

Laboratorio Sistemas de Bases de Datos 1



PROYECTO 1

MANUAL TECNICO

FECHA: 17/09/2023

Juan Josue Zuleta Beb
Carné: 202006353

Introducción

Este informe se adentra en los aspectos técnicos esenciales que sustentan nuestra aplicación, la cual se ha concebido para gestionar y procesar de manera efectiva la información relacionada con las votaciones, obtenida a través de archivos CSV. Este proyecto se ha apoyado en dos tecnologías clave: MySQL y Node.js, que han sido cuidadosamente seleccionadas para garantizar la eficacia y robustez de la aplicación. A lo largo de este informe, exploraremos en detalle cómo estas tecnologías se han integrado de manera sinérgica para crear una solución que no solo facilita el manejo de datos electorales, sino que también optimiza el proceso de adquisición, almacenamiento y análisis de información proveniente de archivos Excel en formato CSV. Además, examinaremos las ventajas y características que hacen de esta aplicación una herramienta versátil y poderosa para la gestión de datos de votaciones.

Objetivos

Se busca proporcionar a los usuarios las herramientas necesarias para que puedan instalar, comprender y utilizar eficazmente nuestra aplicación. A través de una guía completa de instalación, una descripción detallada de archivos relevantes y una explicación minuciosa de los métodos de implementación, buscamos garantizar una experiencia fluida y exitosa para quienes interactúen con nuestro sistema.

Facilitar la Instalación y Comprensión: Nuestro primer objetivo es proporcionar una guía exhaustiva que instruya a los usuarios sobre la instalación adecuada de la aplicación, así como sobre la comprensión del código subyacente. Esto implica detallar los pasos necesarios para configurar la aplicación de manera efectiva, asegurando que los usuarios puedan acceder y utilizar la aplicación sin problemas desde el inicio. Además, se brindará una descripción minuciosa de los componentes y archivos esenciales del sistema, lo que ayudará a los usuarios a orientarse en la configuración y comprender la estructura de la aplicación.

Métodos de Implementación: Nuestro segundo objetivo se centra en la implementación efectiva de métodos que garanticen un acceso oportuno y adecuado a la aplicación. Exploraremos los métodos específicos utilizados en la inicialización del sistema, brindando una visión detallada de cómo se integran en el proceso general. Esto permitirá a los usuarios comprender no solo el qué, sino también el cómo detrás de la inicialización de la aplicación.

Requisitos Mínimos del Sistema

Sistema operativo 64 bits

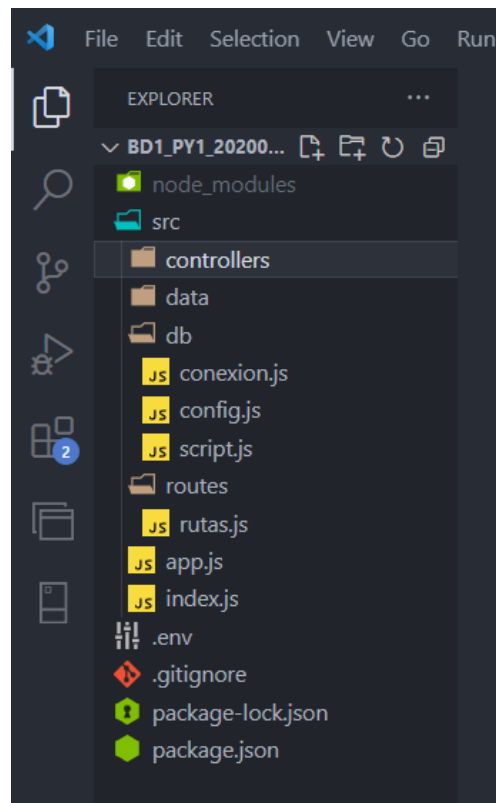
- Microsoft Windows 10/11 (incl.64-bit)
- macOS 12 o superior
- Linux 11 o superior (Cualquier distribución)
- Procesador a 2.1 GHz o superior
- 4 GB (64 bits) de RAM
- 3 GB de espacio disponible en el disco duro
- Tarjeta de vídeo compatible con DirectX 9 o superior.
- Navegador web (Recomendado: Google Chrome)

Software (Indispensable tenerlo instalado)

