de la tabla candidato.

FROM candidato: Especifica que los datos se obtendrán de la tabla candidato.

WHERE id_partido IS NULL: Filtra las filas de la tabla candidato donde el valor de la columna id_partido es NULL. Esto significa que solo se seleccionarán los candidatos que no tienen asignado un partido político.

```sql
-- Muestra el número de candidatos por partido político
SELECT p.nombre AS partido, COUNT(c.id_candidato) AS total_candidatos
FROM partido_politico p
LEFT JOIN candidato c ON p.id_partido = c.id_partido
GROUP BY p.nombre;
```

SELECT p.nombre AS partido: Selecciona el nombre del partido político de la tabla partido_politico y lo nombra como "partido".

COUNT(c.id_candidato) AS total_candidatos: Cuenta el número de candidatos (id_candidato) de la tabla candidato y nombra esta cuenta como "total_candidatos".

FROM partido_politico p: Especifica que los datos se obtendrán de la tabla partido_politico y se la denomina con el alias "p".

LEFT JOIN candidato c ON p.id_partido = c.id_partido: Realiza una unión izquierda entre las tablas partido_politico y candidato basada en la coincidencia de los valores de id_partido, permitiendo que se incluyan partidos políticos aunque no tengan candidatos asociados.

GROUP BY p.nombre: Agrupa los resultados por el nombre del partido político.

```sql
-- Esta consulta une la tabla candidato con la tabla partido_politico para obtener información sobre el partido político al que está afiliado cada candidato.
SELECT c.nombre, c.ap_paterno, p.nombre AS partido
FROM candidato c
INNER JOIN partido_politico p ON c.id_partido = p.id_partido;
```

SELECT c.nombre, c.ap_paterno, p.nombre AS partido: Selecciona el nombre y los apellidos del candidato de la tabla candidato y el nombre del partido político de la tabla partido_politico, renombrando este último como "partido".

FROM candidato c: Especifica que los datos se obtendrán de la tabla candidato y se la denomina con el alias "c".

INNER JOIN partido_politico p ON c.id_partido = p.id_partido: Realiza una unión interna entre las tablas candidato y partido_politico basada en la coincidencia de los valores de id_partido, asegurando que solo se devuelvan registros que tengan coincidencias en ambas tablas.

```sql
-- Lista los municipios que no tienen un distrito asociado
SELECT m.nombre AS municipio
FROM municipios m
LEFT JOIN distritos d ON m.id_estado = d.id_estado
WHERE d.id_distrito IS NULL;
```

SELECT m.nombre AS municipio: Selecciona el nombre del municipio de la tabla municipios y lo renombra como "municipio".

FROM municipios m: Especifica que los datos se obtendrán de la tabla municipios y se la denomina con el alias "m".

LEFT JOIN distritos d ON m.id_estado = d.id_estado: Realiza una unión izquierda entre las tablas municipios y distritos basada en la coincidencia de los valores de id_estado, asegurando que se devuelvan todos los registros de la tabla municipios y sólo los registros coincidentes de la tabla distritos.

WHERE d.id_distrito IS NULL: Filtra las filas resultantes de la unión izquierda para incluir solo aquellas donde no haya un distrito asociado, es decir, donde el id_distrito de la tabla distritos sea NULL.

```sql
-- Mostrar los 5 estados que tienen el mayor número de habitantes
SELECT nombre AS estado
FROM estados
ORDER BY numero_habitantes DESC LIMIT 5;
```

SELECT nombre AS estado: Selecciona el nombre de los estados de la tabla estados y lo renombra como "estado".

FROM estados: Específica que los datos se obtendrán de la tabla estados.

ORDER BY numero_habitantes DESC: Ordena los resultados en orden descendente basado en el número de habitantes en la columna numero_habitantes.

