

Parcial 1

Ingrid Echeverri Montoya 1943542-2711

22 de diciembre de 2021

Bases de datos 1 – 80

Jefferson A. Peña Torres

Tecnología en Sistemas de Información

Universidad del Valle

Tabla de contenido

Introducción.....	3
Links.....	3
Desarrollo Modelos	3
Modelo Entidad-Relación.....	3
Primera versión del diagrama entidad-relación	4
Modelo Relacional	4
Segunda versión del diagrama entidad-relación	5
Tercera versión del diagrama entidad-relación	6
Cuarta versión del diagrama entidad-relación.....	7
Codigo SQL.....	8
Dependencias funcionales y Normalización.....	10
Despliegue	12
Backend – FrontEnd	14
Webgrafía.....	26

Introducción

En este documento se demuestra el desarrollo del problema pedido en el parcial 1 de bases de datos. El problema dado es crear una aplicación web llamada Attendance, la cual permite a los estudiantes y personal marcar la asistencia a clase o sede y permite a los administradores crear cursos y agregar detalles del personal incluyendo estudiantes.

Links

Youtube: <https://youtu.be/OEhgt36RHmw>

Github: <https://github.com/Ingrid-E/Attendance>

Desarrollo Modelos

Modelo Entidad-Relación

Para comenzar la aplicación Attendance primero se determinan las **entidades** con sus **atributos** y las **relaciones** según el enunciado dado:

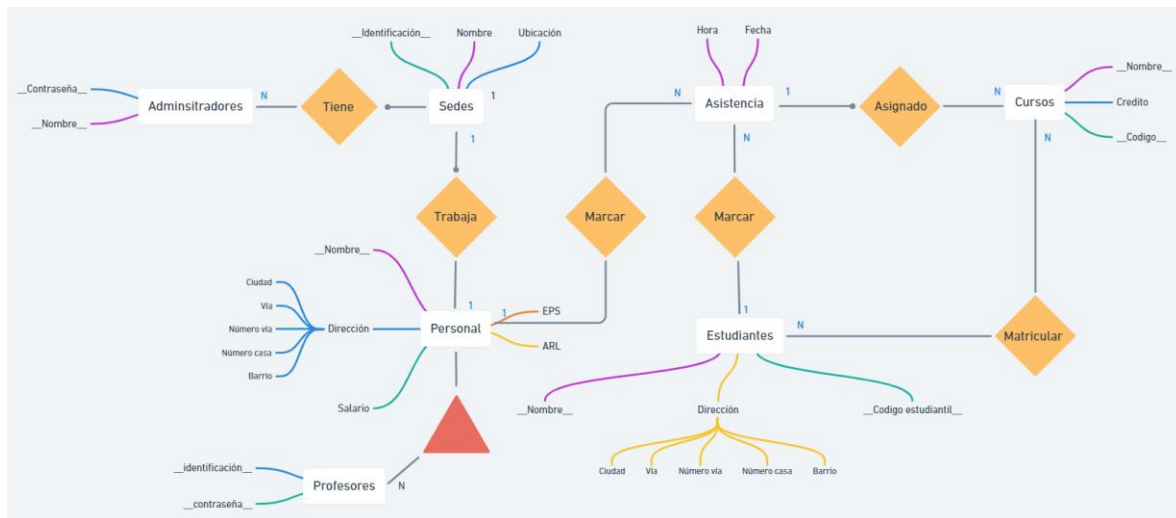
“Un **administrador** tiene un **nombre** y una **contraseña** con los que puede acceder a la aplicación. Este puede **tener** varias **sedes** de las cuales se conoce un **nombre**, una **ubicación** y una **identificación única**. Las sedes tienen un **personal** asociado que se **encarga de lo misional y no misional** de la institución educativa. Es por esta razón que se ha considerado, **el salario**, **la dirección**, **la EPS**, **el ARL** y el **nombre completo** de cada persona. Los **profesores** hacen parte de los miembros del personal, pero no los estudiantes. Cada **estudiante** está **matriculado** en uno o varios **cursos**, de ellos se conoce la **dirección**, el **nombre completo** y el **código de estudiante**.”

Todo el personal y los estudiantes **marcan su asistencia a la sede**. La **asistencia** mantiene información de la **hora** y **fecha** en que se registra la asistencia”

Una vez identificado las entidades se comienza a crear el diagrama entidad-relación.

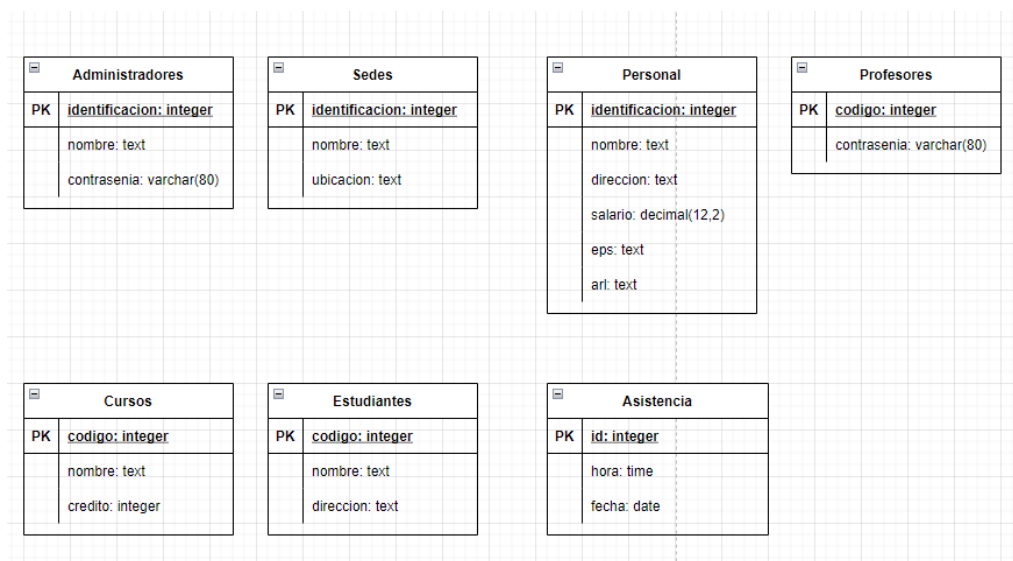
Para este se utilizó la herramienta whimsical para organizarlo de mejor forma.