- IDE: Visual Studio Code
Puede conseguirse en: <https://code.visualstudio.com/>
- Node JS:
Puede conseguirse en: <https://nodejs.org/en>
- MySQL:
Puede conseguirse en: <https://www.mysql.com/>
- Insomnia/Postman:
Puede conseguirse en: <https://insomnia.rest/>
<https://www.postman.com/>

Estructura raíz:

El proyecto tiene la siguiente estructura de directorios:

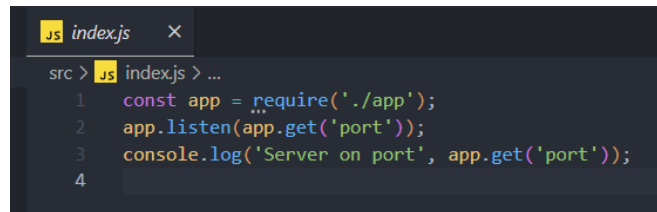


Directorio de la App: **src**

El archivo app.js configura una aplicación Express básica con soporte para solicitudes JSON, habilita CORS, configura un puerto para escuchar las solicitudes entrantes y monta las rutas definidas en el archivo './routes/rutas'.

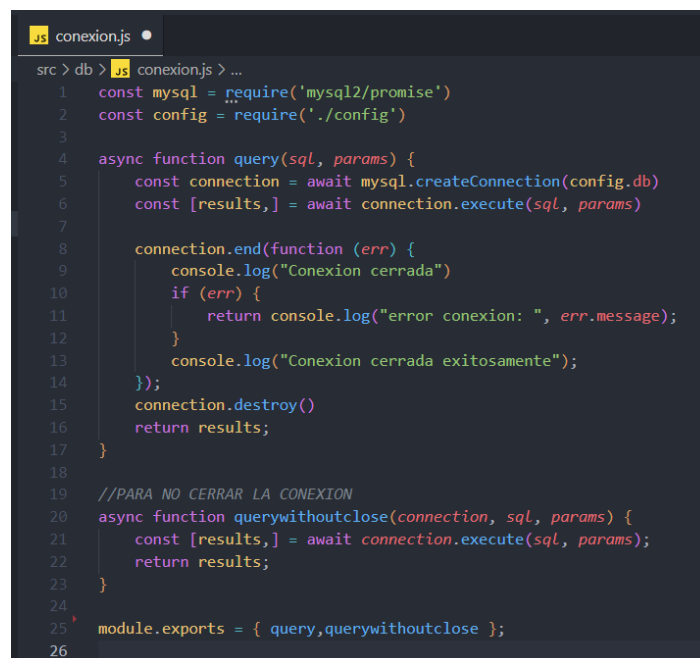
```
src > JS app.js > ...
1  const express = require('express');
2  const cors = require('cors');
3  require('dotenv').config();
4  const app = express();
5  //configuracion
6  app.set('port', process.env.SERVER_PORT || 3000);
7
8  //middlewares
9  app.use(express.json());
10 app.use(cors());
11
12 //routes
13 app.use(require('./routes/rutas'));
14
15 module.exports = app;
16
```

El archivo index.js se encarga de iniciar el servidor de la aplicación Express configurada en el archivo app.js. Una vez que el servidor está en funcionamiento, muestra un mensaje en la consola que indica en qué puerto se está ejecutando la aplicación. Esto marca el punto de inicio de la aplicación web y permite que las solicitudes HTTP sean manejadas por la instancia de Express configurada en app.js.



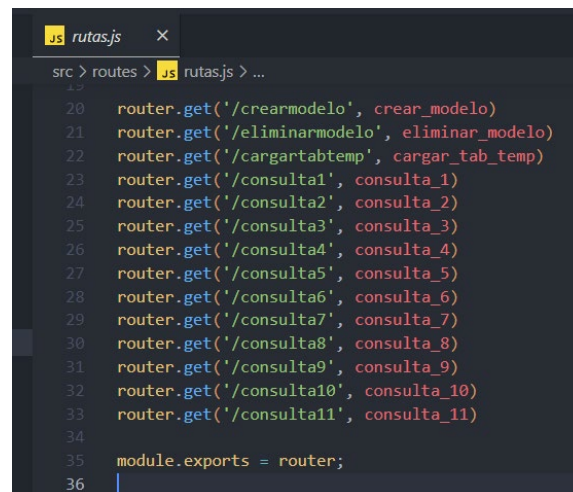
```
JS index.js X
src > JS index.js > ...
1  const app = require('./app');
2  app.listen(app.get('port'));
3  console.log('Server on port', app.get('port'));
4
```

El archivo conexion.js encapsula la funcionalidad de conexión a una base de datos MySQL utilizando mysql2/promise. Proporciona funciones para ejecutar consultas SQL y manejar la apertura y cierre de conexiones de manera adecuada, lo que facilita el acceso a la base de datos desde otros módulos de la aplicación.



