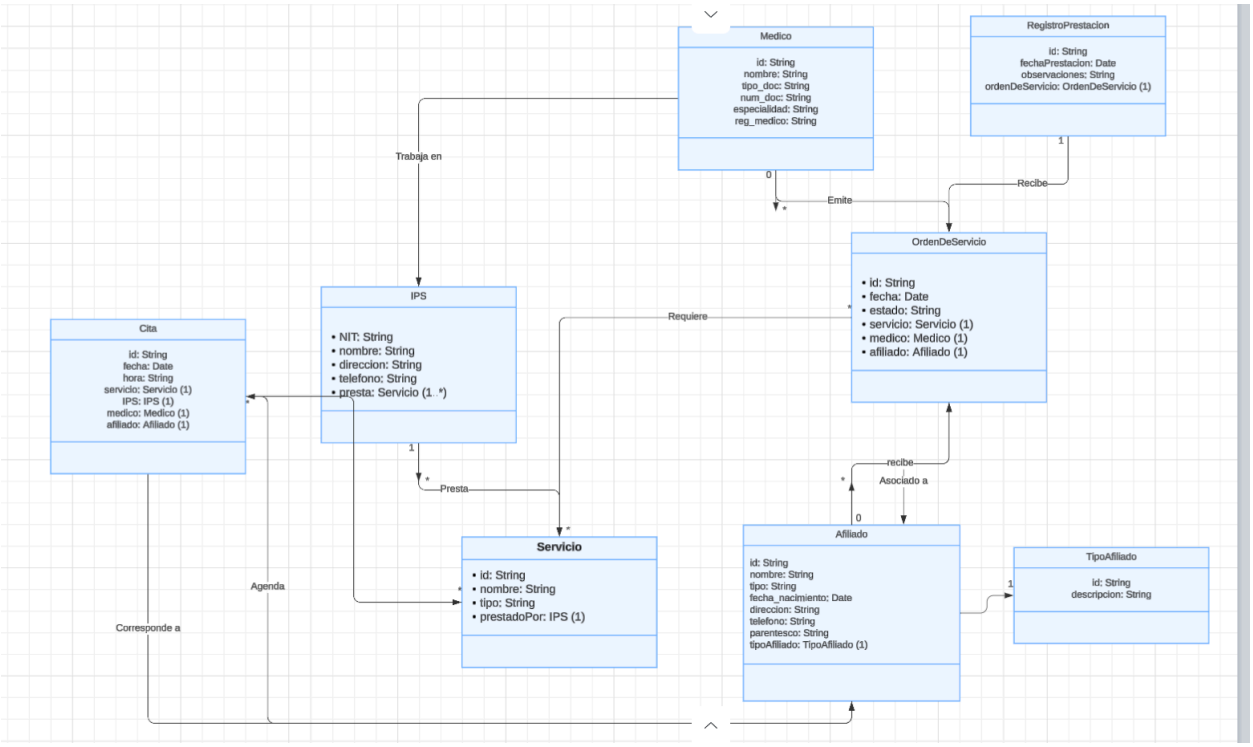
Santiago Guevara-202325999

Juan Felipe Lopez-202320114

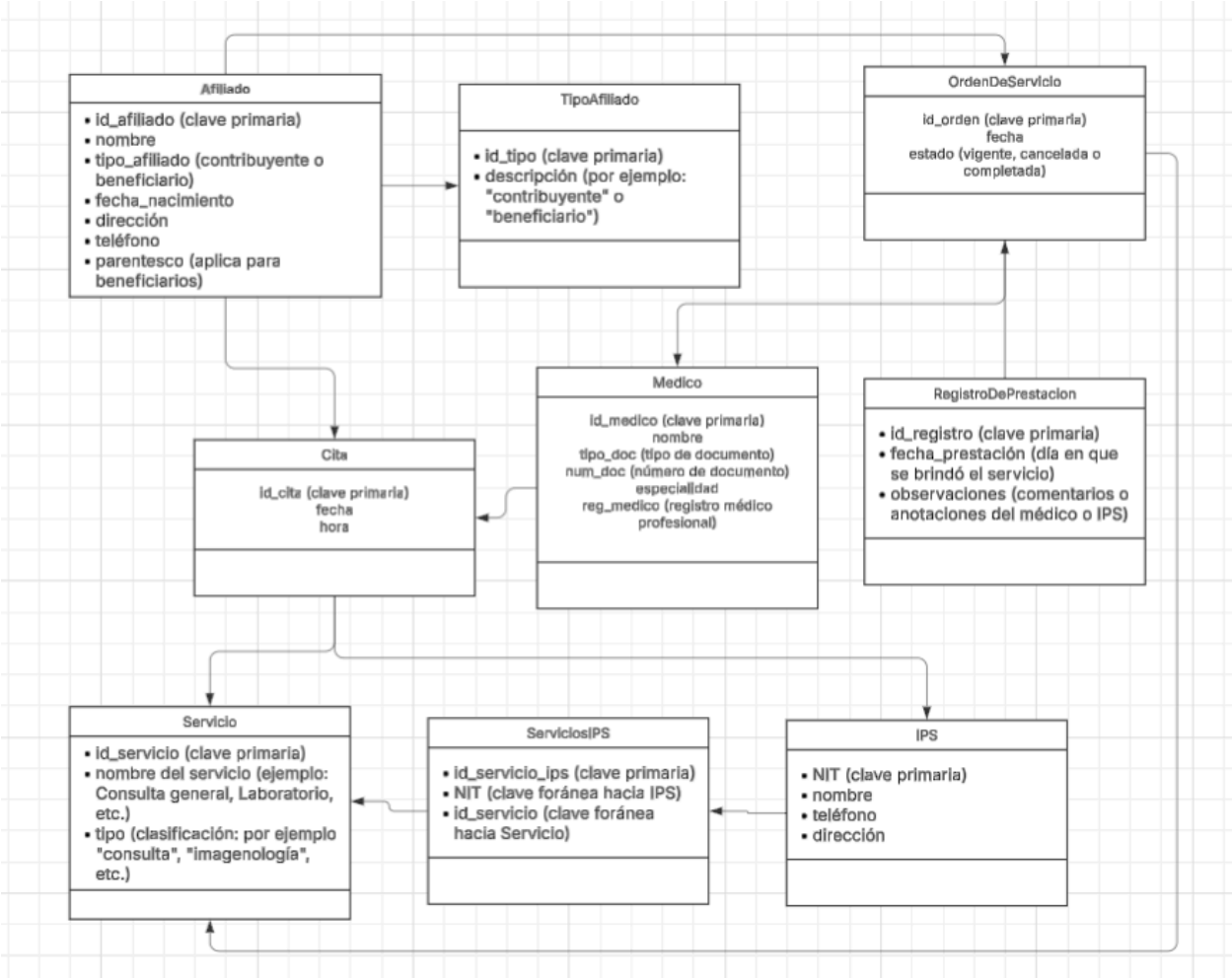
Tomas Uribe-202321557

Entrega del proyecto

1)UML



Modelo E/R



-Las clases que cambiaron son la eliminación de eps y la creacion de una nueva clase llamada RegistrodePrestacion

-Las relaciones del modelo estan en la Forma Normal de Boyce-Codd (BCNF) por las siguientes razones:

Para que un esquema esté en **BCNF**, debe cumplir con las siguientes condiciones:

Debe estar en 3FN, es decir:

- Estar en 1FN: Todos los atributos deben contener valores atómicos, sin conjuntos o listas.
- Estar en 2FN: No debe haber dependencias parciales; cada atributo no clave debe depender completamente de la clave primaria.

- Estar en 3FN: No debe haber dependencias transitivas.

Afiliado

- Clave primaria: `id_afiliado`.
- Atributos: `nombre`, `tipo_afiliado`, `fecha_nacimiento`, `dirección`, `teléfono`, `parentesco (para beneficiarios)`.
- Dependencias funcionales:

  `id_afiliado → {nombre, tipo_afiliado, fecha_nacimiento, dirección, teléfono, parentesco}`.

No hay dependencias parciales ni transitivas. Todos los atributos dependen completamente de la clave primaria.
Conclusión: Esta relación está en BCNF.

TipoAfiliado

- Clave primaria: `id_tipo`.
- Atributos: `descripción`.
- Dependencias funcionales:

  `id_tipo → descripción`.

No hay dependencias parciales ni transitivas.
Conclusión: Esta relación está en BCNF.