Primera versión del diagrama entidad-relación

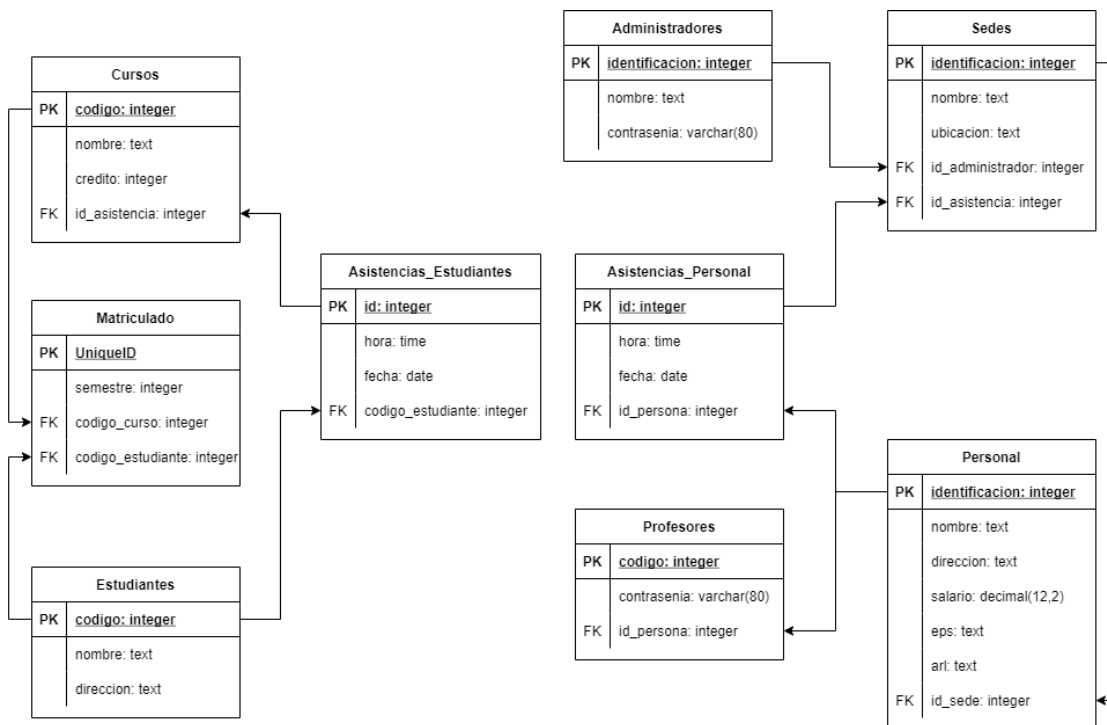


Modelo Relacional

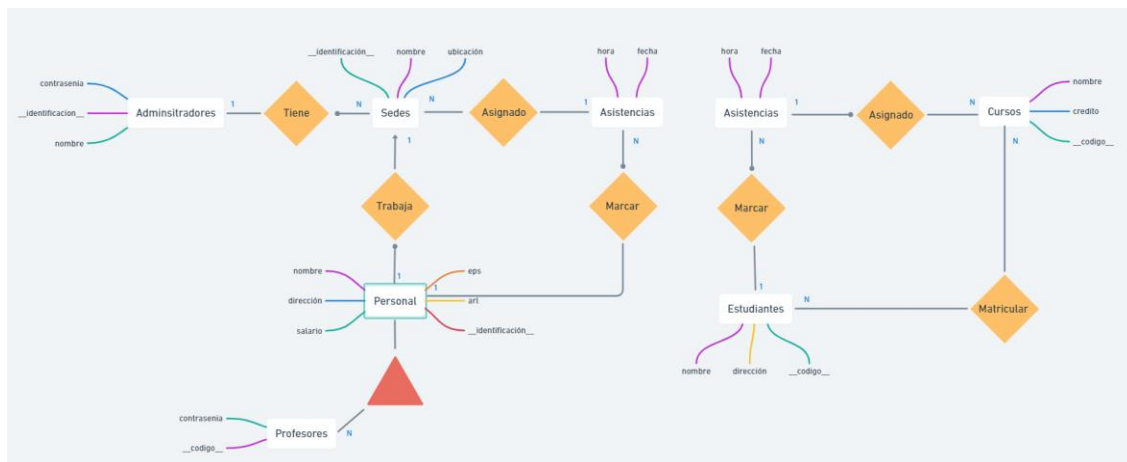
Se inicia el modelo relacional creando las tablas de las entidades con sus atributos.



Luego según su cardinalidad se comienza a agregar las claves foráneas que los conecta. Inicialmente en el desarrollo del diagrama se separó asistencia en 2 partes, asistencia_estudiantes y asistencia_personal. Esto para que los estudiantes marquen su asistencia a los cursos y el personal a la sede.