LIMIT 5: Limita el resultado a solo los primeros 5 estados después de ordenarlos por el número de habitantes.

```
36    -- Mostrar la información de los candidatos
37 •  CREATE VIEW vista_candidatos AS
38    SELECT id_candidato, nombre, ap_paterno, ap_materno
39    FROM candidato;
40
41 •  SELECT * FROM vista_candidatos;
```

Esta consulta crea una vista que contiene las columnas id_candidato, nombre, ap_paterno y ap_materno de la tabla candidato.
Luego, se seleccionan todos los datos de la vista recién creada con la consulta:
Esta consulta selecciona todas las columnas de la vista vista_candidatos, lo que devolverá la información básica de todos los candidatos.

```
43    -- Mostrar la información del candidato junto con el partido asociado
44 •  CREATE VIEW vista_candidatos_partidos AS
45    SELECT c.nombre, c.ap_paterno, c.ap_materno, p.nombre AS partido
46    FROM candidato c
47    INNER JOIN partido_politico p ON c.id_partido = p.id_partido;
48
49 •  SELECT * FROM vista_candidatos_partidos;
```

CREATE VIEW vista candidatos partidos: Esta línea crea una vista llamada vista_candidatos_partidos.

SELECT c.nombre, c.ap_paterno, c.ap_materno, p.nombre AS partido: Esta parte de la consulta selecciona las columnas que se incluirán en la vista. c.nombre, c.ap_paterno y c.ap_materno son columnas de la tabla candidato, que representan el nombre, el apellido paterno y el apellido materno del candidato, respectivamente. p.nombre es una columna de la tabla partido_politico, que representa el nombre del partido político al que está asociado el candidato. El alias AS partido se utiliza para renombrar la columna p.nombre como partido.

FROM candidato c INNER JOIN partido politico p ON c.id partido = p.id partido: Esta parte de la consulta especifica las tablas que se van a unir y los criterios de unión. candidato c y partido_politico p son alias de las tablas candidato y partido_politico, respectivamente. Se realiza un INNER JOIN entre estas dos tablas utilizando la columna id_partido como criterio de unión.

SELECT * FROM vista_candidatos_partidos: Después de crear la vista, esta línea selecciona todos los datos de la vista vista_candidatos_partidos para mostrar la información del candidato junto con el nombre del partido político asociado. La consulta devolverá todas las filas y columnas de la vista.

```
51    -- Mostrar a los candidatos que pertenezcan al partido politico cuyo id_partido sea igual a 1
52    CREATE VIEW vista_candidatos_filtrados AS
53    SELECT id_candidato, nombre, ap_paterno, ap_materno
54    FROM candidato
55    WHERE id_partido = 1; -- Filtro por partido político específico
56
57    SELECT * FROM vista_candidatos_filtrados;
```

CREATE VIEW vista_candidatos_filtrados: Esto crea una vista llamada vista_candidatos_filtrados.

SELECT id_candidato, nombre, ap_paterno, ap_materno: Esta parte de la consulta selecciona las columnas id_candidato, nombre, ap_paterno y ap_materno de la tabla candidato. Estas columnas representan el ID del candidato, su nombre, apellido paterno y apellido materno, respectivamente.

FROM candidato: Especifica que los datos provienen de la tabla candidato.

WHERE id_partido = 1: Este es un filtro que limita los resultados a los candidatos que pertenecen al partido político con un id_partido igual a 1.

SELECT * FROM vista_candidatos_filtrados: Después de crear la vista, esta línea selecciona todos los datos de la vista vista_candidatos_filtrados para mostrar los candidatos que cumplen con el criterio especificado (en este caso, pertenecer al partido político con id_partido igual a 1). La consulta devolverá todas las filas y columnas de la vista que cumplen con el filtro.

```
59    -- Mostrar el número de candidatos por el id de cada partido
60    CREATE VIEW vista_candidatos_por_partido AS
61    SELECT id_partido, COUNT(*) AS total_candidatos
62    FROM candidato
63    GROUP BY id_partido;
64
65    SELECT * FROM vista_candidatos_por_partido;
```

CREATE VIEW vista_candidatos_por_partido: Esto crea una vista llamada vista_candidatos_por_partido.

