

Proyecto 1 Sistemas Transaccionales

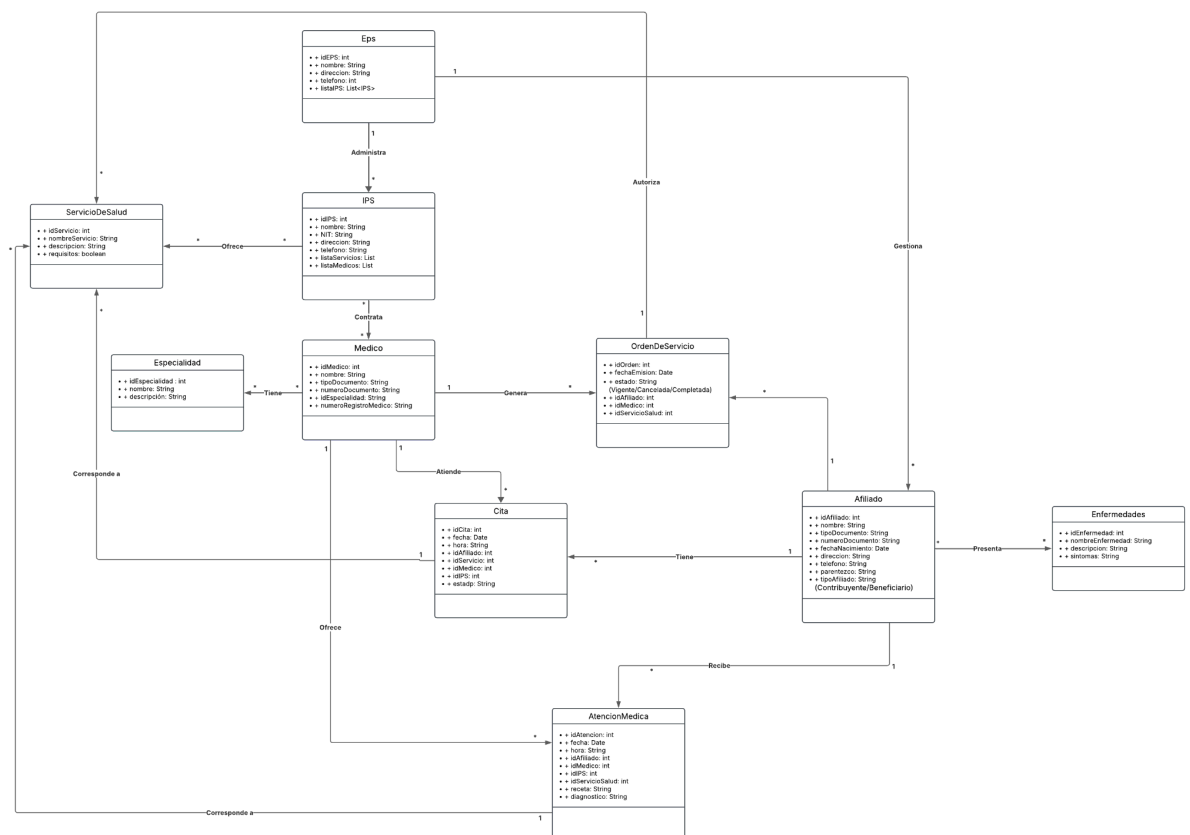
Grupo: Andrea Aroca, Paula Carreño, Santiago Delgado y Samuel Ovalle

a) Diagrama UML (todos):

➤ Actores (clases) principales del caso:

- **EPS:** Administra afiliados y contrata IPS para prestar servicios de salud.
- **Afiliado:** Persona inscrita en la EPS que usa sus servicios médicos
- **Enfermedades:** Problemas de salud que los afiliados pueden tener.
- **Médicos:** Profesionales que atienden pacientes y dan órdenes médicas
- **Especialidades:** Lista de especialidades posibles de cada médico
- **IPS:** Clínicas y hospitales que prestan servicios de salud
- **Orden Servicio:** Autorización de un médico para un tratamiento o consulta
- **Cita:** Fecha y hora programada para un servicio de salud.
- **Servicio Salud:** Consulta, examen o tratamiento ofrecido por una IPS.
- **Atención Médica:** Registro del momento en que un paciente recibe un servicio de salud

➤ Imagen UML:



Este diagrama UML representa el caso de EPSAndes, cada entidad está representada por una clase que contiene los atributos para describir su función dentro del sistema y las relaciones necesarias para conectar los procesos.

La clase **EPS** administra los servicios para sus afiliados y contratan a las IPS para la prestación de servicios médicos. Esta clase incluye atributos como su identificación, nombre y dirección. Por su parte, las **IPS** son los hospitales o clínicas que brindan los servicios de salud a los afiliados, contiene datos como su nombre, dirección y los médicos asociados, servicios. etc.

Los **Afiliados** son los usuarios del sistema que tienen un nombre, fecha de nacimiento y contacto, así como su tipo de afiliación (contributivo o beneficiario), si tiene un parentesco (familiar también registrado en la eps e identificación. Estos afiliados pueden estar relacionados con una o varias **Enfermedades**, las cuales están descritas en otra clase que incluye su identificador, nombre, descripción y síntomas.

El sistema también incluye a los **Médicos**, quienes son los responsables de atender a los afiliados y emitir órdenes de servicio. Esta clase tiene atributos como nombre, número de documento, número de registro médico para validar su licencia y la especialidad que poseen. Esta última está definida en la clase **Especialidad**, que describe las áreas médicas en las que los profesionales están capacitados para atender.

La clase **Servicio de Salud** detalla los diferentes tipos de consultas, exámenes o tratamientos ofrecidos por las IPS. Para acceder a estos servicios, los médicos emiten una **Orden de Servicio**, que sirve como autorización oficial para que un afiliado reciba el tratamiento o consulta necesarios. Estas órdenes contienen información como la fecha de emisión y el servicio al que están asociadas.

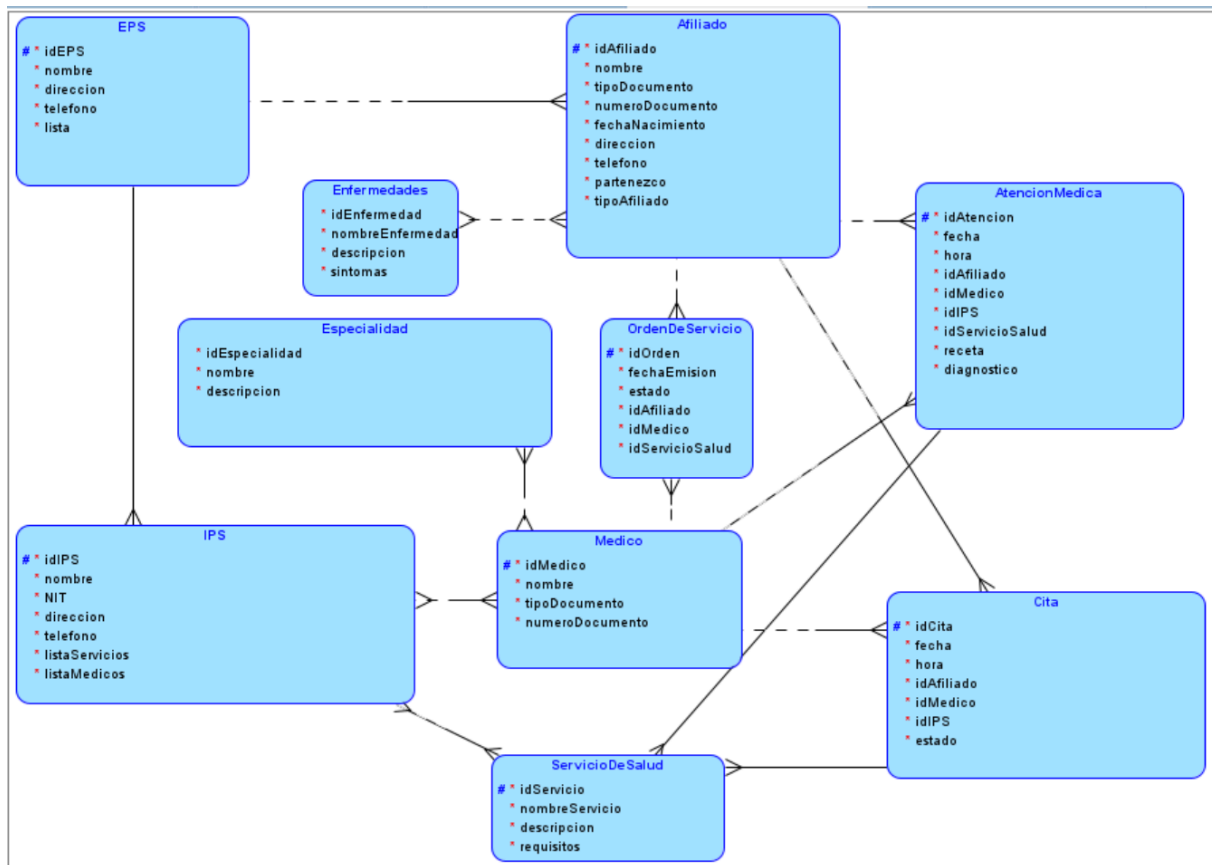
La clase **Citas** tiene atributos como la fecha y hora de la cita, el médico asignado y el servicio relacionado. Finalmente, la clase **Atención Médica** donde se registra el servicio prestado guardando información como el diagnóstico realizado y los resultados obtenidos durante la atención.

➤ Link diagrama UML:

https://lucid.app/lucidchart/7d71d573-e84c-47ff-bdb8-db42fb33b3de/edit?viewport_lo c=-753%2C-1326%2C4498%2C2008%2C0_0&invitationId=inv_445efc7a-928d-452b-9325-97d84bea8819

➤

b) Modelo E/R:



c) Modelo en Tablas

EPS			
idEPS (PK)	nombre	ubicación	telefono
PK, ND, NN	NN	NN	NN
1	Sanitas	Bogota	35678

EPS-IPS	
idEPS	idIPS
FK, NN	FK, NN
1	3
1	4
1	5

AFILIADO									
idAfiliado	nombre	tipoDocumento	numDocumento	fechaNacimiento	direccion	telefono	cedulaCotizante	tipoAfiliado	parentesco
PK, ND, NN	NN	NN	NN, ND	NN	NN	NN	opcional	NN	opcional
1	juan	cedula	123456789	11/02/1980	Calle 116 # 20-30	3205673849	123456789	contribuyente	
2	maria	cedula	987654321	9/05/2007	Calle 116 # 20-30	3115679840	123456789	Beneficiarios	Papá

Enfermedades			
idEnfermedad	nombreEnfermedad	descripcion	sintomas
PK, ND, NN	NN	opcional	opcional
3	Hipertencion	Persona con niveles altos de tensión	Dolor de cabeza, palidez
4	Apendicitis	Inflamación del apéndice, bolsa adherida al intestino grueso	Dolor al lado derecho del estomago

Presenta	
idAfiliado	idEnfermedad
PK, FK, NN	PK, FK, NN
1	3
1	4

IPS				
idIPS	nombre	NIT	direccion	telefono
PK, ND, NN	NN	NN, ND	NN	NN
1	Clinica del country	1234	Calle 100 #15-20	6173840

SERVICIO-SALUD			
idServicio	nombreServicio	descripcion	requisitos
PK, ND, NN	NN	opcional	NN
7	Consulta General	Atencion medica general	Cita previa obligatoria

SERVICIOS-IPS	
idIPS	idServicio
FK, NN	FK, NN
1	7

MEDICOS			
idMedico	nombre	tipoDocumento	numDocumento
PK, ND, NN	NN	NN	NN, ND
10	Dr. Martinez	CC	51479803

MEDICOS-IPS	
idMedico	idIPS
FK, NN	FK, NN
10	1

ESPECIALIDAD		
idEspecialidad	nombre	descripcion
PK, ND, NN	NN	opcional
1	Cardiologia	Especialidad dedicada al sistema cardiovascular

ESPECIALIDAD-MEDICO	
idMedico	idEspecialidad
FK, NN	FK, NN
10	1

CITA						
idCita	fecha	hora	idAfiliado	idMedico	idIPS	estado
PK, ND, NN	NN	NN	FK, NN	FK, NN	FK, NN	NN
101	17/02/2025	10:00	1	10	1	programada

ATENCIONMEDICA								
idAtencionMedica	idCita	fecha	hora	idAfiliado	idIPS	idServicio	receta	diagnostico
PK, ND, NN	FK, NN	NN	NN	FK, NN	FK, NN	FK, NN	opcional	NN
201	101	17/02/2025	10:00	1	1	10	Paracetamol 500mg	Gripa aguda

ORDENSERVICIO					
idOrden	fechaEmision	estado	idAfiliado	idMedico	idServicio
PK, ND, NN	NN	NN	FK, NN	FK, NN	FK, NN
301	17/02/2025	emitido	1	10	7

d) Verificación del nivel de normalización de las tablas

1. Primera Forma Normal (1FN)

Para cumplir con la 1FN, cada atributo debe ser atómico, es decir, no debe contener listas ni conjuntos de valores. Todas las tablas en el modelo EPSANDES cumplen con esta condición, ya que cada atributo contiene un único valor y no existen atributos multivaluados.