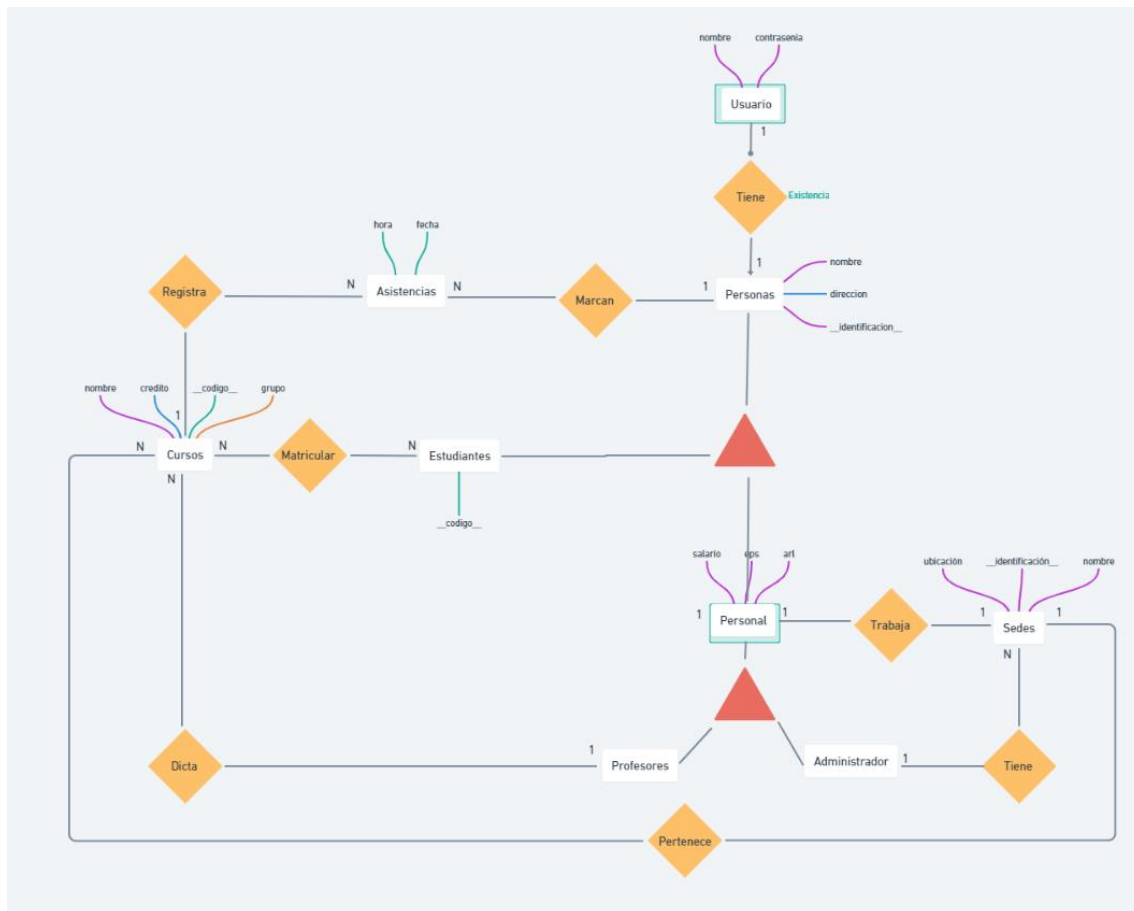


Segunda versión del diagrama entidad-relación



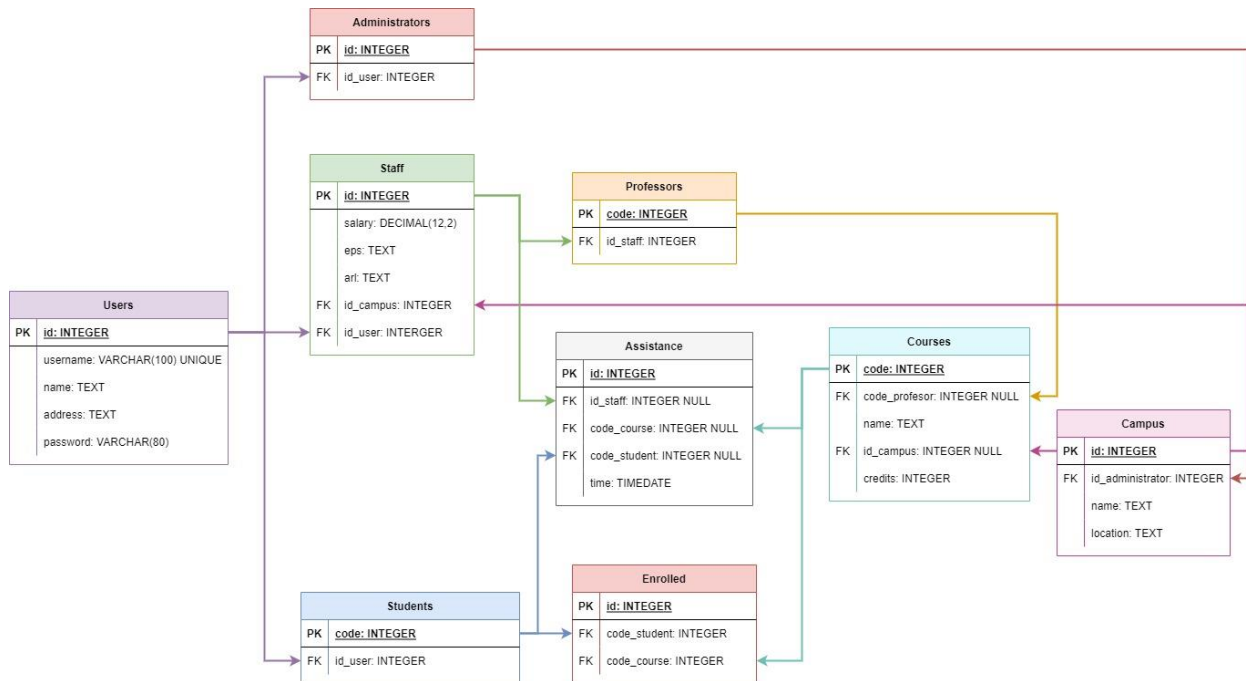
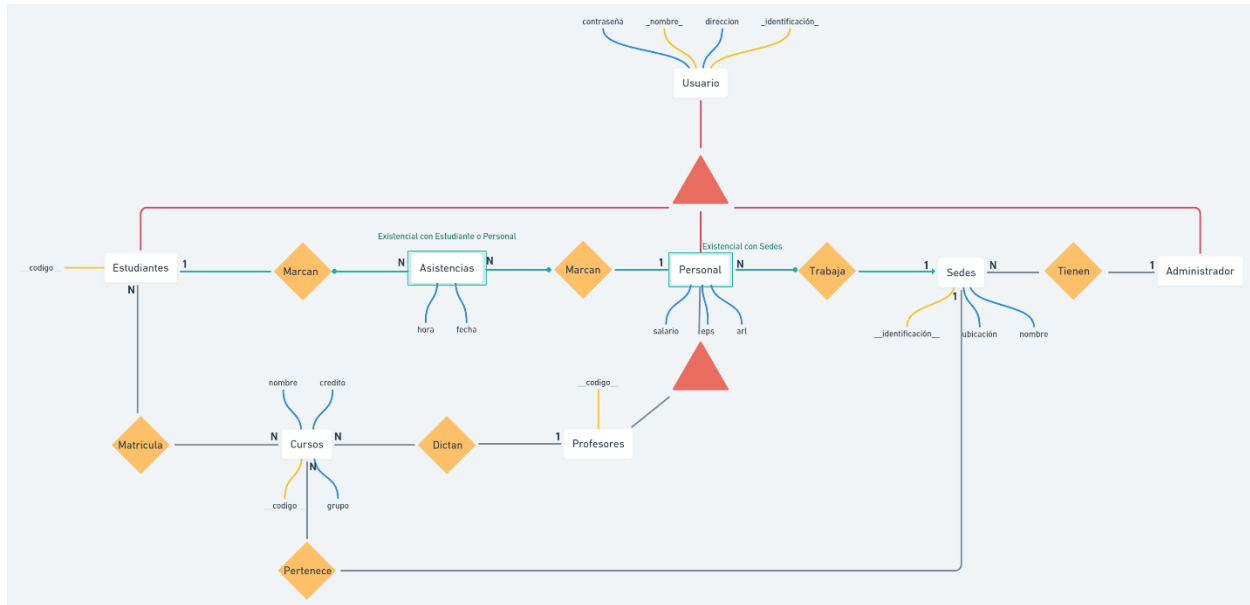
Pero como se puede ver en los diagramas se crean dos “islas” separando la parte de estudiantes y personal, también se puede ver repetición de datos. Por esta razón se volvió a crear el diagrama entidad-relación para mejorar el diseño de la base de datos sin tantos datos redundantes.

Tercera versión del diagrama entidad-relación



En este diagrama queda mejor dividido los datos agregando persona para generalizar estudiante y personal de este. Administrador se agrego a personal con la idea de que el también tiene que tener un saldo, pero esto no es pedido en el texto y a la hora de crear la sede genera problemas. Analizando el diagrama podemos agrupar usuarios y personas para mejorar las consultas de búsqueda y separar Administrador de personal.

Cuarta versión del diagrama entidad-relación



Ultima versión del diagrama de bases de datos.

Enlace a la versión final del diagrama -> <https://whimsical.com/modelo-entidad-relacion-9r9ZEmUjJvsvyTSkQ2dcYBH>

Codigo SQL

Archivo con las instrucciones SQL para crear la base de datos y tablas de la base de datos.

```
CREATE DATABASE attendance; //Solo se corre 1 vez
```

```
CREATE TABLE IF NOT EXISTS users(  
    id INTEGER PRIMARY KEY,  
    username VARCHAR(100) UNIQUE NOT NULL,  
    name TEXT NOT NULL,  
    address TEXT NOT NULL,  
    password VARCHAR(80) NOT NULL  
);
```

```
CREATE TABLE IF NOT EXISTS students(  
    code INTEGER PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,  
    id_user INTEGER NOT NULL,  
    FOREIGN KEY(id_user)  
        REFERENCES users(id)  
);
```

```
CREATE TABLE IF NOT EXISTS administrators (  
    id INTEGER PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,  
    id_user INTEGER NOT NULL UNIQUE,  