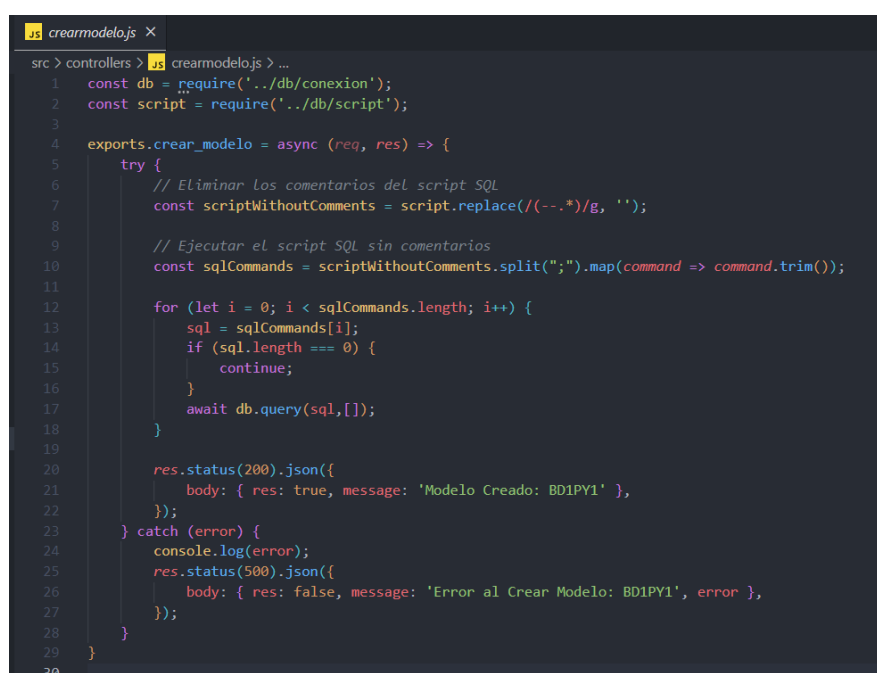
```
JS conexion.js
src > db > JS conexion.js > ...
1  const mysql = require('mysql2/promise')
2  const config = require('./config')
3
4  async function query(sql, params) {
5      const connection = await mysql.createConnection(config.db)
6      const [results,] = await connection.execute(sql, params)
7
8      connection.end(function (err) {
9          console.log("Conexion cerrada")
10         if (err) {
11             return console.log("error conexion: ", err.message);
12         }
13         console.log("Conexion cerrada exitosamente");
14     });
15     connection.destroy()
16     return results;
17 }
18
19 //PARA NO CERRAR LA CONEXION
20 async function querywithoutclose(connection, sql, params) {
21     const [results,] = await connection.execute(sql, params);
22     return results;
23 }
24
25 module.exports = { query, querywithoutclose };
26
```

El archivo rutas.js se utiliza para definir las rutas de una aplicación Express y asociar cada ruta con su respectivo controlador. Cuando un usuario accede a una de estas rutas, el controlador correspondiente se ejecuta para manejar la solicitud y realizar las acciones necesarias, como consultas a la base de datos o procesamiento de datos. Esto ayuda a organizar y modularizar el código de la aplicación web de manera efectiva.



```
src > routes > js rutas.js > ...
20 router.get('/crearmodelo', crear_modelo)
21 router.get('/eliminarmodelo', eliminar_modelo)
22 router.get('/cargartabtemp', cargar_tab_temp)
23 router.get('/consulta1', consulta_1)
24 router.get('/consulta2', consulta_2)
25 router.get('/consulta3', consulta_3)
26 router.get('/consulta4', consulta_4)
27 router.get('/consulta5', consulta_5)
28 router.get('/consulta6', consulta_6)
29 router.get('/consulta7', consulta_7)
30 router.get('/consulta8', consulta_8)
31 router.get('/consulta9', consulta_9)
32 router.get('/consulta10', consulta_10)
33 router.get('/consulta11', consulta_11)
34
35 module.exports = router;
36
```