2. Segunda Forma Normal (2FN)

Para cumplir con la 2FN, es necesario que cada atributo dependa completamente de la clave primaria y no existan dependencias parciales. En este modelo:

- Todas las relaciones con claves primarias simples cumplen con la 2FN.
- Las relaciones con claves compuestas han sido diseñadas correctamente para evitar dependencias parciales.
- Se garantiza que los atributos en cada tabla dependen únicamente de su clave primaria.

3. Tercera Forma Normal (3FN)

Para cumplir con la 3FN, no deben existir dependencias transitivas entre atributos. La mayoría de las tablas en EPSANDES cumplen con esta condición excepto OrdenDeServicio y Cita donde el estado de la orden podría depender transitivamente del afiliado y el servicio asignado.

4. Forma Normal de Boyce-Codd (BCNF)

Para que una tabla esté en BCNF, todas las dependencias funcionales deben ser determinadas por una superllave. Dado que la mayoría de las tablas en EPSANDES cumplen con la 3FN sin problemas adicionales, también cumplen con BCNF. La única excepción es OrdenDeServicio, donde se detectó una posible dependencia transitoria que puede requerir una descomposición adicional.

Cambios:

Cita: El campo estado depende transitivamente de idAfiliado y idServicio, no solo de la clave primaria idCita.

Problema: estado podría depender de idAfiliado y idServicio, generando una dependencia transitiva.

Solución: Separar el estado en una tabla aparte:

- CitaBase (idCita PK, fecha, hora, idAfiliado FK, idMedico FK, idIPS FK)
- EstadoCita (idCita PK, estado)

Nueva estructura:

CitaBase

idCita (PK)	fecha	hora	idAfiliado (FK)	idMedico (FK)	idIPS (FK)
101	17/02/2025	10:00	1	10	1

EstadoCita

idCita (PK, FK)	Estado
101	programada

OrdenServicio: El estado de la orden podría depender de idAfiliado y idServicio en lugar de la clave primaria completa.

Problema: El estado podría depender de idAfiliado y idServicio, en lugar de la clave primaria completa.

Solución: Descomponer la relación:

- OrdenBase (idOrden PK, fechaEmision, idAfiliado FK, idMedico FK, idServicio FK)
- EstadoOrden (idOrden PK, estado)

Nueva estructura:

OrdenBase

idOrden (PK)	fechaEmision	idAfiliado (FK)	idMedico (FK)	idServicio(FK)
301	17/02/2025	1	10	7

EstadoOrden

idOrden (PK, FK)	estado
301	emitido

e) **Escenarios de prueba (Santiago → Sábado/Domingo):**

1. Pruebas de Funcionalidad

RF1 - Registrar IPS

Escenario:

- **Caso exitoso:** Se registra una IPS con un nombre único, NIT válido y dirección.
- **Caso fallido:** Intentar registrar una IPS con un NIT ya existente (debe fallar).

RF2 - Registrar un Servicio de Salud

Escenario:

- **Caso exitoso:** Se registra un servicio de salud con una descripción válida.
- **Caso fallido:** Intentar registrar un servicio sin descripción (debe fallar).

RF3 - Asignar un Servicio de Salud a una IPS

Escenario:

- **Caso exitoso:** Se asigna un servicio de salud a una IPS existente.
- **Caso fallido:** Intentar asignar un servicio a una IPS que no existe (debe fallar).

RF4 - Registrar Médico

Escenario:

- **Caso exitoso:** Se registra un médico con una especialidad válida.
- **Caso fallido:** Intentar registrar un médico sin número de registro (debe fallar).

RF5 - Registrar Afiliado

Escenario:

- **Caso exitoso:** Se registra un afiliado con un tipo de documento válido.
- **Caso fallido:** Intentar registrar un afiliado con un número de documento repetido (debe fallar).

RF6 - Registrar una Orden de Servicio de Salud

Escenario:

- **Caso exitoso:** Se registra una orden de servicio para un afiliado con un médico existente.
- **Caso fallido:** Intentar registrar una orden con un servicio que no existe (debe fallar).

RF7 - Agendar un Servicio de Salud

Escenario:

- **Caso exitoso:** Se agenda un servicio en una fecha y hora disponibles.
- **Caso fallido:** Intentar agendar un servicio en una fecha sin disponibilidad (debe fallar).

RF8 - Registrar la Prestación de un Servicio de Salud

Escenario:

- **Caso exitoso:** Se registra una atención médica realizada a un afiliado.
- **Caso fallido:** Intentar registrar una prestación sin médico asociado (debe fallar).

2. Pruebas de Integridad en la Base de Datos

Pruebas de unicidad de tuplas

- Insertar una tupla con una clave primaria válida.
- Intentar insertar otra tupla con la misma clave primaria (debe fallar).

Pruebas de integridad referencial

- Insertar una orden de servicio con un afiliado y médico válidos (debe ser exitoso).
- Intentar insertar una orden de servicio con un afiliado que no existe (debe fallar).

Pruebas de restricciones de chequeo

- Intentar insertar una cita médica en una fecha fuera del horario permitido (debe fallar).
- Intentar insertar un afiliado menor de edad como contribuyente (debe fallar).

3. Pruebas de Consultas y Estadísticas

RFC1 - Consultar la Agenda de Disponibilidad

Escenario:

- **Caso exitoso:** Se consulta la disponibilidad de un servicio en las próximas 4 **semanas**.
- **Caso fallido:** Intentar consultar disponibilidad de un servicio inexistente (debe fallar).

RFC2 - Mostrar los 20 Servicios más Solicitados

Escenario:

- **Caso exitoso:** Se muestra el ranking de los 20 servicios más solicitados en un periodo.
- **Caso fallido:** Intentar consultar sin definir un rango de fechas válido (debe fallar).

RFC3 - Mostrar el Índice de Uso de los Servicios

Escenario:

- **Caso exitoso:** Se calcula el índice de uso correctamente.
- **Caso fallido:** Intentar calcular el índice con un servicio sin registros (debe mostrar 0).

RFC4 - Mostrar Utilización de Servicios por un Afiliado

Escenario:

- **Caso exitoso:** Se consulta correctamente el historial de servicios de un afiliado.
- **Caso fallido:** Intentar consultar sin especificar un afiliado (debe fallar).

1. Análisis y revisión del modelo de datos:

2. Creación de la base de datos utilizando SQL Developer:

3. Aplicación usando Java Spring:

4. Sentencias SQL para los requerimientos de consulta (RFC1 a RFC4):

a. Documentación las tablas usadas:

- 1. RFC1 - CONSULTAR LA AGENDA DE DISPONIBILIDAD QUE TIENE UN SERVICIO DE SALUD INGRESADO POR EL USUARIO EN LAS SIGUIENTES 4 SEMANAS.**

Tablas usadas:

IPS_MEDICO: Relaciona médicos con IPS y servicios.

AGENDA: Contiene los horarios disponibles y asignados para los médicos.

SERVICIO_SALUD: Define los servicios de salud disponibles.

IPS: Contiene información de las Instituciones Prestadoras de Salud.

MEDICO: Almacena los datos de los médicos.

Joins:

IPS_MEDICO.ID_IPS_MEDICO = AGENDA.ID_IPS_MEDICO: Para relacionar los horarios de agenda con los médicos y servicios.

IPS_MEDICO.ID_SERVICIO = SERVICIO_SALUD.ID_SERVICIO: Para filtrar por un servicio específico.

IPS_MEDICO.ID_IPS = IPS.ID_IPS: Para obtener el nombre de la IPS.

IPS_MEDICO.ID_MEDICO = MEDICO.ID_MEDICO: Para obtener el nombre del médico.

Explicación del join:

Se usó un INNER JOIN implícito (en el WHERE) porque solo nos interesan los registros que tienen coincidencias en todas las tablas. Esto asegura que solo se muestren las citas disponibles para el servicio específico (ID 13) en el rango de fechas dado.

Propósito:

Esta consulta muestra las citas disponibles para un servicio específico (en este caso, Dermatología, ID 13) en un rango de fechas, incluyendo la fecha, hora, nombre de la IPS y nombre del médico.

Resultados obtenidos:

Dermatología | 01/03/24 07:00 | Clínica Santa Fe | Dra. Laura Torres

Dermatología | 01/03/24 08:00 | Clínica Santa Fe | Dra. Laura Torres

Dermatología | 01/03/24 08:30 | Clínica Santa Fe | Dra. Laura Torres

Dermatología | 01/03/24 15:30 | Clínica Santa Fe | Dra. Laura Torres

Dermatología | 05/03/24 13:30 | Clínica Santa Fe | Dra. Laura Torres

Dermatología | 15/03/24 09:30 | Clínica Santa Fe | Dra. Laura Torres

Dermatología | 20/03/24 14:30 | Clínica Santa Fe | Dra. Laura Torres

Dermatología | 29/03/24 16:30 | Clínica Santa Fe | Dra. Laura Torres

Total de filas seleccionadas: 8.

2. RFC2 - MOSTRAR LOS 20 SERVICIOS MÁS SOLICITADOS. Los que fueron más solicitados en un período de tiempo dado:

Tablas usadas:

ORDEN_BASE: Contiene las órdenes médicas emitidas.

IPS_MEDICO: Relaciona médicos con IPS y servicios.

IPS: Contiene información de las Instituciones Prestadoras de Salud.

EPS: Almacena los datos de las EPS.

MEDICO: Contiene los datos de los médicos.

SERVICIO_SALUD: Define los servicios de salud.

Joins:

IPS_MEDICO.ID_IPS = IPS.ID_IPS: Para obtener el nombre de la IPS.

IPS_MEDICO.ID_EPS = EPS.ID_EPS: Para obtener el nombre de la EPS.

IPS_MEDICO.ID_MEDICO = MEDICO.ID_MEDICO: Para obtener el nombre del médico.

IPS_MEDICO.ID_SERVICIO = SERVICIO_SALUD.ID_SERVICIO: Para obtener la descripción del servicio.

Explicación del join:

Se usó un INNER JOIN implícito (en el WHERE) porque solo nos interesan los registros que tienen coincidencias en todas las tablas. Esto asegura que solo se muestren los detalles de la IPS específica (ID 104) y sus servicios asociados.