    FOREIGN KEY(id_user)  
        REFERENCES users(id)  
);
```

```
CREATE TABLE IF NOT EXISTS campus(  
    id INTEGER PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,  
    id_administrator INTEGER,  
    name TEXT NOT NULL,  
    location TEXT NOT NULL,  
    FOREIGN KEY(id_administrator)  
        REFERENCES administrators(id)  
);
```

```
CREATE TABLE IF NOT EXISTS staff (  
    id INTEGER PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,  
    salary DECIMAL(12,2),  
    eps TEXT DEFAULT 'N/A',  
    arl TEXT DEFAULT 'N/A',
```



```

    id_campus INTEGER NOT NULL,
    id_user INTEGER NOT NULL UNIQUE,
    FOREIGN KEY(id_campus)
        REFERENCES administrators(id),
    FOREIGN KEY(id_user)
        REFERENCES users(id)
);

CREATE TABLE IF NOT EXISTS professors (
    code INTEGER PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
    id_staff INTEGER NOT NULL,
    FOREIGN KEY(id_staff)
        REFERENCES staff(id)
);

CREATE TABLE IF NOT EXISTS courses (
    code INTEGER PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
    code_professor INTEGER,
    id_campus INTEGER,
    name TEXT NOT NULL,
    credits INTEGER,
    FOREIGN KEY(code_professor)
        REFERENCES professors(code),
    FOREIGN KEY(id_campus)
        REFERENCES campus(id)
);

CREATE TABLE IF NOT EXISTS enrolled (
    id INTEGER PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    code_student INTEGER NOT NULL,
    code_course INTEGER NOT NULL,
    FOREIGN KEY(code_student)
        REFERENCES students(code),
    FOREIGN KEY(code_course)
        REFERENCES courses(code)
);

CREATE TABLE IF NOT EXISTS assistance (
    id INTEGER PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    id_staff INTEGER,
    code_course INTEGER,
    code_student INTEGER,
    time TIMESTAMP NOT NULL,
    FOREIGN KEY(id_staff)
        REFERENCES staff(id),