SELECT id_partido, COUNT(*) AS total_candidatos: Esta parte de la consulta selecciona el id_partido de la tabla candidato y cuenta el número de registros para cada id_partido. El COUNT(*) cuenta el número de filas y el resultado se almacena en la columna total_candidatos.

FROM candidato: Especifica que los datos provienen de la tabla candidato.

GROUP BY id_partido: Esto agrupa los resultados por el id_partido, de modo que se cuenten los candidatos para cada partido político.

SELECT * FROM vista_candidatos_por_partido: Después de crear la vista, esta línea selecciona todos los datos de la vista vista_candidatos_por_partido para mostrar el número de candidatos por cada id_partido. La consulta devolverá una fila por cada id_partido, mostrando el número total de candidatos para cada uno.

```
67    -- Mostrar todos los datos que tiene la tabla candidatos mediante una vista
68    CREATE VIEW vista_candidatos_ordenados AS
69    SELECT *
70    FROM candidato
71    ORDER BY ap_paterno, ap_materno, nombre;
72
73    SELECT * FROM vista_candidatos_ordenados;
```

CREATE VIEW vista_candidatos_ordenados: Esto crea una vista llamada vista_candidatos_ordenados.

SELECT * FROM candidato ORDER BY ap_paterno, ap_materno, nombre: Esta parte de la consulta selecciona todos los datos de la tabla candidato y los ordena por apellido paterno, apellido materno y nombre en orden ascendente.

SELECT * FROM vista_candidatos_ordenados: Después de crear la vista, esta línea selecciona todos los datos de la vista vista_candidatos_ordenados para mostrar todos los datos de la tabla candidato ordenados como se especificó en la vista.

```
75    -- Mostrar a los candidatos con su respectivo partido político no mostrando su id
76    CREATE VIEW vista_candidatos_con_partidos AS
77    SELECT id_candidato, nombre, ap_paterno, ap_materno,
78           (SELECT nombre FROM partido_politico WHERE id_partido = candidato.id_partido) AS partido
79    FROM candidato;
80
81    SELECT * FROM vista_candidatos_con_partidos;
```

CREATE VIEW vista_candidatos_con_partidos: Crea una vista llamada vista_candidatos_con_partidos.

SELECT id_candidato, nombre, ap_paterno, ap_materno, (SELECT nombre FROM partido_politico WHERE id_partido = candidato.id_partido) AS partido FROM candidato: Esta consulta selecciona el ID del candidato, su nombre y apellidos, y utiliza una subconsulta para obtener el nombre del partido político correspondiente al ID del partido político en la tabla partido_politico. El resultado de la subconsulta se alía con el alias partido.

SELECT * FROM vista_candidatos_con_partidos: Después de crear la vista, esta línea selecciona todos los datos de la vista vista_candidatos_con_partidos para mostrar a los candidatos con su respectivo partido político, pero sin mostrar el ID del partido político.

```
83    -- Mostrar al candidato con su partido y coalicion
84    CREATE VIEW vista_candidatos_completos AS
85    SELECT c.nombre, c.ap_paterno, c.ap_materno, p.nombre AS partido, co.nombre AS coalicion
86    FROM candidato c
87    LEFT JOIN partido_politico p ON c.id_partido = p.id_partido
88    LEFT JOIN coalicion co ON c.id_coalicion = co.id_coalicion;
89
90    SELECT * FROM vista_candidatos_completos;
```

CREATE VIEW vista_candidatos_completos: Define una vista llamada vista_candidatos_completos.

SELECT c.nombre, c.ap_paterno, c.ap_materno, p.nombre AS partido, co.nombre AS coalicion FROM candidato c LEFT JOIN partido_politico p ON c.id_partido = p.id_partido LEFT JOIN coalicion co ON c.id_coalicion = co.id_coalicion: Esta consulta selecciona el nombre y apellidos del candidato de la tabla candidato, el nombre del partido político de la tabla

partido_politico, y el nombre de la coalición de la tabla coalicion, utilizando LEFT JOIN para unir las tablas de candidato con las de partido político y coalición.