Propósito:

La primera parte de la consulta cuenta el número total de órdenes médicas por servicio en un rango de fechas.

La segunda parte muestra los detalles de la IPS con ID 104, incluyendo el nombre de la IPS, la EPS a la que pertenece, el nombre del médico y el servicio que ofrece.

Resultados obtenidos:

Primera parte (órdenes médicas por servicio):

ID_SERVICIO | TOTAL_ORDENES

6 | 1

7 | 1

2 | 1

8 | 1

9 | 1

4	1
5	1
10	1
3	1
11	1

Total de filas seleccionadas: 10.

Segunda parte (detalles de la IPS 104):

ID_IPS	NOMBRE_IPS	NOMBRE_EPS	NOMBRE_MEDICO	DESCRIPCION
104	Clínica Santa Fe	EPS SANITAS	Dra. Laura Torres	Dermatología
104	Clínica Santa Fe	EPS SANITAS	Dra. Camila Suárez	Urgencias

3. RFC3 - MOSTRAR EL ÍNDICE DE USO DE CADA UNO DE LOS SERVICIOS PROVISTOS

Tablas usadas:

SERVICIO_SALUD: Define los servicios de salud disponibles.

CITA_BASE: Contiene las citas médicas programadas.

Joins:

SERVICIO_SALUD.ID_SERVICIO = CITA_BASE.ID_SERVICIO: Para relacionar las citas con los servicios.

Explicación del join:

Se usó un LEFT JOIN porque queremos incluir todos los servicios disponibles, incluso si no han sido usados en el período de tiempo especificado. Esto asegura que se muestren todos los servicios, no solo aquellos con citas programadas.

Propósito:

Esta consulta calcula el índice de uso de cada servicio en un período de tiempo específico. El índice se obtiene dividiendo el número de servicios usados (citas) entre el total de servicios disponibles.

ID_SERVICIO	SERVICIO	SERVICIOS_USADOS	SERVICIOS_DISPONIBLES	INDICE_USO
1	Consulta general	0	15	0
2	Urgencias	0	15	0

3	Hospitalización	0	15	0
4	Cirugía ambulatoria	0	15	0
5	Laboratorio clínico	0	15	0
6	Radiología e imágenes	0	15	0
7	Fisioterapia	0	15	0
8	Psicología	1	15	0.07
9	Odontología general	0	15	0
10	Medicina interna	0	15	0
11	Pediatría	1	15	0.07
12	Ginecología y obstetricia	1	15	0.07
13	Dermatología	0	15	0
14	Cardiología	0	15	0
15	Oftalmología	0	15	0

Total de filas seleccionadas: 15.

4. RFC4 - MOSTRAR LA UTILIZACIÓN DE SERVICIOS DE EPSANDES POR UN AFILIADO DADO, EN UN PERIODO DADO (RANGO DE FECHAS INDICADO).

Tablas usadas:

CITA_BASE: Contiene las citas médicas programadas.

AFILIADO: Almacena los datos de los afiliados.

MEDICO: Contiene los datos de los médicos.

IPS: Contiene información de las Instituciones Prestadoras de Salud.

Joins:

CITA_BASE.ID_AFILIADO = AFILIADO.ID_AFILIADO: Para obtener el nombre del afiliado.

CITA_BASE.ID_MEDICO = MEDICO.ID_MEDICO: Para obtener el nombre del médico.

CITA_BASE.ID_IPS = IPS.ID_IPS: Para obtener el nombre de la IPS.

Explicación del join:

Se usó un INNER JOIN implícito (en el WHERE) porque solo nos interesan los registros que tienen coincidencias en todas las tablas. Esto asegura que solo se muestren las citas del afiliado específico (ID 2019) en el rango de fechas dado.

Propósito:

Esta consulta muestra todas las citas de un afiliado específico (ID 2019) en un rango de fechas, incluyendo la fecha de la cita, el nombre del médico, el nombre de la IPS y el número total de citas.

Resultados obtenidos:

ID_AFILIADO | NOMBRE_AFILIADO | FECHA_CITA | NOMBRE_MEDICO | NOMBRE_IPS | TOTAL_CITAS

2019 | Santiago López | 08/03/24 | Dr. Daniel Pineda | Clínica Universitaria | 1

5. Población de Tablas:

Se realizó la población de las tablas de la base de datos con información de prueba para simular el funcionamiento del sistema en un entorno real. Lo que se hizo en cada tabla:

- En la tabla AFILIADO, se insertaron datos de personas afiliadas a una EPS, incluyendo sus nombres, fechas de nacimiento, tipo de afiliación (contributivo o beneficiario) y la EPS a la que pertenecen.
- En la tabla AFILIADO_ENFERMEDAD, se relacionaron algunos afiliados con enfermedades específicas, indicando la fecha en que fueron diagnosticados.
- En la tabla ATENCION_MEDICA, se agregaron detalles de atenciones médicas realizadas, como diagnósticos y resultados.
- En la tabla CITA_BASE, se insertaron datos de citas médicas programadas, incluyendo la fecha, hora, el afiliado, el médico, la IPS y el servicio a prestar.
- En la tabla ENFERMEDAD, se incluyó una lista de enfermedades comunes con sus síntomas y descripciones.
- En la tabla EPS, se agregaron datos de EPS, como su nombre y dirección de ejemplo.
- En la tabla ESPECIALIDAD, se insertaron las especialidades médicas disponibles, como cardiología, neurología, entre otras.
- En la tabla ESTADO_CITA, se definió el estado de algunas citas (pendientes o completadas).
- En la tabla ESTADO_ORDEN, se indicó el estado de las órdenes médicas (pendientes, canceladas o atendidas).
- En la tabla IPS, se incluyó información de instituciones prestadoras de salud (IPS), con su nombre, dirección y la EPS a la que están asociadas.
- En la tabla IPS_SERVICIO, se relacionaron las IPS con los servicios de salud que ofrecen.

- En la tabla MEDICO, se insertaron datos de médicos, incluyendo su nombre, registro médico y especialidad.
- En la tabla ORDEN_BASE, se agregaron detalles de órdenes médicas, con la fecha de emisión, el afiliado, el médico y el servicio solicitado.
- En la tabla SERVICIO_SALUD, se incluyó una lista de servicios de salud, como consultas generales, urgencias, hospitalización, entre otros.
- En la tabla ESPECIALIDAD_SERVICIO, se relacionaron las especialidades médicas con los servicios que pueden ofrecer.
- En la tabla IPS_MEDICO, se vincularon los médicos con las IPS en las que trabajan y los servicios que brindan.
- Finalmente, en la tabla AGENDA, se insertaron horarios disponibles para 2 médicos en específico y en una ips para un mes.

En resumen, se simuló un sistema de salud completo con datos de prueba, permitiendo visualizar cómo interactúan las diferentes entidades.

6. Escenarios de prueba:

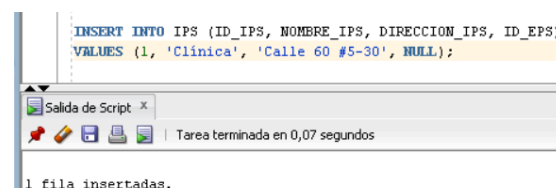
1. Pruebas de Unicidad de Tuplas

a. Caso Exitoso: Insertar una IPS con un NIT único

■ Sentencia SQL:

```
INSERT INTO IPS (ID_IPS, NOMBRE_IPS, DIRECCION_IPS, ID_EPS)
```

```
VALUES (1, 'Clínica', 'Calle 60 #5-30', NULL);
```



b. Caso Fallido: Insertar una IPS con un NIT ya existente

■ Sentencia SQL:

```
INSERT INTO IPS (ID_IPS, NOMBRE_IPS, DIRECCION_IPS, ID_EPS)
```

```
VALUES (1, 'Hospital Norte', 'Calle 24 #7-40', NULL);
```

■ Error Esperado: Violación de restricción de unicidad en NIT

```
INSERT INTO IPS (ID_IPS, NOMBRE_IPS, DIRECCION_IPS, ID_EPS)
VALUES (1, 'Hospital Norte', 'Calle 24 #7-40', NULL);
```

Salida de Script x

Tarea terminada en 0,112 segundos

Error que empieza en la línea: 176 del comando :

```
INSERT INTO IPS (ID_IPS, NOMBRE_IPS, DIRECCION_IPS, ID_EPS)
VALUES (1, 'Hospital Norte', 'Calle 24 #7-40', NULL)
Informe de error -
ORA-00001: restricción única (ISIS2304A24202510.PK_IPS) violada
```

2. Pruebas de Integridad

a. Caso Exitoso: Insertar un médico con una especialidad válida

■ Sentencia SQL:

```
INSERT INTO ESPECIALIDAD (ID_ESPECIALIDAD,
NOMBRE_ESPECIALIDAD)
```

```
VALUES (1, 'Cardiología');
```

```
INSERT INTO MEDICO (ID_MEDICO, NOMBRE_MEDICO,
REGISTRO_MEDICO, ID_ESPECIALIDAD)
```

```
VALUES (1, 'Juan', 'MED-1234', 1);
```

```
INSERT INTO ESPECIALIDAD (ID_ESPECIALIDAD, NOMBRE_ESPECIALIDAD)
VALUES (1, 'Cardiología');

INSERT INTO MEDICO (ID_MEDICO, NOMBRE_MEDICO, REGISTRO_MEDICO, ID_ESPECIALIDAD)
VALUES (1, 'Juan', 'MED-1234', 1);
```

Salida de Script x

Tarea terminada en 0,113 segundos

Error que empieza en la línea: 219 del comando :

```
INSERT INTO ESPECIALIDAD (ID_ESPECIALIDAD, NOMBRE_ESPECIALIDAD)
VALUES (1, 'Cardiología')
Informe de error -
ORA-00001: restricción única (ISIS2304A24202510.PK_ESPECIALIDAD) violada

https://docs.oracle.com/error-help/db/ora-00001/

More Details :
https://docs.oracle.com/error-help/db/ora-00001/

Error que empieza en la línea: 222 del comando :
INSERT INTO MEDICO (ID_MEDICO, NOMBRE_MEDICO, REGISTRO_MEDICO, ID_ESPECIALIDAD)
VALUES (1, 'Juan', 'MED-1234', 1)
Informe de error -
ORA-00001: restricción única (ISIS2304A24202510.PK_MEDICO) violada
```

b. Caso Fallido:

■ Sentencia SQL: Insertar un médico con especialidad no existente

```
INSERT INTO MEDICO (ID_MEDICO, NOMBRE_MEDICO,
REGISTRO_MEDICO, ID_ESPECIALIDAD)
```

```
VALUES (2, 'Diana', 'MED-5678', 5);
```


- Error esperado: Violación de clave foránea en idEspecialidad

```
INSERT INTO MEDICO (ID_MEDICO, NOMBRE_MEDICO, REGISTRO_MEDICO, ID_ESPECIALIDAD)
VALUES (2, 'Diana', 'MED-5678', 5);
```

Salida de Script x Tarea terminada en 0,114 segundos

Error que empieza en la línea: 225 del comando :
 INSERT INTO MEDICO (ID_MEDICO, NOMBRE_MEDICO, REGISTRO_MEDICO, ID_ESPECIALIDAD)
 VALUES (2, 'Diana', 'MED-5678', 5)
 Informe de error -
 ORA-00001: restricción única (ISIS2304A24202510.PK_MEDICO) violada

3. Pruebas de restricción de chequeo

- Caso Exitoso:** Insertar un afiliado con fecha de nacimiento válida

- Sentencia SQL:

```
INSERT INTO AFILIADO (ID_AFILIADO,
NOMBRE_AFILIADO, FECHA_NACIMIENTO,
TIPO_AFILIACION)

VALUES (1, 'Andres Perez', DATE '2003-05-10', 'Contributivo');
```

```
INSERT INTO AFILIADO (ID_AFILIADO, NOMBRE_AFILIADO, FECHA_NACIMIENTO, TIPO_AFILIACION)
VALUES (1, 'Andres Perez', DATE '2003-05-10', 'Contributivo');
```

1 fila insertadas.