El archivo crearmodelo.js maneja la creación de un modelo en una base de datos utilizando un script SQL previamente definido. Limpia el script de comentarios, divide los comandos SQL y los ejecuta en la base de datos. Luego, envía una respuesta HTTP con el resultado de la operación.



```
src > controllers > js crearmodelo.js > ...
1 const db = require('../db/conexion');
2 const script = require('../db/script');
3
4 exports.crear_modelo = async (req, res) => {
5   try {
6     // Eliminar los comentarios del script SQL
7     const scriptWithoutComments = script.replace(/(--.*)/g, '');
8
9     // Ejecutar el script SQL sin comentarios
10    const sqlCommands = scriptWithoutComments.split(";").map(command => command.trim());
11
12    for (let i = 0; i < sqlCommands.length; i++) {
13      sql = sqlCommands[i];
14      if (sql.length === 0) {
15        continue;
16      }
17      await db.query(sql, []);
18    }
19
20    res.status(200).json({
21      body: { res: true, message: 'Modelo Creado: BD1PY1' },
22    });
23  } catch (error) {
24    console.log(error);
25    res.status(500).json({
26      body: { res: false, message: 'Error al Crear Modelo: BD1PY1', error },
27    });
28  }
29 }
30
```

Para la carga masiva de datos desde archivos CSV en una base de datos MySQL. Se crea una serie de tablas temporales para almacenar temporalmente los datos antes de transferirlos a las tablas permanentes. Cada paso se ejecuta de manera asincrónica y se manejan los errores en caso de que ocurran. Este controlador es parte de una aplicación Express y se accede a él a través de una ruta específica.

```
// Cargar datos de ciudadanos
const datosClientes = fs.readFileSync(filePath_Ciudadanos, 'utf-8');
const lines = datosClientes.split('\n');
for (let i = 1; i < lines.length; i++) {
  const fields = lines[i].split(',');
  const dpi = fields[0];
  const nombre = fields[1];
  const apellido = fields[2];
  const direccion = fields[3];
  const telefono = fields[4];
  const edad = fields[5];
  const genero = fields[6];

  // Insertar los datos en la tabla temporal
  await db.queryWithoutClose(connection, `INSERT INTO BD1PY1.CIUDADANOTMP
}
```

Ejemplo de función empleada para extraer los datos del archivo csv.

Para la eliminación del modelo únicamente se envía un script SQL medio de una función asíncrona la cual destruye la base de datos y todas sus tablas. Este mismo método se utiliza para realizar consultas en la base de datos enviando scripts SQL con las respectivas query para obtener las tablas con los resultados deseados.

```
exports.consulta_1 = async (req, res) => {

  const scriptConsulta = `

  -- NOMBRE DE LOS CANDIDATOS A PRESIDENTE Y VICEPRESIDENTE POR PARTIDO

  SELECT
    PRES.nombres AS Presidente,
    VICE.nombres AS Vicepresidente,
    PART.siglas AS Partido
  FROM
    bd1py1.candidato PRES
    INNER JOIN
    bd1py1.candidato VICE ON PRES.id_partido = VICE.id_partido
    AND PRES.id_cargo = 1
    AND VICE.id_cargo = 2
    INNER JOIN
    bd1py1.partido PART ON PRES.id_partido = PART.id_partido;

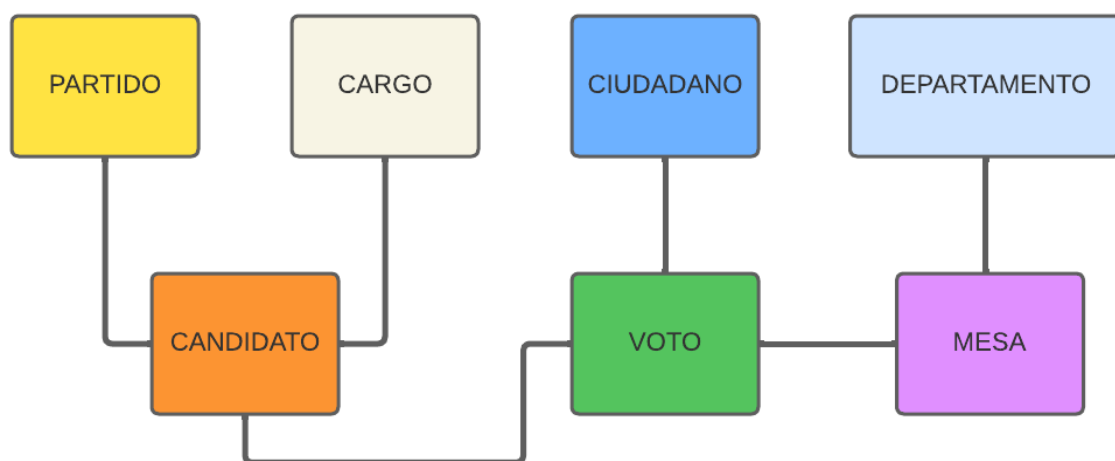
  `;
}
```