```

```

FOREIGN KEY(code_course)
REFERENCES courses(code),
FOREIGN KEY(code_student)
REFERENCES students(code)
);

```

Dependencias funcionales y Normalización

Determinamos que datos nos permite la consulta eficiente de otros datos.

Users				
id	username	name	address	password
a1	b1	c1	d1	e1
a2	b2	c2	d1	e2
a3	b3	c2	d2	e3
a4	b4	c3	d3	e3
a5	b5	c4	d4	e3
a6	b6	c5	d5	e4
a7	b7	c6	d5	e5
a8	b8	c7	d6	e6
a9	b9	c8	d7	e7
a10	b10	c8	d8	e8
id , username -> name, address, password				
id -> name,address,password,username				

Administrators	
id	id_user
1	a2
2	a5
3	a7
id -> id_personal	

Campus			
id	id_administrator	name	location
1	f1	g1	h1
2	f2	g2	h2
3	f1	g3	h3
id -> id_administrator, name, location			

Staff					
id	salary	eps	arl	id_campus	id_user
1	i1	j1	k1	1	a1
2	i2	j1	k2	1	a3
3	i3	j2	k2	2	a4
id-> salary,eps,arl,id_campus,id_user					

Students	
code	id_user
k1	a6
k2	a8
k3	a9
k4	a10
code -> id_user	

Professors	
code	id_staff
l1	2
l2	1
code -> id_staff	

Enrolled		
id	code_student	code_course
1	k1	p3
2	k1	p2
3	k2	p2
4	k3	p2

id -> codigo_student, code_course

Courses				
code	code_professor	name	id_campus	credits
p1	l1	q1	m1	r1
p2	l1	q2	m1	r1
p3	l2	q1	m1	r2
p4	l2	q3	m2	r3

code -> code_professor, name, id_campus,credits

Assistance				
id	id_staff	code_course	id_student	time
1	2	null	null	s1
2	null	p2	k1	s1
3	null	p2	k2	s2
4	3	null	null	s3

5	1	l1	null	s4
---	---	----	------	----

id -> id_staff, code_course, id_student, time

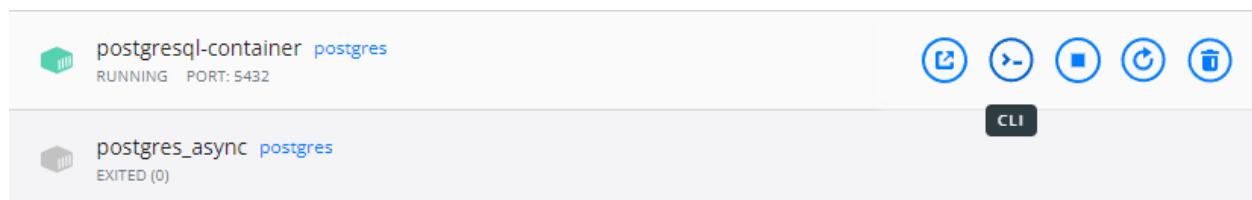
Como se ve en las tablas anteriores con valores imaginarios (a1,b1,...) todos los atributos son valores atómicos. En la relación Estudiante – Curso que era muchos a muchos se creo una tabla llamada matriculados (enrolled) donde se tienen los valores de forma atómica. Todas las claves son las claves primarias en el diagrama relacional se encuentran como PK y los atributos no primos dependen de esta. Podemos entonces decir que nuestro diseño se encuentra normalizado 3FN porque cada columna esta identificada de manera única por la llave primaria.

Despliegue

Creamos la base de datos en un contenedor de PostgreSQL utilizando Docker. Para comenzar se crea un contenedor nuevo desde la consola que acceda a un puerto en específico, en este caso 5432.

```
docker run --name postgresql-container -d -p 5432:5432 -e POSTGRES_PASSWORD=root postgres
```

Se inicia y se entra a la consola por Docker.



Para entrar a la base de datos de postgres utilizamos: # psql -U postgres. En este punto se puede ver las bases de datos creadas.

```
postgres=# \l
```

List of databases					
Name	Owner	Encoding	Collate	Ctype	Access privileges
postgres	postgres	UTF8	en_US.utf8	en_US.utf8	
template0	postgres	UTF8	en_US.utf8	en_US.utf8	=c/postgres +
template1	postgres	UTF8	en_US.utf8	en_US.utf8	=c/postgres +
					postgres=CTc/postgres

```
(3 rows)
```