- Caso Fallido:** Insertar un afiliado con fecha de nacimiento inválida

- Sentencia SQL:

```
INSERT INTO AFILIADO (ID_AFILIADO,
NOMBRE_AFILIADO, FECHA_NACIMIENTO,
TIPO_AFILIACION)

VALUES (2, 'Pedrito', DATE '2028-09-20', 'Contributivo');
```

- Error esperado: La fecha de nacimiento debe ser anterior a la actual.

```
INSERT INTO AFILIADO (ID_AFILIADO, NOMBRE_AFILIADO, FECHA_NACIMIENTO, TIPO_AFILIACION)
VALUES (2, 'Pedrito', DATE '2028-09-20', 'Contributivo');
```

Salida de Script x

Tarea terminada en 0,106 segundos

1 fila insertadas.

4. Pruebas de funcionalidad:

a. Requerimiento Funcional 1 - Registrar IPS

■ Sentencia SQL:

```
INSERT INTO IPS (ID_IPS, NOMBRE_IPS, DIRECCION_IPS, ID_EPS)
VALUES (3, 'Centro Médico 1', 'Calle 60a #10-20', NULL);
```

```
INSERT INTO IPS (ID_IPS, NOMBRE_IPS, DIRECCION_IPS, ID_EPS)
VALUES (3, 'Centro Médico 1', 'Calle 60a #10-20', NULL);
```

Salida de Script x

Tarea terminada en 0,076 segundos

1 fila insertadas.

b. Requerimiento Funcional 2 - Registrar Servicio de Salud

■ Sentencia SQL:

```
INSERT INTO SERVICIO_SALUD (ID_SERVICIO,
DESCRIPCION)
VALUES (1, 'Consulta médica básica');
```

```
INSERT INTO SERVICIO_SALUD (ID_SERVICIO, DESCRIPCION)
VALUES (1, 'Consulta médica básica');
```

Salida de Script x

Tarea terminada en 0,194 segundos

1 fila insertadas.

c. Requerimiento Funcional 3 - Asignar un Servicio de Salud a IPS

■ Sentencia SQL:

```
INSERT INTO IPS_SERVICIO (ID_IPS, ID_SERVICIO) VALUES
(3, 1);
```

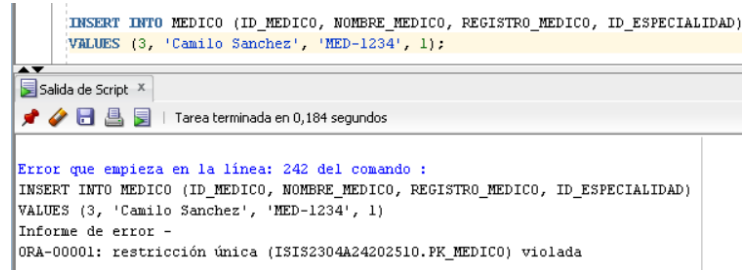
```
INSERT INTO IPS_SERVICIO (ID_IPS, ID_SERVICIO) VALUES (3, 1);
```

1 fila insertadas.

d. Requerimiento Funcional 4 - Registrar Médico

- Sentencia SQL:

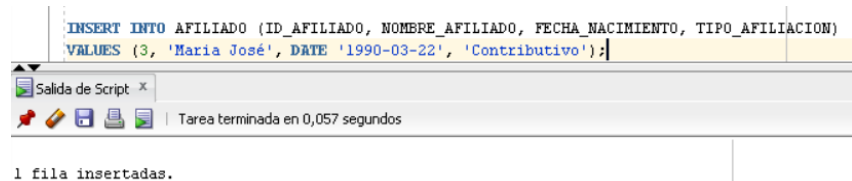
```
INSERT INTO MEDICO (ID_MEDICO, NOMBRE_MEDICO, REGISTRO_MEDICO, ID_ESPECIALIDAD)
VALUES (3, 'Camilo Sanchez', 'MED-1234', 1);
```



e. Requerimiento Funcional 5 - Registrar un Afiliado

- Sentencia SQL:

```
INSERT INTO AFILIADO (ID_AFILIADO, NOMBRE_AFILIADO, FECHA_NACIMIENTO, TIPO_AFILIACION)
VALUES (3, 'Maria José', DATE '1990-03-22', 'Contributivo');
```



f. Requerimiento Funcional 6 - Registrar una orden de servicio de salud para un afiliado por parte de un médico

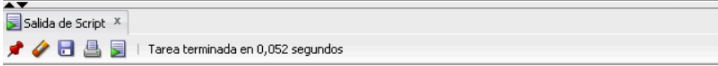
- Sentencia SQL:

```
INSERT INTO ORDEN_BASE (ID_ORDEN, FECHA_EMISION, ID_AFILIADO, ID_MEDICO, ID_SERVICIO)
VALUES (1, DATE '2025-03-01', 3, 3, 1);
```

```
INSERT INTO ESTADO_ORDEN (ID_ORDEN, ESTADO)
VALUES (1, 'Vigente');
```

```
INSERT INTO ORDEN_BASE (ID_ORDEN, FECHA_EMISION, ID_AFILIADO, ID_MEDICO, ID_SERVICIO)
VALUES (1, DATE '2025-03-01', 3, 3, 1);

INSERT INTO ESTADO_ORDEN (ID_ORDEN, ESTADO) VALUES (1, 'Vigente');
```



Salida de Script x

Tarea terminada en 0,052 segundos

1 fila insertadas.

1 fila insertadas.

g. Requerimiento Funcional 7 - Agendar un servicio de salud por parte de un afiliado

■ Sentencia SQL:

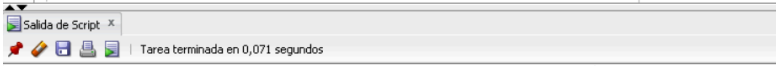
```
INSERT INTO CITA_BASE (ID_CITA, FECHA_CITA,
HORA_CITA, ID_AFILIADO, ID_MEDICO, ID_IPS,
ID_SERVICIO)
```

```
VALUES (1, DATE '2025-03-10', '10:00', 3, 3, 3, 1);
```

```
INSERT INTO ESTADO_CITA (ID_CITA, ESTADO) VALUES (1,
'Agendada');
```

```
INSERT INTO CITA_BASE (ID_CITA, FECHA_CITA, HORA_CITA, ID_AFILIADO, ID_MEDICO, ID_IPS, ID_SERVICIO)
VALUES (1, DATE '2025-03-10', '10:00', 3, 3, 3, 1);

INSERT INTO ESTADO_CITA (ID_CITA, ESTADO) VALUES (1, 'Agendada');
```



Salida de Script x

Tarea terminada en 0,071 segundos

1 fila insertadas.

1 fila insertadas.

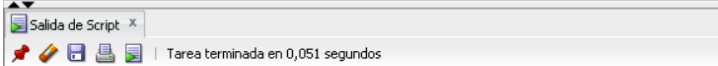
h. Requerimiento Funcional 8 - Registrar la prestación de un servicio de salud a un afiliado por parte de una IPS

■ Sentencia SQL:

```
INSERT INTO ATENCION_MEDICA (ID_ATENCION, ID_CITA,
DIAGNOSTICO, RESULTADOS)
```

```
VALUES (1, 1, 'Resfriado', 'Recetado Paracetamol');
```

```
INSERT INTO ATENCION_MEDICA (ID_ATENCION, ID_CITA, DIAGNOSTICO, RESULTADOS)
VALUES (1, 1, 'Resfriado', 'Recetado Paracetamol');
```



Salida de Script x

Tarea terminada en 0,051 segundos

1 fila insertadas.

ENTREGA 2

RFC5 – CONSULTA LA AGENDA DE DISPONIBILIDAD DE UN SERVICIO DE SALUD - SERIALIZABLE:

Este RFC tiene como objetivo consultar la agenda de disponibilidad de un servicio de salud, utilizando el nivel de aislamiento SERIALIZABLE para garantizar la consistencia y evitar posibles problemas de concurrencia, como la lectura no repetible o las lecturas fantasma.

Sentencia SQL:

```
SELECT    TO_CHAR(AGENDA.FECHA_HORA, 'DD/MM/YYYY HH24:MI:SS') AS
FECHA_HORA,
          IPS.NOMBRE_IPS,
          MEDICO.NOMBRE_MEDICO,
          SERVICIO_SALUD.DESCRIPCION

FROM AGENDA

JOIN IPS_MEDICO ON AGENDA.ID_IPS_MEDICO = IPS_MEDICO.ID_IPS_MEDICO

JOIN IPS ON IPS_MEDICO.ID_IPS = IPS.ID_IPS

JOIN MEDICO ON IPS_MEDICO.ID_MEDICO = MEDICO.ID_MEDICO

JOIN SERVICIO_SALUD ON IPS_MEDICO.ID_SERVICIO = SERVICIO_SALUD.ID_SERVICIO

WHERE AGENDA.FECHA_HORA BETWEEN TO_DATE('01/03/24', 'DD/MM/YY') AND
TO_DATE('16/03/24', 'DD/MM/YY')

AND AGENDA.ESTADO = 'DISPONIBLE'

AND (IPS_MEDICO.ID_SERVICIO IS NULL OR IPS_MEDICO.ID_SERVICIO = 13)

AND (IPS_MEDICO.ID_MEDICO IS NULL OR IPS_MEDICO.ID_MEDICO = 4)
```

- **Descripcion de la sentencia:** La sentencia SQL busca obtener información sobre la disponibilidad de agendas médicas en el sistema EPSANDES, filtrando por fechas, servicios de salud y médicos específicos. La consulta selecciona la fecha y hora de la tabla 'AGENDA', junto con el nombre de la IPS, el nombre del médico y la descripción del servicio de salud. Para ello, realiza joins entre las tablas 'AGENDA', 'IPS_MEDICO', 'IPS', 'MEDICO' y 'SERVICIO_SALUD', relacionándolas mediante sus claves foráneas.

Para probarlo, los filtros aplicados incluyen un rango de fechas entre el 1 y el 16 de marzo de 2024, donde el estado de la agenda debe ser "DISPONIBLE". Además, se filtra opcionalmente por el servicio de salud con ID 13 y el médico con ID 4, lo que significa que si estos parámetros son nulos, la consulta ignorará estos filtros. Por ejemplo, si no se

proporciona un ID de médico, la consulta devolverá todas las agendas disponibles para el servicio especificado dentro del rango de fechas.

- **Resultados:**

FECHA_HORA	NOMBRE_IPS	NOMBRE_MEDICO	DESCRIPCION
01/03/2024 08:15:00	Clínica Santa Fe	Dra. Laura Torres	Dermatología
01/03/2024 12:00:00	Clínica Santa Fe	Dra. Laura Torres	Dermatología
01/03/2024 14:45:00	Clínica Santa Fe	Dra. Laura Torres	Dermatología
01/03/2024 18:00:00	Clínica Santa Fe	Dra. Laura Torres	Dermatología
05/03/2024 09:30:00	Clínica Santa Fe	Dra. Laura Torres	Dermatología
15/03/2024 13:15:00	Clínica Santa Fe	Dra. Laura Torres	Dermatología

**Consideramos pertinente agregar una cuarta columna con el nombre de la IPS, ya que un mismo médico puede atender en diferentes instituciones. Esto permite a los afiliados identificar no solo la disponibilidad del médico, sino también el lugar exacto donde se brindará el servicio, evitando confusiones al agendar.