Médico

- Clave primaria: `id_medico`.
- Atributos: `nombre`, `tipo_doc`, `num_doc`, `especialidad`, `reg_medico`.
- Dependencias funcionales:

  `id_medico → {nombre, tipo_doc, num_doc, especialidad, reg_medico}`.

Todos los atributos dependen completamente de la clave primaria y no hay dependencias transitivas.
Conclusión: Esta relación está en BCNF.

OrdenDeServicio

- Clave primaria: `id_orden`.
- Atributos: `fecha`, `estado`.
- Dependencias funcionales:

  `id_orden → {fecha, estado}`.

No hay dependencias parciales ni transitivas.
Conclusión: Esta relación está en BCNF.

RegistroDePrestacion

- Clave primaria: `id_registro`.
- Atributos: `fecha_prestacion`, `observaciones`.
- Dependencias funcionales:

  `id_registro → {fecha_prestacion, observaciones}`.

No hay dependencias parciales ni transitivas.
Conclusión: Esta relación está en BCNF.

Cita

- Clave primaria: `id_cita`.
- Atributos: `fecha`, `hora`.
- Dependencias funcionales:

  `id_cita → {fecha, hora}`.

No hay dependencias parciales ni transitivas.
Conclusión: Esta relación está en BCNF.

Servicio

- Clave primaria: `id_servicio`.
- Atributos: `nombre del servicio`, `tipo`.
- Dependencias funcionales:
  - `id_servicio → {nombre del servicio, tipo}`.

No hay dependencias parciales ni transitivas.
Conclusión: Esta relación está en BCNF.

ServicioIPS

- Clave primaria: `id_servicio_ips`.
- Claves foráneas: `NIT`, `id_servicio`.
- Dependencias funcionales:

  `id_servicio_ips → {NIT, id_servicio}`.

No hay dependencia parcial ni transitiva.
Conclusión: Esta relación está en BCNF.

IPS

- Clave primaria: `NIT`.
- Atributos: `nombre`, `teléfono`, `dirección`.
- Dependencias funcionales:

  `NIT → {nombre, teléfono, dirección}`.

No hay dependencias parciales ni transitivas.
Conclusión: Esta relación está en BCNF.

2.

El script.sql se encuentra en el documento “Creacion tablas punto 2.sql” en la carpeta “docs” en el repositorio.

```sql
-- TABLA: EPS
CREATE TABLE EPS (
    id_eps      NUMBER          NOT NULL PRIMARY KEY,
    nombre      VARCHAR2(200)   NOT NULL
);

-- TABLA: IPS
CREATE TABLE IPS (
    nit         VARCHAR2(20)    NOT NULL PRIMARY KEY,
    nombre      VARCHAR2(200)   NOT NULL,
    telefono    VARCHAR2(50)    NOT NULL,
    direccion   VARCHAR2(200)   NOT NULL,
    id_eps      NUMBER          NOT NULL,
    FOREIGN KEY (id_eps) REFERENCES EPS (id_eps)
);

-- TABLA: SERVICIO_SALUD (Ahora con FK de IPS)
CREATE TABLE Servicio_Salud (
    id_servicio  NUMBER         NOT NULL PRIMARY KEY,
    nombre       VARCHAR2(200)  NOT NULL,
    tipo         VARCHAR2(100)  NOT NULL,
    nit_ips      VARCHAR2(20)   NOT NULL,  -- FK a IPS
    FOREIGN KEY (nit_ips) REFERENCES IPS (nit)
);

-- TABLA: MÉDICO
CREATE TABLE Medico (
    id_medico       NUMBER          NOT NULL PRIMARY KEY,
    nombre          VARCHAR2(200)   NOT NULL,
    tipo_doc        VARCHAR2(10)    NOT NULL,
    num_doc         VARCHAR2(50)    NOT NULL UNIQUE,
    especialidad    VARCHAR2(100),
    reg_medico      VARCHAR2(50)    NOT NULL UNIQUE
);
```

```sql
-- TABLA: AFILIADO
CREATE TABLE Afiliado (
    id_afiliado     NUMBER          NOT NULL PRIMARY KEY,
    nombre          VARCHAR2(200)   NOT NULL,
    tipo_afiliado   VARCHAR2(15)    NOT NULL CHECK (tipo_afiliado IN ('Contribuyente','Beneficiario')),
    fecha_nacimiento DATE           NOT NULL,
    direccion       VARCHAR2(200),
    telefono        VARCHAR2(50),
    parentesco      VARCHAR2(100) NULL,  -- Para beneficiarios
    id_contribuyente NUMBER NULL,
    FOREIGN KEY (id_contribuyente) REFERENCES Afiliado (id_afiliado)
);

-- TABLA: ORDEN SERVICIO
CREATE TABLE Orden_Servicio (
    id_orden     NUMBER         NOT NULL PRIMARY KEY,
    fecha        DATE           NOT NULL,
    estado       VARCHAR2(15)   NOT NULL CHECK (estado IN ('VIGENTE','CANCELADA','COMPLETADA')),
    id_afiliado  NUMBER         NOT NULL,
    FOREIGN KEY (id_afiliado) REFERENCES Afiliado (id_afiliado)
);

-- TABLA: CITA
CREATE TABLE Cita (
    id_cita      NUMBER         NOT NULL PRIMARY KEY,
    fecha_hora   TIMESTAMP      NOT NULL,
    id_orden     NUMBER         NULL, -- Puede estar vacía
    id_afiliado  NUMBER         NOT NULL,
    id_medico    NUMBER         NOT NULL,
    nit_ips      VARCHAR2(20)   NOT NULL,
    id_servicio  NUMBER         NOT NULL,
    FOREIGN KEY (id_orden) REFERENCES Orden_Servicio (id_orden),
    FOREIGN KEY (id_afiliado) REFERENCES Afiliado (id_afiliado),
    FOREIGN KEY (id_medico) REFERENCES Medico (id_medico),
    FOREIGN KEY (nit_ips) REFERENCES IPS (nit),
    FOREIGN KEY (id_servicio) REFERENCES Servicio_Salud (id_servicio)
```