Se tiene las que ya vienen por defecto, con el comando **CREATE DATABASE**

attendance; se crea la base de datos. Para entrar en la bd creada utilizamos \c attendance y se comienza a copiar las instrucciones sql para crear las tablas.

```
postgres=# \c attendance
You are now connected to database "attendance" as user "postgres".
attendance=# CREATE TABLE IF NOT EXISTS users(
attendance(# id INTEGER PRIMARY KEY,
attendance(# username VARCHAR(100) UNIQUE NOT NULL,
attendance(#     name TEXT NOT NULL,
attendance(#     address TEXT NOT NULL,
attendance(#     password VARCHAR(80) NOT NULL
attendance(# );
CREATE TABLE
attendance=#
attendance=# CREATE TABLE IF NOT EXISTS students(
attendance(#     code INTEGER PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
attendance(#     id_user INTEGER NOT NULL,
attendance(#     FOREIGN KEY(id_user)
attendance(#         REFERENCES users(id)
attendance(# );
CREATE TABLE
attendance=#
attendance=# CREATE TABLE IF NOT EXISTS administrators (
attendance(#     id INTEGER PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
attendance(#     id_user INTEGER NOT NULL UNIQUE,
attendance(#     FOREIGN KEY(id_user)
attendance(#         REFERENCES users(id)
attendance(# );
CREATE TABLE
```

Se utiliza \dt para ver las tablas que se crearon

```
attendance=# \dt
          List of relations
 Schema |      Name      | Type  | Owner
-----+-----+-----+-----
 public | administrators | table | postgres
 public | assistance     | table | postgres
 public | campus         | table | postgres
 public | courses        | table | postgres
 public | enrolled       | table | postgres
 public | professors     | table | postgres
 public | staff          | table | postgres
 public | students       | table | postgres
 public | users          | table | postgres
(9 rows)
```

Ya con las tablas creadas de forma correcta, se puede comenzar a desarrollar el backend.

Backend – FrontEnd

Para iniciar con la parte del backend y frontend del proyecto, se utilizó una guía de FreeCodeCamp para crear paso a paso las carpetas necesarias y vincularlas.

How to create a React frontend and a Node/Express backend and connect them



by João Henrique

<https://www.freecodecamp.org/news/create-a-react-frontend-a-node-express-backend-and-connect-them-together-c5798926047c/>

Se utilizo `npx create-react-app client` para crear las carpetas base con react y `npx express-generator api` para crear las carpetas base con express. Estas carpetas contienen la estructura base para comenzar el desarrollo de la aplicación. Una dificultad presentada, es que la versión actual de react es la 6 y la mayoría de los tutoriales tenían versiones anteriores en el cual se usaban clases y no funciones, esta dificultad se pudo superar gracias a tutoriales en internet y foros en stackoverflow.

Comenzamos en la carpeta client conectando la base de datos, en un archivo llamado `bd.js` se tiene el siguiente código para realizar la conexión.

```
const {Pool} = require('pg')
const client = new Pool({
  host: "0.0.0.0",
  user: "postgres",
  port: "5432",
  password: "root",
  database: "attendance"
})

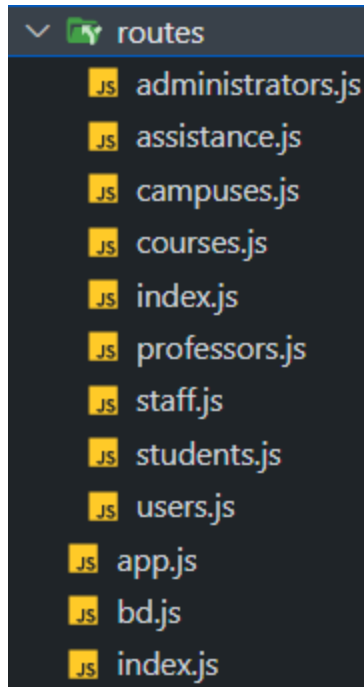
client.connect();

module.exports = client;
```

En los archivos que se necesita hacer llamadas a la base de datos importamos client de la siguiente forma:

```
var client = require("../bd");
```

Con la conexión ya establecida de este modo se empieza crear las llamadas api de cada tabla en nuestra bd, para esto se creo un archivo individual para cada una con su respectivo nombre en el cual contiene los métodos necesarios (get,push,put,delete).



Ejemplo de cursos (courses):

```
var express = require("express");
var router = express.Router();
var client = require("../bd");

router.get("/", async (req, res) => {
  try {
    const response = await client.query(`SELECT * FROM courses`);
    return res.status(200).json(response.rows);
  } catch (error) {
    res.status(500).json(error);
  }
});

router.get("/enrolled/:code", async (req, res) => {
  const {code} = req.params
  try {
    const response = await client.query(`SELECT * FROM enrolled
WHERE code_student = $1 or code_course = $1`, [code]);
    return res.status(200).json(response);
  } catch (error) {
    res.status(500).json({error: error});
  }
});
```



```

router.post("/", async (req, res) => {
  let { code, name, credits, id_campus, code_professor } = req.body;
  try {
    const create = await client.query(
      `INSERT INTO courses(code, name, credits, code_professor,
id_campus) VALUES ($1, $2, $3, $4, $5)`,
      [code, name, credits, code_professor, id_campus]
    );
    res.status(201).json(create);
  } catch (error) {
    if (error.code === "23505") {
      return res.status(409).json("Already exists");
    } else {
      return res.status(500).json(error);
    }
  }
});

```

```

router.delete("/:code", (req, res) => {
  const { code } = req.params;
  try {
    const deleted = client.query("DELETE FROM courses WHERE code=$1",
[code]);
    return res.status(200).json({ message: "Deleted" });
  } catch (error) {
    return res.status(500).send(error);
  }
});

```

```

router.put("/:code", (req, res) => {
  const { code } = req.params;
  let { name, credits, id_campus, code_professor } = req.body;

  try {
    const update = client.query(
      `UPDATE courses
      SET name = $1,
      credits = $2,
      id_campus = $3,
      code_professor = $4
      WHERE code = $5
      `,
      [name, credits, id_campus, code_professor, code]
    );
  }
});