SELECT * FROM vista_candidatos_completos: Luego, esta línea selecciona todos los datos de la vista vista_candidatos_completos para mostrar al candidato con su partido político y coalición.

```
92    -- Calcular la edad del candidato mediante la fecha registrada
93    CREATE VIEW vista_candidatos_edad AS
94    SELECT nombre, ap_paterno, ap_materno, fnac, YEAR(CURDATE()) - YEAR(fnac) AS edad
95    FROM candidato;
96
97    SELECT * FROM vista_candidatos_edad;
```

CREATE VIEW vista_candidatos_edad: Define una vista llamada vista_candidatos_edad.

SELECT nombre, ap_paterno, ap_materno, fnac, YEAR(CURDATE()) - YEAR(fnac) AS edad FROM candidato: Esta consulta selecciona el nombre y apellidos del candidato, así como su fecha de nacimiento (fnac). Calcula la edad restando el año actual (YEAR(CURDATE())) del año de nacimiento (YEAR(fnac)).

SELECT * FROM vista_candidatos_edad: Luego, esta línea selecciona todos los datos de la vista vista_candidatos_edad para mostrar el nombre, apellidos, fecha de nacimiento y edad de los candidatos.

```
99    -- Estados que tengan mas de 10000
100   SELECT nombre FROM estados
101   UNION
102   SELECT nombre FROM municipios WHERE num_habitantes > 10000;
```

SELECT nombre FROM estados: Selecciona el nombre de todos los estados.

UNION: Combina los resultados de la primera consulta con los resultados de la siguiente consulta, eliminando duplicados.

SELECT nombre FROM municipios WHERE num_habitantes > 10000: Selecciona el nombre de los municipios que tienen más de 10,000 habitantes.

```sql
-- Mostrar todos los candidatos que sean de la Ciudad de Mexico
SELECT *
FROM candidato
WHERE direccion = 'Mexico City';
```

La cláusula WHERE filtra las filas de la tabla candidato donde la columna direccion es igual a 'Mexico City', lo que devuelve todos los candidatos que tienen su dirección en la Ciudad de México.

```sql
-- Vista que muestra los datos del candidato mas el estado
DROP VIEW IF EXISTS vista_candidatos_por_estado;
CREATE VIEW vista_candidatos_por_estado AS
SELECT c.id_candidato 'Id', c.nombre 'Nombre', c.ap_paterno'Apellido Paterno', c.ap_materno'Apellido Materno',
e.nombre 'Estado',c.fnac 'Fecha de Nacimiento', c.telefono, c.email
FROM candidato c
LEFT JOIN demarcacion d ON c.id_demarcacion = d.id_demarcacion
LEFT JOIN estados e ON d.id_estado = e.id_estado;

SELECT * FROM vista_candidatos_por_estado;
```

Crea una vista llamada vista_candidatos_por_estado que muestra información sobre los candidatos junto con el estado al que están asociados. Aquí está el significado de cada parte del script:
DROP VIEW IF EXISTS vista_candidatos_por_estado;: Esta línea elimina la vista vista_candidatos_por_estado si ya existe. El IF EXISTS evita errores si la vista no existe.

CREATE VIEW vista_candidatos_por_estado AS ...: Esta línea crea la vista vista_candidatos_por_estado. Una vista es una tabla virtual que se compone de filas y columnas como una tabla real en una base de datos. La vista se define como una consulta SQL almacenada que se ejecuta cada vez que se accede a la vista.

SELECT ...: Esta es la consulta SQL que define los datos que se mostrarán en la vista. En este caso, la consulta selecciona varias columnas de la tabla candidato (alias c) y la tabla estados (alias e). Utiliza las cláusulas LEFT JOIN para combinar las tablas candidato, demarcacion, y estados basadas en sus claves primarias y foraneas.

FROM candidato c: Esto especifica que los datos se seleccionan de la tabla candidato y se le asigna un alias c.

LEFT JOIN demarcacion d ON c.id_demarcacion = d.id_demarcacion: Esta cláusula une la tabla demarcacion (alias d) con la tabla candidato (c) utilizando la columna id_demarcacion.