Configuración de la transacción:

```
@Transactional(isolation = Isolation.SERIALIZABLE, rollbackFor = Exception.class)
```

```
public List<Map<String, String>> consultarAgendaSerializable(LocalDateTime fechaInicio,
LocalDateTime fechaFin, Long idServicio, Long idMedico) throws InterruptedException {

    return ejecutarConsultaConTemporizador(fechaInicio, fechaFin, idServicio, idMedico);

}
```

Para este RFC, se establece el nivel de aislamiento **SERIALIZABLE**, lo que garantiza que la transacción sea completamente aislada y que no se produzcan efectos secundarios no deseados como lecturas no repetibles o lecturas fantasma. Esto proporciona la mayor consistencia posible al consultar la agenda de disponibilidad de un servicio de salud.

RFC6 – CONSULTAR LA AGENDA DE DISPONIBILIDAD DE UN SERVICIO DE SALUD - READ COMMITED:

Este RFC tiene el mismo objetivo que el RFC5, pero utiliza el nivel de aislamiento READ COMMITED para permitir una mayor concurrencia a expensas de una consistencia más estricta en las transacciones.

Sentencia SQL: Este requerimiento tiene la misma sentencia SQL descrita anteriormente, lo diferente el nivel de aislamiento de la transacción.

Configuración de la transacción:

```
@Transactional(isolation = Isolation.READ_COMMITTED, rollbackFor = Exception.class)
```

```

    public List<Map<String, String>> consultarAgendaReadCommitted(LocalDateTime fechaInicio,
        LocalDateTime fechaFin, Long idServicio, Long idMedico) throws InterruptedException {

        return ejecutarConsultaConTemporizador(fechaInicio, fechaFin, idServicio, idMedico);

    }

```

En este RFC, se establece el nivel de aislamiento **READ COMMITTED**, lo que permite leer solo datos que han sido confirmados por otras transacciones y evita que se lean datos sucios. Sin embargo, aún pueden ocurrir lecturas no repetibles.

RF9 – AGENDAR UN SERVICIO DE SALUD POR PARTE DE UN AFILIADO – TRANSACCIONAL:

Clase: Agendamiento

Descripción:

Administra la creación de agendamientos de servicios de salud de manera transaccional para garantizar la integridad de los datos.

Método relevante:

```
@Transactional(timeout = 30)
```

```

public void insertAgendamiento(Long idAfiliado, Long idServicio, String idMedico, Long idIPS,
    String fecha, String hora, Long ordenId)

```

Características:

- Utiliza manejo de transacciones para asegurar consistencia en el agendamiento.
- Válida si es necesario una orden previa dependiendo del tipo de servicio.
- Maneja rollback en caso de fallos en la asignación.

ESCENARIO DE PRUEBA DE CONCURRENCIA 1:

(Consulta RFC6)

Abrir dos consultas de disponibilidad al mismo tiempo, mientras un tercera agenda el servicio. Mostrar que las lecturas son consistentes y no hay lecturas sucias gracias a READ COMMITTED.

(RFC5 SERIALIZABLE vs RF9 Registro Transaccional)

Descripción del escenario:

Primero se ejecuta el RFC5: Consulta de la agenda de disponibilidad de un servicio de salud, utilizando aislamiento SERIALIZABLE y simulando 30 segundos de espera (Thread.sleep(30000)).

Antes de que pasen los 30 segundos, se ejecuta de manera concurrente el RF9, que es el registro de un nuevo agendamiento (orden de servicio de salud) para un afiliado.

Tiempo Acción

- t0 - Usuario A envía petición GET a /agendamientos/filtros/serializable (RFC5).
 - t5 - Usuario B, en paralelo, envía una petición POST a /agendamientos/crear (RF9).
 - t30 - Finaliza la consulta RFC5 (después de dormir 30s).
 - t32 - Se completa la inserción de la nueva orden de servicio en RF9.
-

Descripción de lo sucedido:

¿Acaso el componente que implementa RF9 debió esperar a que terminara la ejecución de la consulta RFC5 para poder registrar la orden de servicio?

El registro de la orden de servicio (RF9) no debió esperar a que la consulta RFC5 terminara su ejecución.

Al revisar nuevamente la tabla de agendamientos, vemos que no se agregó el nuevo servicio solicitado por RF9. Lo que nos da a entender que el uso de serializable bloquea las filas involucradas, por lo que no permite ninguna escritura concurrente hasta que la lectura termine.

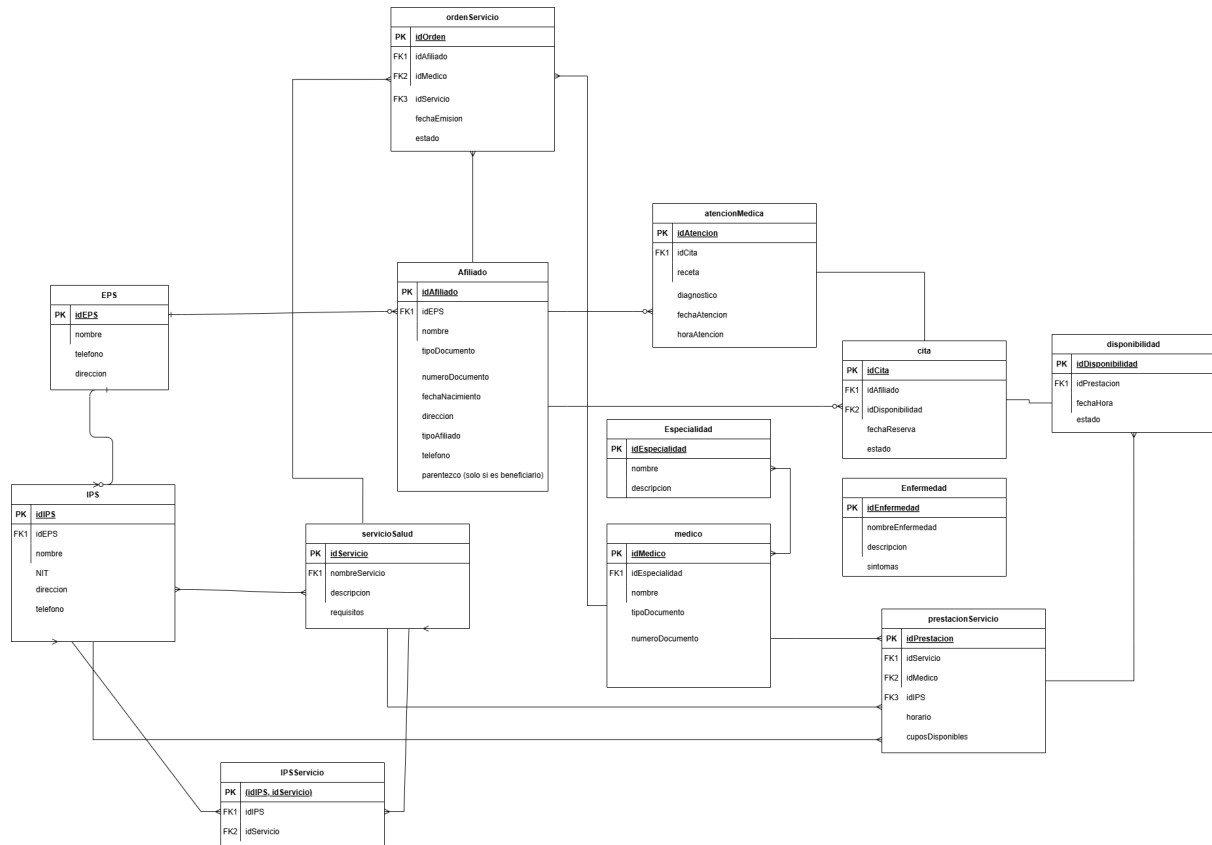
6. ESCENARIO DE PRUEBA DE CONCURRENCIA 2:

- **Los pasos para la ejecución concurrente de RFC6 y RF9 a través de la línea de tiempo**
 - **Paso 1:**
 - La base de datos debe tener al menos una entrada de servicio de salud disponible en la agenda.
 - **Paso 2:**
 - Ejecución de RFC6. El usuario debe enviar una solicitud para consultar la disponibilidad de un servicio de salud con las condiciones de rango de fechas, médico y servicio de salud.
 - **Paso 3:**
 - La consulta devuelve los resultados en función de los filtros proporcionados teniendo en cuenta que la transacción tiene un temporizador de 30 segundos.
 - **Paso 4:**
 - Ejecución de RF9. Mientras se ejecuta RFC6, otro usuario debe intentar agendar un servicio de salud utilizando el componente transaccional que implementa RF9.
 - Este proceso realiza una confirmación de disponibilidad del servicio seleccionado y agenda el servicio de salud si aún sigue disponible. En caso de no estar disponible, la transacción realiza rollback

- **Paso 5:**
 - La consulta RFC6 debe devolver la disponibilidad actual solamente de lo que haya sido confirmado.
 - Si se modifica después de la consulta el sistema no reflejará los cambios hasta que la transacción esté confirmada.
- **Descripción de lo sucedido: ¿acaso el componente que implementa RF9 debió esperar a que terminara la ejecución de la consulta RFC6 para poder registrar el ingreso de productos?**
 - No es necesario esperar a que la ejecución de la consulta RFC6 termine gracias a que el proceso de RF9 puede modificarse sin que haya finalizado RFC6. Sin embargo, en tal caso que no haya disponibilidad de agenda, RF9 debe realizar el rollback
- **El resultado presentado por RFC6: presente el resultado de esta consulta. Diga si allí apareció la orden de servicio ingresada al ejecutar el componente que implementa RF9 de manera simultánea.**
 - No aparece la orden de servicio ingresada en los resultados de RFC6, dado que el proceso de RF9 está en ejecución y el aislamiento READ COMMITTED permite leer datos comprometidos, sin embargo no garantiza que la transacción sea consistente entre las lecturas de las consultas concurrentes.

ENTREGA 3 FINAL

1. Revise el caso de estudio propuesto. Identifique los elementos fundamentales que hacen parte del negocio que se describe.
2. Análisis y modelo conceptual para todos los requerimientos funcionales (RF1-RF7)



https://drive.google.com/file/d/1664ZuOwf1PsNGcD3ti-dJvVJg_cR7kk6/view?usp=sharing

Se añadió PrestacionServicio (renombrando IPS_MEDICO) para asociar no solo IPS y Servicio sino también al Médico y guardar los atributos de horario y cuposDisponibles. Esto permite gestionar la disponibilidad de cada combinación IPS–Médico–Servicio.

Incorporamos Disponibilidad que equivale a la AGENDA con PK propio, FK a PrestacionServicio y un fechaHora más un estado, para poder listar turnos disponibles y luego reservarlos.

Modificamos Cita para que apunte a Disponibilidad en lugar de repetir IPS/Médico/Servicio y tener un solo FK, simplificando la integridad y cumpliendo el RF7 (consulta y agendamiento de horarios).

Movimos los estados de Orden y Cita a columnas dentro de OrdenDeServicio y Cita , en lugar de tablas separadas, para simplificar consultas.