```

```

        return res.status(200).json(update);
    } catch (error) {
        return res.status(500).send(error);
    }
});

```

module.exports = router;

Ver todos los archivos: <https://github.com/Ingrid-E/Attendance/tree/main/api/routes>

En app.js creamos las rutas para estos archivos:

```

var indexRouter = require('./routes/index');
var coursesRouter = require('./routes/courses');
var studentsRouter = require('./routes/students');
var professorsRouter = require('./routes/professors');
var campusesRouter = require('./routes/campuses');
var staffRouter = require('./routes/staff');
var assistanceRouter = require('./routes/assistance');
var adminRouter = require('./routes/administrators');
var usersRouter = require('./routes/users');

app.use('/', indexRouter);
app.use('/courses', coursesRouter);
app.use('/students', studentsRouter);
app.use('/users', usersRouter);
app.use('/staff', staffRouter);
app.use('/admin', adminRouter);
app.use('/assistance', assistanceRouter);
app.use('/campuses', campusesRouter);
app.use('/professors', professorsRouter);

```

Para obtener el valor de estas llamadas en nuestro frontend creamos un archivo

llamado client.js en la parte de react. Para no repetir tanto código al llamar información.

```

const baseUrl = "http://localhost:9000"

export async function get(endpoint) {
    const response = await fetch(baseUrl + endpoint, {
        method: "GET",
        headers: {
            Accept: "application/json",
        },
    },

```

```

    });
    if (!response.ok) {
        // make the promise be rejected if we didn't get a 2xx
response
        const err = new Error("Not 2xx response");
        err.response = response;
        throw err;
    } else {
        return await response.json();
    }
}

export async function post(endpoint, data) {
    const response = await fetch(baseUrl + endpoint, {
        method: "POST",
        headers: {
            'Accept': 'application/json',
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(data)
    });
    if (!response.ok) {
        // make the promise be rejected if we didn't get a 2xx response
        const err = new Error("Not 2xx response");
        err.response = response;
        throw err;
    } else {
        return await response.json();
    }
}

export async function del(endpoint) {
    const response = await fetch(baseUrl + endpoint, {
        method: "DELETE",
        headers: {
            'Accept': 'application/json',
            'Content-Type': 'application/json'
        }
    });
    if (!response.ok) {
        // make the promise be rejected if we didn't get a 2xx response
        const err = new Error("Not 2xx response");
        err.response = response;
    }
}

```

```

        throw err;
    } else {
        return await response.json();
    }
}

export async function put(endpoint,data) {
    console.log("Calling Put")
    const response = await fetch(baseUrl + endpoint, {
        method: "PUT",
        headers: {
            'Accept': 'application/json',
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(data)
    });
    if (!response.ok) {
        // make the promise be rejected if we didn't get a 2xx response
        const err = new Error("Not 2xx response");
        err.response = response;
        throw err;
    } else {
        return await response.json();
    }
}

```

Esto nos facilita el trabajo, en nuestro código podemos realizar llamadas de esta forma.

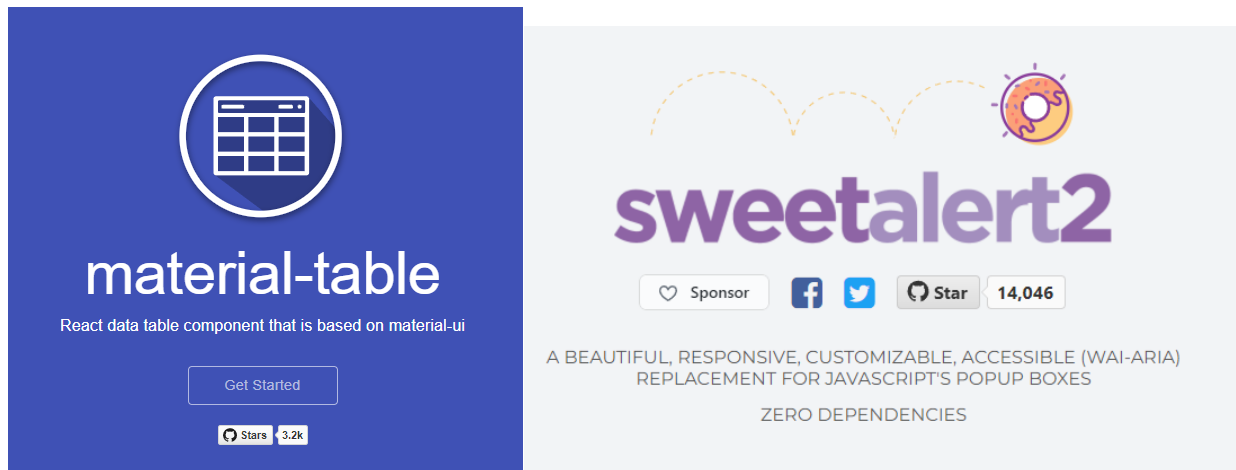
```

async function getCampuses() {
    try {
        const response = await get("/campuses");
        response.forEach((element) => {
            campuses.current[element.id] = element.name + " " +
element.location
        });
    } catch (error) {
        console.log(error);
    }
}

```

Para el diseño del frontend se utilizó librerías de Material UI como Material Tables,

Buttons y Icons. Para las alertas se utilizó [SweetAlerts2](#).



La estructura básica de los componentes y paginas creadas es la siguiente. Se importan todas las librerías que se van a utilizar y los estilos css. Luego se crea la función con el mismo nombre del archivo, la cual contiene un return () con la estructura html.

Dentro de la función tenemos algún setState para guardar datos importantes y que en su cambio se renderice de nuevo para visualizar los cambios. En algunos tenemos un useEffect() para los métodos que queremos que se llamen solo 1 vez y no con cada renderización. De resto tenemos los métodos necesarios para realizar operaciones y llamar información del backend.