3.

Documentación de las clases de Java Spring y de la arquitectura de la aplicación

Este proyecto implementa una arquitectura basada en Spring Boot y Spring Data JPA, con una separación en capas para manejar la funcionalidad de la base de datos (repositorios), la lógica de negocio (opcional, no incluida si no se requiere) y la capa de presentación/exposición (controladores REST). A continuación, se describen cada uno de los módulos y las clases principales, junto con la estructura de la aplicación y las relaciones entre ellas.

ARQUITECTURA GENERAL

La aplicación sigue una **arquitectura en capas**:

1. **Modelo (Entidades)**
    - Clases con anotaciones JPA (por ejemplo, @Entity, @Id, @Table).
    - Representan las tablas de la base de datos (por ejemplo, Afiliado, IPS, Servicio, etc.).
    - Pueden tener llaves primarias simples o compuestas (con @Embeddable y @EmbeddedId).
2. **Repositorios**
    - Interfaces que extienden JpaRepository<Entity, PK>.
    - Contienen métodos CRUD y consultas personalizadas usando @Query.
    - Se comunican directamente con la base de datos.
3. **Controladores (REST)**
    - Clases con @RestController.
    - Exponen endpoints HTTP (por ejemplo, GET /ips, POST /ips/new/save) que reciben peticiones de Postman o un frontend, y responden en formato JSON o texto.
    - Invocan los métodos de los repositorios para las operaciones de base de datos

DETALLE DE CADA CAPA Y CLASES

Entidades (modelo)

Representan las **tablas** de la base de datos. Cada clase se anota con @Entity y @Table(name="..."). Sus campos se anotan con @Id (clave primaria), @Column, etc. Cuando hay llaves foráneas, se usan @ManyToOne y @JoinColumn. Para relaciones M:N con PK compuesta, se usa @Embeddable.

Ejemplos:

1. **IPS**
    - **Propósito**: Modela la tabla IPS, con atributos nit, nombre, direccion, y telefono.
    - **Anotaciones**: @Entity, @Table(name="IPS"), @Id en nit.
    - **Responsabilidad**: Encapsula la información de una IPS.
2. **Servicio**

    - **Propósito**: Tabla SERVICIO, con idServicio, nombreServicio, tipoServicio.
    - **Anotaciones**: @Entity, @Table(name="SERVICIO"), @Id en idServicio.
    - **Responsabilidad**: Representa un tipo de servicio de salud (consulta, examen, etc.).
3. **IPS_Servicio**