3. Diseño de la base de datos para todos los requerimientos funcionales (RF1-RF7)

a. Análisis de la carga de trabajo (workload).

a. Identificación de entidades y sus atributos:

Entidad	Atributos principales
EPS	idEPS (PK), nombre, dirección, teléfono
IPS	idIPS (PK), idEPS (FK), nombre, NIT, dirección, teléfono
servicioSalud	idServicio (PK), nombreServicio, descripción, requisitos
especialidad	idEspecialidad (PK), nombre, descripción
médico	idMedico (PK), nombre, tipoDocumento, numeroDocumento, idEspecialidad
Afiliado	idAfiliado (PK), nombre, tipoDocumento, fechaNacimiento, tipoAfiliado, telefono, parentesco?, idEPS (FK)
OrdenServicio	idOrden (PK), fechaEmision, estado, idAfiliado (FK), idMedico (FK), idServicio (FK)
prestacionServicio	idPrestacion (PK), idServicio (FK), idMedico (FK), idIPS (FK), horario, cuposDisponibles
disponibilidad	idDisponibilidad (PK), idPrestacion (FK), fechaHora, estado
cita	idCita (PK), idAfiliado (FK), idDisponibilidad (FK), fechaReserva, estado
atencionMedica	idAtencion (PK), idCita (FK), receta, diagnostico, fechaAtencion, horaAtencion
enfermedad	idEnfermedad (PK), nombreEnfermedad, descripción, síntomas
afiliadoEnfermedad	idAfiliado (FK), idEnfermedad (FK), fechaDiagnostico (PK compuesta)
IPSServicio	idIPS (PK), idServicio (PK)

b. Cuantificación estimada de registros

A continuación, se presentan las cantidades estimadas de registros que tendría la base de datos para cada una de las entidades, basadas en el comportamiento proyectado del sistema de una EPS promedio en Colombia

Entidad	Cantidad estimada
EPS	5–10 (una por cada instancia del sistema)
IPS	Hasta 10.000
servicioSalud	100
especialidad	50
médico	1.000-9.000
Afiliado	900.000-10.000.000
OrdenServicio	90.000 / mes = 1.080.000 / año
prestacionServicio	20.000
disponibilidad	2.000.000 / año (citas posibles)
cita	90.000 / mes = 1.000.000 / año
atencionMedica	800.000 / año
enfermedad	500
afiliadoEnfermedad	1-3 / afiliado (2-3 millones aprox)
IPSServicio	5.000 relaciones

c. Análisis de operaciones de lectura y escritura (cómo se accede a los datos)

Este análisis describe cómo se accede a cada entidad en conjunto con otras. Se hace énfasis en el uso funcional del sistema: quién consulta qué y para qué. Esto es importante para identificar datos que se acceden juntos y que podrían beneficiarse de embebido o referencias

Entidad principal	Entidades leídas juntas	Entidades escritas juntas	Comentario

cita	disponibilidad, afiliado	cita	Se reserva una cita con base en la disponibilidad de un servicio
ordenServicio	afiliado, médico, servicioSalud	ordenServicio	Al crear una orden se consulta afiliado y médico
prestacionServicio	médico, servicio, IPS	prestacionServicio	Se consulta para mostrar horarios disponibles
disponibilidad	prestacionServicio	disponibilidad	Turnos por prestación (agendamiento)
atencionMedica	cita	atencionMedica	Cita atendida genera receta/diagnóstico
afiliado	ordenServicio, cita, enfermedades	afiliadoEnfermedad	Para mostrar historial completo del paciente

d. Cuantificación de operaciones de lectura y escritura por mes

Esta tabla cuantifica la carga de trabajo (workload) mensual por entidad. Se utilizaron las tasas diarias del enunciado para hacer la conversión a cifras mensuales, asumiendo 30 días por mes.

Para calcular las operaciones de lectura y escritura de cada entidad, nos basamos en los datos proporcionados en el enunciado. Por ejemplo, para los servicios de salud, se menciona que su creación o modificación ocurre aproximadamente una vez al mes, pero su consulta es muy frecuente, cerca de 10.000 veces al día.

En el caso de las IPS, se estima que se crean o modifican 10 veces al mes y se consultan unas 5.000 veces al día. Así, tomamos como referencia los promedios mensuales o diarios indicados y los convertimos según fuera necesario (por ejemplo, dividiendo por 30 para pasar de mes a día). Para entidades como los afiliados o las órdenes de servicio, que tienen operaciones más intensivas, utilizamos directamente los datos diarios proporcionados (como las 3.000 lecturas y escrituras por día en órdenes de servicio). Este análisis nos permitió cuantificar de forma aproximada el uso que tendrá cada entidad y justificar decisiones posteriores en el diseño de la base de datos, como la indexación y la distribución de datos.

Entidad	Operaciones de Lectura/día	Operaciones de Escritura/día
---------	----------------------------	------------------------------

EPS	50	1
IPS	500	10
servicioSalud	10.000	5
especialidad	200	1
médico	5.000	50
Afiliado	10.000	9.000
OrdenServicio	3.000	3.000
prestacionServicio	5.000	50
disponibilidad	5.000	2.000
cita	10.000	2.000
atencionMedica	5.000	2.000
enfermedad	1.000	10
afiliadoEnfermedad	2.000	500

b. Describan las colecciones de datos y las relaciones entre ellas (NoSQL) que corresponden al modelo conceptual UML propuesto.

i. La lista de entidades con la descripción de cada una de ellas

- **EPS:** Guarda idEPS (PK), nombre, dirección y teléfono.
- **IPS:** Tiene como atributos idIPS (PK), idEPS (FK), nombre, NIT, dirección, teléfono.
- **servicioSalud:** Los atributos son idServicio (PK), nombreServicio (FK), descripción, requisitos.
- **Médico:** Los atributos son idMedico (PK), nombre, tipoDocumento, numeroDocumento, idEspecialidad (FK).
- **prestacionServicio:** Los atributos son idPrestacion (PK), idServicio (FK), idMédico (FK), idIPS (FK), horario, cuposDisponibles.
- **Disponibilidad:** Los atributos son idDisponibilidad (PK), idPrestacion (FK) fechaHora, estado.
- **Afiliado:** idAfiliado (PK), idEPS (FK), nombre, tipoDocumento, numeroDocumento, fechaNacimiento, direccion, tipoAfiliado, telefono y parentesco (solo si es beneficiario).
- **ordenServicio:** idOrden (PK), idAfiliado (FK), idMedico (FK), idServicio (FK), fechaEmision, estado.
- **Cita:** idCita (PK), idAfiliado (FK), idDisponibilidad (FK), fechaReserva, estado.
- **atencionMedica:** idAtencion (PK), idCita (FK), receta, diagnostico, fechaAtencion, horaAtencion.
- **Especialidad:** idEspecialidad (PK), nombre, descripcion.

- **Enfermedad:** idEnfermedad (PK), nombreEnfermedad, descripcion, sintomas.
 - **IPSServicio:** idIPS (PK), idServicio (PK),
- ii. **Las relaciones entre entidades y su cardinalidad (uno a uno, uno a muchos, o muchos a muchos).**
1. **EPS - Afiliado**
 - **Relación:** uno a muchos
 - **Cardinalidad:** 1 EPS : N Afiliados
 - **Explicación:** Una EPS puede tener muchos afiliados; cada afiliado pertenece a una sola EPS.
 2. **EPS – IPS**
 - **Relación:** uno a muchos
 - **Cardinalidad:** 1 EPS : N IPS
 - **Explicación:** Una EPS contrata múltiples IPS; cada IPS está registrada bajo una única EPS.
 3. **IPS – servicioSalud**
 - **Relación:** muchos a muchos (N a N)
 - **Implementación:** a través de la entidad de unión ipsServicio
 - **Explicación:** Una IPS puede ofrecer varios servicios de salud y un mismo servicio puede ser provisto por varias IPS.
 4. **Especialidad – medico**
 - **Relación:** Muchos a muchos
 - **Cardinalidad:** N Especialidad : N Médicos
 - **Explicación:** Cada especialidad agrupa a múltiples médicos; cada médico puede ejercer varias especialidades.
 5. **medico – prestacionServicio**
 - **Relación:** uno a muchos
 - **Cardinalidad:** 1 Médico : N Prestaciones
 - **Explicación:** Un médico puede prestar varios servicios (prestaciones); cada prestación está asociada a un único médico.
 6. **servicioSalud – prestacionServicio**
 - **Relación:** uno a muchos
 - **Cardinalidad:** 1 Servicio : N Prestaciones
 - **Explicación:** Cada servicio de salud puede aparecer en múltiples prestaciones; cada prestación referencia un único servicio.
 7. **IPS – prestacionServicio**
 - **Relación:** uno a muchos
 - **Cardinalidad:** 1 IPS : N Prestaciones
 - **Explicación:** Una IPS puede generar varias prestaciones de servicio; cada prestación ocurre en una sola IPS.
 8. **prestacionServicio – disponibilidad**
 - **Relación:** uno a muchos
 - **Cardinalidad:** 1 Prestación : N Disponibilidades
 - **Explicación:** Cada prestación genera varios turnos (disponibilidades); cada disponibilidad corresponde a una prestación.

9. disponibilidad – cita

- **Relación:** uno a uno (0..1)
- **Cardinalidad:** 1 Disponibilidad : 0 o 1 Cita
- **Explicación:** Un turno puede quedar reservado en una única cita.

10. Afiliado – cita

- **Relación:** uno a muchos
- **Cardinalidad:** 1 Afiliado : N Citas
- **Explicación:** Un afiliado puede agendar varias citas; cada cita corresponde a un solo afiliado.

11. cita – atencionMedica

- **Relación:** uno a uno (0..1)
- **Cardinalidad:** 1 Cita : 0 o 1 Atención
- **Explicación:** Cada cita puede generar un registro de atención médica.

12. Afiliado – ordenServicio

- **Relación:** uno a muchos
- **Cardinalidad:** 1 Afiliado : N Órdenes
- **Explicación:** Un afiliado puede recibir varias órdenes de servicio; cada orden pertenece a un afiliado.

13. medico – ordenServicio

- **Relación:** uno a muchos
- **Cardinalidad:** 1 Médico : N Órdenes
- **Explicación:** Un médico puede emitir múltiples órdenes; cada orden es emitida por un solo médico.

14. servicioSalud – ordenServicio

- **Relación:** uno a muchos
- **Cardinalidad:** 1 Servicio : N Órdenes
- **Explicación:** Un servicio de salud puede figurar en varias órdenes de servicio; cada orden referencia un único servicio.

iii. El análisis de selección de esquema de asociación (modelo normalizado o embebido) para cada relación entre entidades.

- **Prestación servicio -> IPS / Médico / Servicio:** En este caso se elige el modelo referenciado debido a que se actualizan poco, pero las consultas a la colección prestacionServicio incluye datos de IPS/Médico.
- **Disponibilidad -> prestaciónServicio:** Para la relación entre Disponibilidad y prestación servicio se usa un modelo embebido debido a que los turnos se consultan y reservan juntos.
- **Cita -> Disponibilidad / Afiliado:** Se utiliza un modelo referenciado para esta relación porque las entidades se consultan y reservan por separado, por lo tanto, la colección Cita guarda solo la referencia al turno disponible

- **OrdenDeServicio -> Afiliado / Médico:** Se implementa un modelo embebido gracias a que la lista de servicios prescritos se lee junto a la orden, entonces embebemos el array de servicios.

iv. **Una descripción gráfica usando Json de cada relación entre entidades en donde presente un ejemplo de datos junto con el esquema de asociación usado (referenciado o embebido).**

- **Prestación Servicio (referenciado):**

```
{
  _id: ObjectId(""),
  idIPS: ObjectId(""), Aquí se hace la referencia a IPS
  idMedico: ObjectId(""),
  idServicio: ObjectId(""),
  horario: "08:00-12:00",
  cuposDisponibles: 20
}
```

- **Disponibilidad (embebido en PrestacionServicio):**

```
{
  _id: ObjectId(""),
  idPrestacion: ObjectId(""),
  turnos: [
    { fechaHora: ISODate("2025-06-01T08:00:00Z"), estado: "disponible" },
    { fechaHora: ISODate("2025-06-01T08:30:00Z"), estado: "reservado" }
  ]
}
```

- **OrdenDeServicio (servicios embebidos):**

```
{
  _id: ObjectId(""),
  idAfiliado: ObjectId(""),
```

```
idMedico: ObjectId(""),
fechaEmision: ISODate("2025-05-20T10:00:00Z"),
estado: "vigente",
servicios: [
  { idServicio: ObjectId(""), cantidad: 1 },
  { idServicio: ObjectId(""), cantidad: 2 }
]
}
```

- c. Cree en MongoDB las colecciones principales de su base según lo descrito antes.