LEFT JOIN estados e ON d.id_estado = e.id_estado: Esta cláusula une la tabla estados (alias e) con la tabla demarcacion (d) utilizando la columna id_estado.

Las columnas seleccionadas en la consulta (c.id_candidato, c.nombre, c.ap_paterno, c.ap_materno, e.nombre, c.fnac, c.telefono, c.email) se mostrarán como columnas en la vista.

```
127     Vista que muestra los datos del candidato mas el cargo solicitado
128   OP VIEW IF EXISTS vista_candidatos_por_cargo_solicitado;
129   EATE VIEW vista_candidatos_por_cargo_solicitado AS
130   LECT c.id_candidato 'id',c.nombre 'Nombre',c.ap_paterno 'Apellido Paterno',c.ap_materno 'Apellido Materno',
131   cargo_solicitado 'Cargo Solicitado',c.fnac 'Fecha de Nacimiento',c.telefono,c.email
132   OM candidato c
133   FT JOIN historial h ON c.id_historial = h.id_historial;
134
135   LECT * FROM vista_candidatos_por_cargo_solicitado;
```

Crea una vista llamada vista_candidatos_por_cargo_solicitado que muestra información sobre los candidatos junto con el cargo que han solicitado.
DROP VIEW IF EXISTS vista_candidatos_por_cargo_solicitado;: Esta línea elimina la vista vista_candidatos_por_cargo_solicitado si ya existe. El IF EXISTS evita errores si la vista no existe.

CREATE VIEW vista candidatos por cargo solicitado AS ...: Esta línea crea la vista vista_candidatos_por_cargo_solicitado. Una vista es una tabla virtual que se compone de filas y columnas como una tabla real en una base de datos. La vista se define como una consulta SQL almacenada que se ejecuta cada vez que se accede a la vista.

SELECT ...: Esta es la consulta SQL que define los datos que se mostrarán en la vista. En este caso, la consulta selecciona varias columnas de la tabla candidato (alias c) y la tabla historial (alias h). Utiliza la cláusula LEFT JOIN para combinar las tablas candidato y historial basadas en sus claves primarias y extranjeras.

FROM candidato c: Esto especifica que los datos se seleccionan de la tabla candidato y se le asigna un alias c.

LEFT JOIN historial h ON c.id_historial = h.id_historial: Esta cláusula une la tabla historial (alias h) con la tabla candidato (c) utilizando la columna id_historial.

Las columnas seleccionadas en la consulta (c.id_candidato, c.nombre, c.ap_paterno, c.ap_materno, h.cargo_solicitado, c.fnac, c.telefono, c.email) se mostrarán como columnas en la vista.

```
137     Vista que muestra la informacion mas la documentacion del candidato
138 ■   OP VIEW IF EXISTS vista_documentacion_candidatos;
139 ■   EATE VIEW vista_documentacion_candidatos AS
140     LECT c.id_candidato 'Id',c.nombre 'Nombre',c.ap_paterno'Apellido Paterno',c.ap_materno'Apellido Materno',
141     c.documento 'Documento',doc.verificacion_eligibilidad 'Verificacion',doc.declaracion_interes 'Declaracion'
142     OM candidato c
143     FT JOIN documentacion doc ON c.id_documentacion = doc.id_documentacion;
144
145 ■   LECT * FROM vista_documentacion_candidatos;
```

crea una vista llamada vista_documentacion_candidatos que muestra la información de documentación asociada a cada candidato. Aquí tienes una explicación de cada parte del script:

DROP VIEW IF EXISTS vista_documentacion_candidatos;: Esta línea elimina la vista vista_documentacion_candidatos si ya existe. El IF EXISTS evita errores si la vista no existe.

CREATE VIEW vista_documentacion_candidatos AS ...: Esta línea crea la vista vista_documentacion_candidatos. Una vista es una tabla virtual que se compone de filas y columnas como una tabla real en una base de datos. La vista se define como una consulta SQL almacenada que se ejecuta cada vez que se accede a la vista.