    - **Propósito**: Tabla intermedia (M:N) entre IPS y Servicio.
    - **Anotaciones**: @Entity, @Table(name="IPS_SERVICIO"), @EmbeddedId.
    - **Responsabilidad**: Modela la asociación “Una IPS puede ofrecer múltiples servicios.”
4. **Afiliado**

    - **Propósito**: Tabla AFILIADO, con atributos idAfiliado, nombre, fechaNacimiento, tipoAfiliado.
    - **Anotaciones**: @Entity, @Table(name="AFILIADO"), @Id en idAfiliado.
    - **Responsabilidad**: Representa a un usuario que se beneficia de la EPS.
5. **Medico**, **OrdenServicio**, **DetalleOrden**, **Cita**, **Prestacion**

    - Cada uno mapea una tabla.
    - Usan anotaciones de JPA para llaves primarias, foráneas, etc.

Repositorios

Las clases en repositorio/ son interfaces que extienden JpaRepository. Contienen métodos como:

- darX() → un @Query para listar.
- insertarX(), actualizarX(), eliminarX(), usando @Modifying, @Transactional y @Query con sentencias nativas.
- Métodos autogenerados como findById(), save(), etc., si no se usan queries nativas.

Controladores

Las clases con @RestController y endpoints tipo @GetMapping, @PostMapping.
Responsabilidad: exponen las operaciones CRUD (listar, crear, editar, eliminar) de cada entidad vía HTTP.

- Inyectan (@Autowired) el repositorio respectivo.
- Procesan la petición y retornan JSON o mensajes de texto.

Flujo de ejecución

1. El **usuario** o Postman envía una petición a un endpoint, por ejemplo POST /ips/new/save.
2. El **controller** (p. ej. IPSController) recibe la petición, la parsea (JSON a objeto IPS), y llama a un método del repositorio (ipsRepo.insertarIPS(...)).
3. El **repositorio** ejecuta la sentencia SQL nativa o usa el método save(...).
4. Oracle hace la inserción, y se confirma la transacción.
5. Se **devuelve** una respuesta ("IPS creada exitosamente", HTTP 201).

4.

El script.sql se encuentra en el documento “Consultas RFC1 al RFC4.sql” en la carpeta “docs” en el repositorio.

Consulta RFC1

```sql
SELECT
    Servicio_Salud.nombre AS nombre_servicio,
    Cita.fecha_hora AS fecha_hora_disponible,
    IPS.nombre AS nombre_ips,
    Medico.nombre AS nombre_medico
FROM Cita
JOIN Servicio_Salud ON Cita.id_servicio = Servicio_Salud.id_servicio
JOIN IPS ON Cita.nit_ips = IPS.nit
JOIN Medico ON Cita.id_medico = Medico.id_medico
WHERE
    Servicio_Salud.id_servicio = &id_servicio
    AND Cita.fecha_hora BETWEEN SYSDATE AND (SYSDATE + INTERVAL '28' DAY)  -- 4 semanas = 28 días
ORDER BY Cita.fecha_hora;
```

Tablas y atributos utilizados

1. Servicio_Salud
    - `id_servicio`: Identificador único del servicio de salud (Clave primaria).
    - `nombre`: Nombre del servicio de salud.
2. Cita
    - `id_servicio`: Identificador del servicio de salud prestado en la cita (Clave foránea).
    - `fecha_hora`: Fecha y hora de la cita, indicando la disponibilidad del servicio.
    - `id_medico`: Identificador del médico asignado a la cita (Clave foránea).
    - `nit_ips`: Identificador de la IPS donde se realiza la cita (Clave foránea).
3. IPS
    - `nit`: Identificador único de la IPS (Clave primaria).
    - `nombre`: Nombre de la IPS que ofrece el servicio.
4. Medico
    - `id_medico`: Identificador único del médico (Clave primaria).
    - `nombre`: Nombre del médico que prestará el servicio.

Uso del INNER JOIN: Se utiliza `INNER JOIN` porque solo se desean obtener registros donde existan relaciones válidas entre las tablas. Esto garantiza que solo se muestren citas programadas para servicios de salud existentes, se incluyan únicamente citas asignadas a médicos registrados, se presenten únicamente citas en IPS registradas y no aparezcan datos huérfanos sin relación.