Se encuentra adjunto el script de la creación de las colecciones en el archivo “Script Punto 3.C y 3.D”.

Para crear las colecciones:

```
db.createCollection("EPS");
db.createCollection("IPS");
db.createCollection("Afilado");
db.createCollection("ipsServicio");
db.createCollection("servicioSalud");
db.createCollection("atencionMedica");
db.createCollection("especialidad");
db.createCollection("medico");
db.createCollection("ordenServicio");
db.createCollection("cita");
db.createCollection("enfermedad");
db.createCollection("prestacionServicio");
db.createCollection("disponibilidad");
```

Para poblar la base de datos:

```
db.EPS.insertMany([  
  
    { idEPS: "EPS001", nombre: "Salud Total", direccion: "Calle 100  
#10-20", telefono: "3001234567" },  
  
    { idEPS: "EPS002", nombre: "Nueva EPS", direccion: "Carrera 15  
#45-78", telefono: "3017654321" }  
  
]);
```

```
db.IPS.insertMany([  
  
    { idIPS: "IPS001", idEPS: "EPS001", nombre: "Clínica Norte", NIT:  
"900123456", direccion: "Calle 80 #20-30", telefono: "3214567890" },  
  
    { idIPS: "IPS002", idEPS: "EPS002", nombre: "Hospital Central",  
NIT: "800765432", direccion: "Av. Caracas #30-50", telefono:  
"3209876543" }  
  
]);
```

```
db.servicioSalud.insertMany([  
  
    { idServicio: "SRV001", nombreServicio: "Consulta General",  
descripcion: "Consulta médica general", requisitos: "Ninguno" },  
  
    { idServicio: "SRV002", nombreServicio: "Odontología", descripcion:  
"Consulta odontológica", requisitos: "Tener cita previa" }  
  
]);
```

```
db.medico.insertMany([
```

```
    { idMedico: "MED001", nombre: "Dra. Ana Pérez", tipoDocumento:
"CC", numeroDocumento: "1234567890", idEspecialidad: "ESP001" },
```

```
    { idMedico: "MED002", nombre: "Dr. Juan Ramírez", tipoDocumento:
"CC", numeroDocumento: "9876543210", idEspecialidad: "ESP002" }
```

```
  );
```

```
db.especialidad.insertMany([
```

```
    { idEspecialidad: "ESP001", nombre: "Medicina General",
descripcion: "Atención primaria" },
```

```
    { idEspecialidad: "ESP002", nombre: "Odontología", descripcion:
"Salud bucal" }
```

```
]);
```

```
db.prestacionServicio.insertMany([
```

```
    { idPrestacion: "PST001", idServicio: "SRV001", idMédico:
"MED001", idIPS: "IPS001", horario: "08:00-12:00", cuposDisponibles:
10 },
```

```
    { idPrestacion: "PST002", idServicio: "SRV002", idMédico:
"MED002", idIPS: "IPS002", horario: "13:00-17:00", cuposDisponibles:
8 }
```

```
]);
```

```
db.disponibilidad.insertMany([
```

```
    { idDisponibilidad: "DISP001", idPrestacion: "PST001", fechaHora:
ISODate("2025-06-01T08:00:00Z"), estado: "disponible" },
```

```
    { idDisponibilidad: "DISP002", idPrestacion: "PST002", fechaHora:
ISODate("2025-06-01T13:00:00Z"), estado: "reservado" }
```

```
]);
```

```
db.Afiliado.insertMany([
```

```
    { idAfilado: "AF001", idEPS: "EPS001", nombre: "Carlos López",  
      tipoDocumento: "CC", numeroDocumento: "1012345678",  
      fechaNacimiento: ISODate("1990-01-01"), direccion: "Calle 50 #10-20",  
      tipoAfilado: "Cotizante", telefono: "3134567890" },
```

```
    { idAfilado: "AF002", idEPS: "EPS002", nombre: "Luisa Fernández",  
      tipoDocumento: "CC", numeroDocumento: "1023456789",  
      fechaNacimiento: ISODate("2000-05-15"), direccion: "Carrera 40  
#60-10", tipoAfilado: "Beneficiario", telefono: "3149876543",  
      parentesco: "Hija" }
```

```
  );
```

```
db.ordenServicio.insertMany([
```

```
  { idOrden: "ORD001", idAfilado: "AF001", idMedico: "MED001",  
    idServicio: "SRV001", fechaEmision:  
    ISODate("2025-05-20T10:00:00Z"), estado: "vigente" },
```

```
  { idOrden: "ORD002", idAfilado: "AF002", idMedico: "MED002",  
    idServicio: "SRV002", fechaEmision:  
    ISODate("2025-05-22T11:30:00Z"), estado: "vigente" }
```

```
]);
```

```
db.cita.insertMany([
```

```
  { idCita: "CIT001", idAfilado: "AF001", idDisponibilidad:  
    "DISP001", fechaReserva: ISODate("2025-05-25T09:00:00Z"), estado:  
    "agendada" },
```

```
  { idCita: "CIT002", idAfilado: "AF002", idDisponibilidad:  
    "DISP002", fechaReserva: ISODate("2025-05-26T14:00:00Z"), estado:  
    "confirmada" }
```

```
]);
```

```
db.atencionMedica.insertMany([
```

```
  { idAtencion: "AT001", idCita: "CIT001", receta: "Paracetamol  
500mg", diagnostico: "Gripe leve", fechaAtencion:  
    ISODate("2025-06-01T08:00:00Z"), horaAtencion: "08:05" },
```

```
    { idAtencion: "AT002", idCita: "CIT002", receta: "Lidocaína",  
      diagnostico: "Dolor dental", fechaAtencion:  
      ISODate("2025-06-01T13:05:00Z"), horaAtencion: "13:10" }
```

```
  );
```

```
db.enfermedad.insertMany([
```

```
  { idEnfermedad: "ENF001", nombreEnfermedad: "Gripe",  
    descripcion: "Infección viral leve", sintomas: "Fiebre, tos, malestar" },
```

```
  { idEnfermedad: "ENF002", nombreEnfermedad: "Caries",  
    descripcion: "Deterioro dental", sintomas: "Dolor de muelas,  
    sensibilidad" }
```

```
]);
```

```
db.ipsServicio.insertMany([
```

```
  { idIPS: "IPS001", idServicio: "SRV001" },
```

```
  { idIPS: "IPS002", idServicio: "SRV002" }
```

```
]);
```

d. Cree los esquemas de validación para cada colección.

Se encuentra adjunto el script de la creación de los esquemas de validación en el archivo “Script Punto 3.C y 3.D”.

```
db.runCommand({
```

```
  collMod: "EPS",
```

```
  validator: {
```

```
    $jsonSchema: {
```

```
      bsonType: "object",
```

```
      required: ["idEPS", "nombre", "direccion", "telefono"],
```

```
properties: {  
    idEPS: { bsonType: "string" },  
    nombre: { bsonType: "string" },  
    direccion: { bsonType: "string" },  
    telefono: { bsonType: "string" }  
}  
}  
}  
});
```

```
db.runCommand({  
    collMod: "IPS",  
    validator: {  
        $jsonSchema: {  
            bsonType: "object",  
            required: ["idIPS", "idEPS", "nombre", "NIT", "direccion", "telefono"],  
            properties: {  
                idIPS: { bsonType: "string" },  
                idEPS: { bsonType: "string" },  
                nombre: { bsonType: "string" },  
                NIT: { bsonType: "string" },  
                direccion: { bsonType: "string" },  
                telefono: { bsonType: "string" }  
            }  
        }  
    }  
})
```

```
});
```

```
db.runCommand({  
  collMod: "servicioSalud",  
  validator: {  
    $jsonSchema: {  
      bsonType: "object",  
      required: ["idServicio", "nombreServicio", "descripcion", "requisitos"],  
      properties: {  
        idServicio: { bsonType: "string" },  
        nombreServicio: { bsonType: "string" },  
        descripcion: { bsonType: "string" },  
        requisitos: { bsonType: "string" }  
      }  
    }  
  }  
});
```

```
db.runCommand({  
  collMod: "medico",  
  validator: {  
    $jsonSchema: {  
      bsonType: "object",  
      required: ["idMedico", "nombre", "tipoDocumento", "numeroDocumento",  
"idEspecialidad"],  
      properties: {
```



```
        idMedico: { bsonType: "string" },
        nombre: { bsonType: "string" },
        tipoDocumento: { bsonType: "string" },
        numeroDocumento: { bsonType: "string" },
        idEspecialidad: { bsonType: "string" }
    }
}
});
```

```
db.runCommand({
    collMod: "especialidad",
    validator: {
        $jsonSchema: {
            bsonType: "object",
            required: ["idEspecialidad", "nombre", "descripcion"],
            properties: {
                idEspecialidad: { bsonType: "string" },
                nombre: { bsonType: "string" },
                descripcion: { bsonType: "string" }
            }
        }
    }
});
```

```
db.runCommand({
```

```
collMod: "prestacionServicio",

validator: {

  $jsonSchema: {

    bsonType: "object",

    required: ["idPrestacion", "idServicio", "idMédico", "idIPS", "horario",
"cuposDisponibles"],

    properties: {

      idPrestacion: { bsonType: "string" },

      idServicio: { bsonType: "string" },

      idMédico: { bsonType: "string" },

      idIPS: { bsonType: "string" },

      horario: { bsonType: "string" },

      cuposDisponibles: { bsonType: "int", minimum: 0 }

    }

  }

}

});
```

```
db.runCommand({

collMod: "disponibilidad",

validator: {

  $jsonSchema: {

    bsonType: "object",

    required: ["idDisponibilidad", "idPrestacion", "fechaHora", "estado"],

    properties: {

      idDisponibilidad: { bsonType: "string" },

      idPrestacion: { bsonType: "string" },
```

```
        fechaHora: { bsonType: "date" },
        estado: { bsonType: "string" }
    }
}
}
});
```

```
db.runCommand({
    collMod: "Afiliado",
    validator: {
        $jsonSchema: {
            bsonType: "object",
            required: ["idAfiliado", "idEPS", "nombre", "tipoDocumento",
"numeroDocumento", "fechaNacimiento", "direccion", "tipoAfiliado",
"telefono"],
            properties: {
                idAfiliado: { bsonType: "string" },
                idEPS: { bsonType: "string" },
                nombre: { bsonType: "string" },
                tipoDocumento: { bsonType: "string" },
                numeroDocumento: { bsonType: "string" },
                fechaNacimiento: { bsonType: "date" },
                direccion: { bsonType: "string" },
                tipoAfiliado: { bsonType: "string" },
                telefono: { bsonType: "string" },
                parentesco: { bsonType: ["string", "null"] }
            }
        }
    }
});
```

```
    }  
  }  
});
```

```
db.runCommand({  
  collMod: "ordenServicio",  
  validator: {  
    $jsonSchema: {  
      bsonType: "object",  
      required: ["idOrden", "idAfiliado", "idMedico", "idServicio",  
"fechaEmision", "estado"],  
      properties: {  
        idOrden: { bsonType: "string" },  
        idAfiliado: { bsonType: "string" },  
        idMedico: { bsonType: "string" },  
        idServicio: { bsonType: "string" },  
        fechaEmision: { bsonType: "date" },  
        estado: { bsonType: "string" }  
      }  
    }  
  }  
});
```

```
db.runCommand({  
  collMod: "cita",  
  validator: {  
    $jsonSchema: {
```

```
    bsonType: "object",

    required: ["idCita", "idAfiliado", "idDisponibilidad", "fechaReserva",
"estado"],

    properties: {

        idCita: { bsonType: "string" },

        idAfiliado: { bsonType: "string" },

        idDisponibilidad: { bsonType: "string" },

        fechaReserva: { bsonType: "date" },

        estado: { bsonType: "string" }

    }

}

}

});
```

```
db.runCommand({

    collMod: "atencionMedica",

    validator: {

        $jsonSchema: {

            bsonType: "object",

            required: ["idAtencion", "idCita", "receta", "diagnostico",
"fechaAtencion", "horaAtencion"],

            properties: {

                idAtencion: { bsonType: "string" },

                idCita: { bsonType: "string" },

                receta: { bsonType: "string" },

                diagnostico: { bsonType: "string" },

                fechaAtencion: { bsonType: "date" },
```

```
        horaAtencion: { bsonType: "string" }  
    }  
}  
}  
});
```

```
db.runCommand({  
    collMod: "enfermedad",  
    validator: {  
        $jsonSchema: {  
            bsonType: "object",  
            required: ["idEnfermedad", "nombreEnfermedad", "descripcion",  
"sintomas"],  
            properties: {  
                idEnfermedad: { bsonType: "string" },  
                nombreEnfermedad: { bsonType: "string" },  
                descripcion: { bsonType: "string" },  
                sintomas: { bsonType: "string" }  
            }  
        }  
    }  
});
```

```
db.runCommand({  
    collMod: "ipsServicio",  
    validator: {  
        $jsonSchema: {
```

```

        bsonType: "object",

        required: ["idIPS", "idServicio"],

        properties: {

            idIPS: { bsonType: "string" },

            idServicio: { bsonType: "string" }

        }

    }

}

});