En el proceso de diseño de la base de datos, se aplicaron tres modelos fundamentales que jugaron un papel esencial en la definición y configuración de la estructura de la base de datos.

Estos modelos fueron empleados de manera estratégica para garantizar una distribución eficiente de los datos y una organización coherente de la información.

Modelo Conceptual:

Iniciando con el modelo conceptual, la cual es una representación de alto nivel de la estructura de la base de datos y s centra en las entidades y sus relaciones.



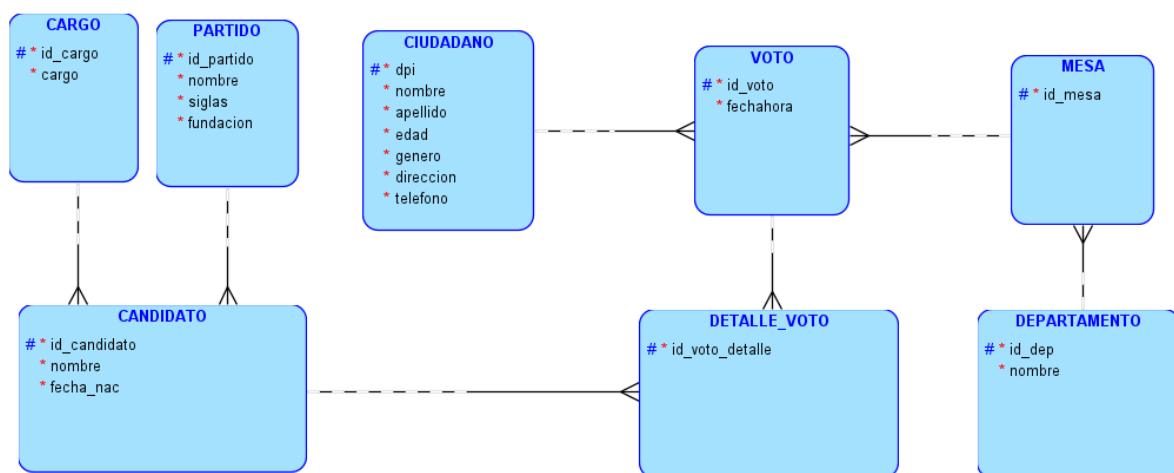
Modelo Conceptual: Proyecto 1

De este modelo se deduce la necesidad de cada entidad y su respectiva relación con las otras entidades, esto presenta una nueva necesidad, la cual es determinar las relaciones a nivel de atributos de entidades para lo cual se diseña el modelo lógico.

Modelo Lógico:

En esta etapa se tradujo el modelo conceptual en un modelo mas detallado que se centra en la estructura de la base de datos sin considerar la plataforma de implementación específica.

Se utilizo Oracle Developer Data Modeler para definir las tablas y las columnas, incluyendo los datos y las restricciones en los atributos obligatorios y los definidos como clave primaria y clave foránea.