Consulta RFC2

```
SELECT
    Servicio_Salud.nombre AS nombre_servicio,
    COUNT(Cita.id_cita) AS cantidad_solicitudes
FROM Cita
JOIN Servicio_Salud ON Cita.id_servicio = Servicio_Salud.id_servicio
WHERE
    Cita.fecha_hora BETWEEN TO_DATE('&fecha_inicio', 'YYYY-MM-DD')
                        AND TO_DATE('&fecha_fin', 'YYYY-MM-DD')
GROUP BY Servicio_Salud.nombre
ORDER BY cantidad_solicitudes DESC
FETCH FIRST 20 ROWS ONLY;
```

Tablas y atributos utilizados

1. Servicio_Salud

- `id_servicio`: Identificador único del servicio de salud (Clave primaria).
- `nombre`: Nombre del servicio de salud.

2. Cita

- `id_cita`: Identificador único de la cita (Clave primaria).
- `id_servicio`: Identificador del servicio de salud solicitado en la cita (Clave foránea a `Servicio_Salud.id_servicio`).
- `fecha_hora`: Fecha y hora en la que se agendó la cita.

Explicación del INNER JOIN: Se utiliza `INNER JOIN` para unir la tabla `Cita` con `Servicio_Salud`, asegurando que solo se obtengan servicios de salud que han sido efectivamente solicitados en citas registradas.Si un servicio no tiene citas asociadas, no aparecerá en el resultado. La consulta filtra las citas dentro de un rango de fechas especificado por el usuario.

Consulta RFC3

```
 SELECT
    Servicio_Salud.nombre AS nombre_servicio,
    COUNT(Cita.id_cita) AS servicios_usados,
    (SELECT COUNT(*) FROM Servicio_Salud) AS servicios_disponibles,
    ROUND(COUNT(Cita.id_cita) / NULLIF((SELECT COUNT(*) FROM Servicio_Salud), 0), 2) AS indice_uso
FROM Cita
JOIN Servicio_Salud ON Cita.id_servicio = Servicio_Salud.id_servicio
WHERE
    Cita.fecha_hora BETWEEN TO_DATE('&fecha_inicio', 'YYYY-MM-DD')
                        AND TO_DATE('&fecha_fin', 'YYYY-MM-DD')
GROUP BY Servicio_Salud.nombre
ORDER BY indice_uso DESC;
```

Tablas y atributos utilizados

1. Servicio_Salud

- `id_servicio`: Identificador único del servicio de salud (Clave primaria).
- `nombre`: Nombre del servicio de salud.

2. Cita

- `id_cita`: Identificador único de la cita (Clave primaria).
- `id_servicio`: Identificador del servicio de salud solicitado en la cita (Clave foránea a `Servicio_Salud.id_servicio`).
- `fecha_hora`: Fecha y hora en la que se agendó la cita.

Explicación del INNER JOIN: Se utiliza `INNER JOIN` para unir la tabla `Cita` con `Servicio_Salud`, asegurando que solo se analicen los servicios de salud que han sido efectivamente utilizados en citas registradas. Si un servicio no ha sido usado en citas dentro del rango de fechas ingresado, no aparecerá en el resultado. La consulta permite calcular el índice de uso de cada servicio en función de la cantidad de veces que ha sido solicitado en citas.

Consulta RFC4

```sql
SELECT
    Servicio_Salud.nombre AS nombre_servicio,
    Cita.fecha_hora AS fecha_utilizacion,
    Medico.nombre AS nombre_medico,
    IPS.nombre AS nombre_ips
FROM Cita
JOIN Servicio_Salud ON Cita.id_servicio = Servicio_Salud.id_servicio
JOIN Medico ON Cita.id_medico = Medico.id_medico
JOIN IPS ON Cita.nit_ips = IPS.nit
WHERE
    Cita.id_afiliado = &id_afiliado
    AND Cita.fecha_hora BETWEEN TO_DATE('&fecha_inicio', 'YYYY-MM-DD')
                            AND TO_DATE('&fecha_fin', 'YYYY-MM-DD')
ORDER BY Cita.fecha_hora;
```

Tablas y atributos utilizados

1. Cita

- `id_cita`: Identificador único de la cita (Clave primaria).
- `id_servicio`: Identificador del servicio de salud utilizado en la cita (Clave foránea a `Servicio_Salud.id_servicio`).
- `id_medico`: Identificador del médico que atendió la cita (Clave foránea a `Medico.id_medico`).
- `nit_ips`: Identificador de la IPS donde se realizó la cita (Clave foránea a `IPS.nit`).
- `id_afiliado`: Identificador del afiliado que tomó la cita.
- `fecha_hora`: Fecha y hora en que se realizó la cita.