SELECT ...: Esta es la consulta SQL que define los datos que se mostrarán en la vista. En este caso, la consulta selecciona varias columnas de la tabla candidato (alias c) y la tabla documentacion (alias doc). Utiliza la cláusula LEFT JOIN para combinar las tablas candidato y documentacion basadas en sus claves primarias y extranjeras.

FROM candidato c: Esto especifica que los datos se seleccionan de la tabla candidato y se le asigna un alias c.

LEFT JOIN documentacion doc ON c.id_documentacion = doc.id_documentacion: Esta cláusula une la tabla documentacion (alias doc) con la tabla candidato (c) utilizando la columna id_documentacion.

Las columnas seleccionadas en la consulta (c.id_candidato, c.nombre, c.ap_paterno, c.ap_materno, doc.documento, doc.verificacion_eligibilidad, doc.declaracion_interes) se mostrarán como columnas en la vista.

```sql
-- Vista que muestra toda la informacion detallada del candidato
DROP VIEW IF EXISTS vista_candidatos_detallada;
CREATE VIEW vista_candidatos_detallada AS
SELECT c.id_candidato 'Id',c.rfc 'RFC',concat(c.nombre,' ',c.ap_paterno ,' ',c.ap_materno ) 'Nombre Completo',
c.fnac 'Fecha de Nacimiento',c.direccion 'Direccion',c.telefono 'Telefono',c.email 'Email',c.propuestas 'Propuestas',
c.finscrip 'Fecha Inscripcion',p.nombre 'Partido Politico',co.nombre 'Coalicion',m.nombre 'Militancia',
ca.nombre_cargo 'Cargo',e.nombre 'Estado',d.id_municipio_o_estado 'Munipio o distrito',h.cargo_solicitado 'Cargo Solicitado Anteriormente',
h.resultado 'Resultado',h.fecha_eleccion 'Fecha de Eleccion',doc.documento 'Documento',doc.verificacion_eligibilidad 'Verificación',
doc.declaracion_interes 'Declaracion de Interes'
FROM candidato c
LEFT JOIN partido_politico p ON c.id_partido = p.id_partido
LEFT JOIN coalicion co ON c.id_coalicion = co.id_coalicion
LEFT JOIN militancia m ON c.id_militancia = m.id_militancia
LEFT JOIN cargo ca ON c.id_cargo = ca.id_cargo
LEFT JOIN demarcacion d ON c.id_demarcacion = d.id_demarcacion
LEFT JOIN estados e ON d.id_estado = e.id_estado
LEFT JOIN historial h ON c.id_historial = h.id_historial
LEFT JOIN documentacion doc ON c.id_documentacion = doc.id_documentacion;

SELECT * FROM vista_candidatos_detallada;
```

crea una vista llamada vista_candidatos_detallada que proporciona una visión detallada de la información de los candidatos, incluyendo información sobre partido político, coalición, militancia, cargo, demarcación, historial y documentación asociada. Aquí tienes una explicación de cada parte del script:

DROP VIEW IF EXISTS vista_candidatos_detallada;: Esta línea elimina la vista vista_candidatos_detallada si ya existe. El IF EXISTS evita errores si la vista no existe.

CREATE VIEW vista_candidatos_detallada AS ...: Esta línea crea la vista vista_candidatos_detallada. Una vista es una tabla virtual que se compone de filas y columnas como una tabla real en una base de datos. La vista se define como una consulta SQL almacenada que se ejecuta cada vez que se accede a la vista.

SELECT ...: Esta es la consulta SQL que define los datos que se mostrarán en la vista. En este caso, la consulta selecciona varias columnas de varias tablas y utiliza la cláusula LEFT JOIN para combinar las tablas basadas en sus claves primarias y extranjeras.

Las columnas seleccionadas en la consulta incluyen información detallada sobre los candidatos, como su RFC, nombre completo, fecha de nacimiento, dirección, teléfono, correo electrónico, propuestas, fecha de inscripción, partido político, coalición, militancia, cargo, estado, municipio o distrito, cargo solicitado anteriormente, resultado, fecha de elección, documento, verificación y declaración de interés.