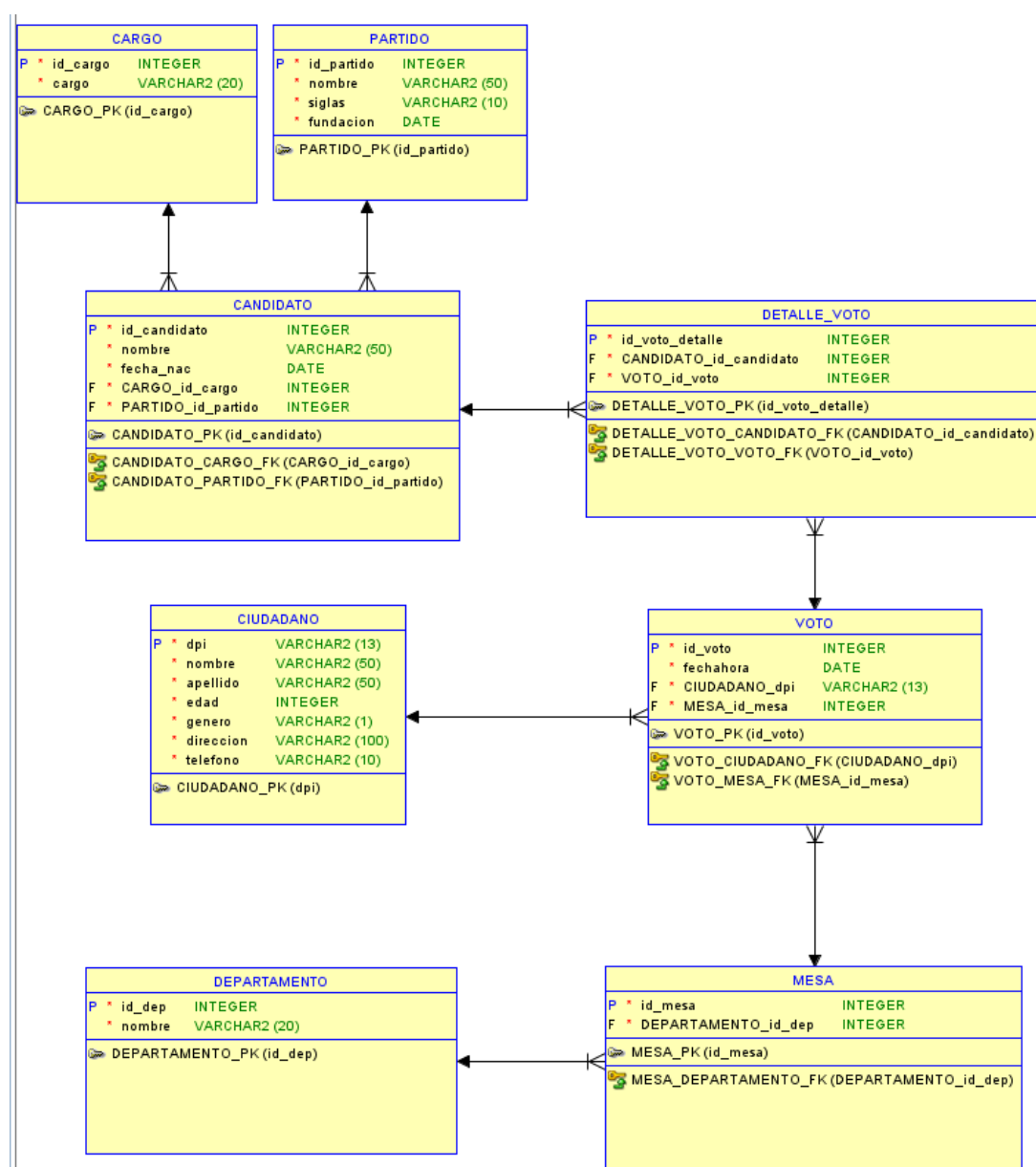
Modelo Logico: Proyecto 1

Analizando los datos proporcionados para la base de datos se determinó que había datos que podrían causar problemas por las llaves primarias en la entidad voto, esto debido a que los datos de algunos atributos no eran atómicos, por lo cual se decidió realizar una nueva entidad llamada detalle voto para almacenar por medio del identificador primario del voto los datos repetidos en una nueva entidad, de esta forma se realiza la Primera forma normal (1FN) la cual trata de asegurarse de que los datos estén organizados en tablas de manera que cada valor sea atómico y no haya duplicación de datos en una columna, mientras se utiliza una clave primaria para identificar de manera única cada registro. Esto establece una base sólida para la normalización adicional y la eficiencia de las consultas en una base de datos.

Modelo Físico:

En esta etapa se tradujo el modelo lógico en un modelo físico considerando la plataforma de implementación, como MySQL, en la cual fue implementada.

Este modelo nos permite reconocer como realmente están relacionadas las entidades, como funcionan las claves primarias y las llaves foráneas. En este modelo se consideran las opciones de seguridad y control de acceso.



Modelo Físico: Proyecto 1