2. Servicio_Salud

- `id_servicio`: Identificador único del servicio de salud (Clave primaria).
- `nombre`: Nombre del servicio de salud.

3. Medico

- `id_medico`: Identificador único del médico (Clave primaria).
- `nombre`: Nombre del médico.

4. IPS

- `nit`: Identificador único de la IPS (Clave primaria).
- `nombre`: Nombre de la IPS.

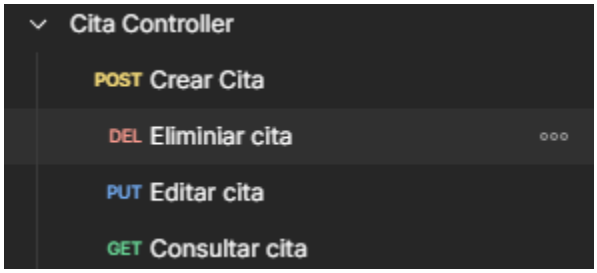
Explicación del INNER JOIN: Se utiliza `INNER JOIN` en todas las relaciones para asegurar que solo se muestren citas que tienen registros en todas las tablas involucradas (servicio, médico e IPS).Si una cita no tiene un servicio registrado, un médico asignado o una IPS asociada, no aparecerá en los resultados. La consulta filtra los servicios utilizados por un afiliado específico en un rango de fechas determinado**.**

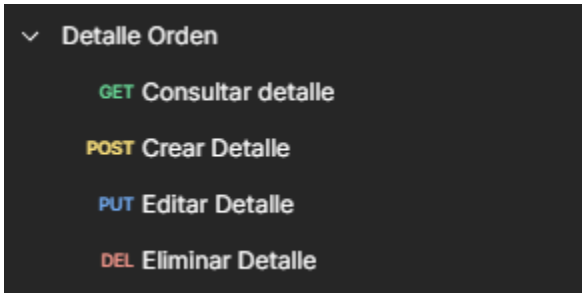
Punto 5.

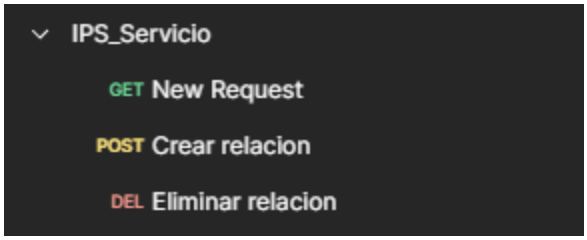
El script.sql se encuentra en el documento “Poblacion punto 5.sql” en la carpeta “docs” en el repositorio.

Punto 6

Escenarios de prueba: realice las inserciones descritas abajo para poder realizar pruebas para los RF. Ver sección “Escenarios de prueba” de este documento. a. Implemente los escenarios de prueba para cada requerimiento funcional (ver secciones de Requerimientos funcionales del documento marco del caso de estudio (RF1-RF8)), con el fin de verificar la persistencia de los datos y el manejo de las restricciones de integridad por parte el SMBD. b. Incluya la traza de la ejecución de las pruebas (scripts de inserción y pantallazos). c. Anexe los scripts SQL para estas operaciones y pantallazos de lo obtenido.







Se diseñaron y ejecutaron pruebas sobre los endpoints REST de los controladores (AfiliadoController, CitaController, DetalleOrdenController y IPS_ServicioController) utilizando Postman.

Para cada endpoint se verificó:

La correcta respuesta del servidor con los códigos de estado esperados (200 OK, 201 Created o 404 Not Found).

Que las operaciones de creación (POST) devolvieran mensajes de confirmación indicando que la entidad o relación fue creada exitosamente.

Que las operaciones de consulta (GET) devolvieran datos completos en formato JSON o un estado 404 si no existía el recurso.

Que las operaciones de edición (POST) devolvieran un mensaje de confirmación de actualización exitosa.

Que las operaciones de eliminación (GET) devolvieran un mensaje de confirmación indicando la eliminación exitosa de la entidad o relación.

Además, en cada prueba se incluyeron scripts automáticos de validación en Postman para verificar el contenido de la respuesta y la estructura esperada de los datos.