Ejemplos:

```
let [state, setState] = useState({
  username: "ini",
  password: "ini",
  type: "",
  error: ""
});

async function getCourses(professor_info) {
  try {
```

```

        const response =await
get(`/courses/assigned/${professor_info.code}`)
    setProfessor({...professor_info, courses: response})
  } catch (error) {
    console.error(error);
  }
}

```

```

const addCourse = async (data) => {
  try {
    if(data.code_professor === 'null'){
      delete data.code_professor
    }
    if(data.id_campus === 'null'){
      delete data.id_campus
    }

    await post(`/courses`, data);
    getCourses();
    swal({
      text:"Curso Agregado",
      icon: "success"
    })
  } catch (error) {
    swal({
      text:"Error Agregando Curso",
      icon: "error"
    })
    if (error.response.status === 409) {
      console.error("El curso ya existe");
    }
  }
};

```

```

return (
  <div className="LogIn">
    <img className="logo" src={Logo}></img>
    <div className="leftSide">
      <form onSubmit={handleSubmit} onChange={handleChange}>
        <h1 className="title">Login</h1>
        <TextField
          type="text"
          name="username"

```

```

        placeholder="Username"
        size="small"
        variant="outlined"
        error = {state.username.trim() === ''}
        helperText = {state.username.trim() === '' ? 'Ingresar un
valor': ''}
        InputProps={{
            startAdornment: (
                <InputAdornment position="start">
                    <AccountCircle
                        style={{
                            fill: "#2C4252"
                        }}
                    />
                </InputAdornment>
            ),
        }}
    />

    <TextField
        type="password"
        name="password"
        placeholder="Password"
        size="small"
        variant="outlined"
        error = {state.password.trim() === ''}
        helperText = {state.password.trim() === '' ? 'Ingresar un
valor': ''}
        InputProps={{
            startAdornment: (
                <InputAdornment position="start">
                    <VpnKey
                        style={{
                            fill: "#2C4252"
                        }}
                    />
                </InputAdornment>
            ),
        }}
    />
    <p className="error-text">{state.error === '' ?
'' : state.error}</p>
    <Button type="submit" className="button"
variant="contained">Log In</Button>
</form>
</div>

```

```
<div className="rightSide"></div>
</div>
```

Ver codigo:

- [Paginas](#)
- [Complementos](#)

Diseño:



Login

LOG IN





Login


Ingresar un valor


LOG IN





 Ingrid



 Cursos

 Personal

 Estudiantes

 Profesores











Cursos

Q

Search

X

+ Agregar

Code	Name	Credits	Professor	Campus	Actions
750030	Base de Datos	4	Luz Marina Rodríguez	Univalle Cali	 
750126	Desarrollo de Videojuegos	3	Luis Alberto Rodríguez	Univalle Palmira	 
760081	Teoría General de Sistemas	3	María del Carmen Gómez	Floar Palmira	 
710193	Arquitectura de Computadores 2	3	Luz Marina Rodríguez	Univalle Cali	 
111048	Algebra Lineal	4	Luis Alberto Rodríguez	Floar Palmira	 

5 rows

|< < 1-5 of 7 > >|

Webgrafía

- Atzeni, P., Ceri, S., Paraboschi, S., & Torlone, R. *Database Systems concepts, languages & architectures*. Database Systems concepts, languages & architectures.
- Codenemy. (2022). *Learn Material Table with React JS in 1 Hour | Material Table Tutorial for Beginners* [Video]. Retrieved 12 January 2022, from <https://www.youtube.com/watch?v=4TOONPrIiKY&t=3278s>.
- Codenemy. (2021). *5. CRUD Operations in Material Table || Material UI* [Video]. Retrieved 12 January 2022, from <https://youtu.be/S7-99HqpWvo>.
- Difference between On Delete Cascade & On Update Cascade in mysql*. Database Administrators Stack Exchange. (2017). Retrieved 12 January 2022, from <https://dba.stackexchange.com/questions/74627/difference-between-on-delete-cascade-on-update-cascade-in-mysql>.
- Difference between text and varchar (character varying)*. Stack Overflow. (2018). Retrieved 12 January 2022, from <https://stackoverflow.com/questions/4848964/difference-between-text-and-varchar-character-varying>.
- Henrique, J. (2018). *How to create a React frontend and a Node/Express backend and connect them*. freeCodeCamp.org. Retrieved 12 January 2022, from <https://www.freecodecamp.org/news/create-a-react-frontend-a-node-express-backend-and-connect-them-together-c5798926047c/>.
- J. Dávila, M. (2020). *Express.js UPDATE de registros a través de un formulario-post (Sequelize y Mysql)* [Video]. Retrieved 12 January 2022, from <https://www.youtube.com/watch?v=vtQT3Qbi384&t=288s>.
- Peña Torres, J. (2022). *Base de datos 1*. Campusvirtual.univalle.edu.co. Retrieved 12 January 2022, from https://campusvirtual.univalle.edu.co/moodle/pluginfile.php/3167075/mod_resource/content/0/750030M-2021-II-ClaseNo9c.pdf.
- SQL Inner-join with 3 tables?*. Stack Overflow. (2019). Retrieved 12 January 2022, from <https://stackoverflow.com/questions/10195451/sql-inner-join-with-3-tables>.
- Updating primary keys in POSTGRESQL*. Stack Overflow. (2014). Retrieved 12 January 2022, from <https://stackoverflow.com/questions/25973576/updating-primary-keys-in-postgresql>.
- Why not use Double or Float to represent currency?*. Stack Overflow. (2019). Retrieved 12 January 2022, from <https://stackoverflow.com/questions/3730019/why-not-use-double-or-float-to-represent-currency>.