```

Explicacion Requerimientos Funcionales y de Consulta

RF1 – Registrar IPS

- **Código Java**
 - **Modelo:** `Ips` con campos `id` (`ObjectId`), `nit`, `nombre`, `direccion`, `telefono`.
 - **Repositorio:** `IpsRepository` extends `MongoRepository<Ips, ObjectId>`.
 - **Servicio:** `IpsService.create(Ips)` valida unicidad de `nit` y persiste.
 - **Controlador:** `IpsController@PostMapping("/ips")` expone `/ips`.
- **Postman**
 - **Request:** `POST {{base_url}}/ips`

Body (raw JSON):

```

json
CopiarEditar
{

    "nit": "902104974",

    "nombre": "IPS Salud Total",

    "direccion": "Av 26 #85-10",

    "telefono": "6017639514"

}

```

-
- **Pre-request:** genera valores aleatorios para **nit** y **telefono**.
- **Test:** verifica **201 Created** y guarda `{{ $response.id }}` en variable **ipsId**.

RF2 – Registrar Servicio de Salud

- **Código Java**

- **Modelo:** **ServicioDeSalud** con campos **id**, **codigo**, **nombre**, **descripcion**, **requiereOrden**.
- **Repositorio:** **ServicioDeSaludRepository** extends **MongoRepository<ServicioDeSalud, ObjectId>**.
- **Servicio:** **ServicioDeSaludService.create(...)** persiste validando campos **@NotBlank**.
- **Controlador:** **ServicioDeSaludController@PostMapping("/servicios")**.

- **Postman**

- **Request:** **POST** `{{base_url}}/servicios`

Body:

```

json
CopiarEditar
{
    "codigo": "SERV01",
    "nombre": "Consulta General",
    "descripcion": "Cita con médico general",
    "requiereOrden": false
}

```

- - **Test:** guarda **servicioId**.
-

RF3 – Asignar Servicio a una IPS

- Código Java

- Controlador:
`IpsController@PostMapping("/ips/{ipsId}/servicios/{servicioId}")`
). Llama a `IpsService.asignarServicio(ipsId, servicioId)` que añade la referencia al documento IPS.

- Postman

- Request: `POST {{base_url}}/ips/{{ipsId}}/servicios/{{servicioId}}`
 - Test: verifica 200 OK.
-

RF4 – Registrar Médico

- Código Java

- Modelo: `Medico` con campos `id`, `tipoDocumento`, `numeroDocumento`, `nombre`, `especialidad`, lista de `ipsId`, lista de `servicioId`.
- Repositorio: `MedicoRepository` extends `MongoRepository<Medico, ObjectId>`.
- Servicio: `MedicoService.create(Medico)` inserta validando campos.
- Controlador: `MedicoController@PostMapping("/medicos")`.

- Postman

- Request: `POST {{base_url}}/medicos`

Body usa las variables:

```
json
CopiarEditar
{
    "tipoDocumento": "CC",
    "numeroDocumento": "1020304050",
    "nombre": "Dr. María Pérez",
    "especialidad": "CARDIOLOGÍA",
```

```
    "ips": [{"{{ipsId}}"],  
    "servicios": [{"{{servicioId}}"]  
}
```

- - Pre-request: genera **medicoId** aleatorio.
 - Test: guarda **medicoId**.
-

RF5 – Registrar Afiliado

- Código Java
 - Modelo: **Afiliado** con campos **id**, **tipoDocumento**, **numeroDocumento**, **nombre**, **fechaNacimiento**, **direccion**, **telefono**, **cotizante** (boolean), y si es beneficiario, **idCotizante** y **parentesco**.
 - Repositorio/Servicio/Controlador análogos a Medico.
- Postman
 - Request: POST **{{base_url}}/afiliados**

Body:

```
json  
CopiarEditar  
{  
  
    "tipoDocumento": "CC",  
  
    "numeroDocumento": "2030405060",  
  
    "nombre": "Juan Contratista",  
  
    "fechaNacimiento": "1980-05-10",  
  
    "direccion": "Calle 10 #20-30",  
  
    "telefono": "3151234567",  
  
    "cotizante": true  
}
```

-
- **Test: guarda afiliadoId.**

RF6 – Registrar Orden de Servicio

- **Código Java**

- **Modelo: OrdenServicio** con **id**, **idMedico**, **idAfiliado**, lista **idServicios**, fecha creación automática.
- **Repositorio: OrdenServicioRepository.**
- **Servicio: OrdenServicioService.create(...).**
- **Controlador: OrdenServicioController@PostMapping("/ordenes").**

- **Postman**

- **Request: POST {{base_url}}/ordenes**

Body:

```
json
CopiarEditar
{
    "idMedico": "{{medicoId}}",
    "idAfiliado": "{{afiliadoId}}",
    "idServicios": ["{{servicioId}}"]
}
```

-
- **Test: guarda ordenId.**

RF7 – Agenda y Reserva de Cita

1. Crear Disponibilidad

- **Controlador: AgendaController@PostMapping("/agenda/disponibilidad")** recibe

`{servicioId, fechaHora, ipsId, medicoId}` y guarda en colección `disponibilidades`.

- Postman:

- **POST** `{{base_url}}/agenda/disponibilidad`
- **Body** con las IDs pre guardadas y una fecha dentro de las 4 semanas.
- **Guarda** `dispId`.

2. Consultar Disponibilidad (siguientes 4 semanas)

- Controlador:

`AgendaController@GetMapping("/agenda/disponibilidad")?servicioId=...` filtra por `servicioId` y rangos de fecha (`now` a `now+4w`).

- Postman: GET

`{{base_url}}/agenda/disponibilidad?servicioId={{servicioId}}` → guarda primer `dispId` del array.

3. Reservar Cita

- Controlador:

`AgendaController@PostMapping("/agenda/reservar")?disponibilidadId=...` cambia `disp.disponible=false` y crea en colección `reservas`.

- Postman: POST

`{{base_url}}/agenda/reservar?disponibilidadId={{dispId}}`.

RFC1 – Estadísticas de Disponibilidad

- Endpoint Java

- `StatsController@GetMapping("/stats/disponibilidad")?codigo=...`

- Llama a `StatsService.getDisponibilidadPorCodigo(codigo)` → `StatsRepositoryCustom.disponibilidadPorCodigo(codigo)` que hace un *pipeline* de agregación MongoDB uniendo colecciones `servicios_salud`, `disponibilidades`, `ips` y `medicos`, y proyectando los campos deseados.

- DTO: `DisponibilidadStats(servicioNombre, fechaHora, ipsNombre, medicoNombre)`.

- Postman

- Request: GET
`{{base_url}}/stats/disponibilidad?codigo={{codigoServ}}`
 - Pre-request: define `codigoServ`.
 - Test: comprueba que el array devuelto contenga objetos con las cuatro propiedades.
-

RFC2 – Top 20 Servicios Más Solicitados

- Endpoint Java

- `StatsController@GetMapping("/stats/servicios-mas-solicitados")?desde=&hasta=`
- `StatsService.getTopServicios(desde, hasta) → StatsRepositoryCustom.topServiciosMásSolicitados(...)` con un *pipeline* que:
 1. Filtra en `ordenes_servicio` por `fechaCreacion` entre `desde` y `hasta`.
 2. Descompone la lista `idServicios`.
 3. Agrupa por servicio, cuenta emisiones, ordena desc, limita a 20.
 4. Hace *lookup* a `servicios_salud` para obtener el nombre.
 5. Proyecta `servicioId, servicioNombre, total`.
- DTO: `ServicioSolicitadoStats(servicioId, servicioNombre, total)`.

- Postman

- Request: GET
`{{base_url}}/stats/servicios-mas-solicitados?desde={{desde}}&hasta={{hasta}}`
- Pre-request: define fechas `desde` (p.ej. primer día del mes) y `hasta` (hoy).
- Test: verifica status **200** y que el array tenga ≤ 20 elementos con las tres propiedades.

RFC1 – Estadísticas de Disponibilidad

- **Endpoint Java**

- `StatsController@GetMapping("/stats/disponibilidad")?codigo=...`
- Llama a `StatsService.getDisponibilidadPorCodigo(codigo)` → `StatsRepositoryCustom.disponibilidadPorCodigo(codigo)` que hace un *pipeline* de agregación MongoDB uniendo colecciones `servicios_salud`, `disponibilidades`, `ips` y `medicos`, y proyectando los campos deseados.
- DTO: `DisponibilidadStats(servicioNombre, fechaHora, ipsNombre, medicoNombre)`.

- **Postman**

- Request: GET
`{{base_url}}/stats/disponibilidad?codigo={{codigoServ}}`
 - Pre-request: define `codigoServ`.
 - Test: comprueba que el array devuelto contenga objetos con las cuatro propiedades.
-

RFC2 – Top 20 Servicios Más Solicitados

- **Endpoint Java**

- `StatsController@GetMapping("/stats/servicios-mas-solicitados")?desde=&hasta=`
- `StatsService.getTopServicios(desde, hasta)` → `StatsRepositoryCustom.topServiciosMásSolicitados(...)` con un *pipeline* que:
 1. Filtra en `ordenes_servicio` por `fechaCreacion` entre `desde` y `hasta`.
 2. Descompone la lista `idServicios`.
 3. Agrupa por servicio, cuenta emisiones, ordena desc, limita a 20.
 4. Hace *lookup* a `servicios_salud` para obtener el nombre.
 5. Proyecta `servicioId`, `servicioNombre`, `total`.
- DTO: `ServicioSolicitadoStats(servicioId, servicioNombre, total)`.

- **Postman**

- **Request: GET**
{{base_url}}/stats/servicios-mas-solicitados?desde={{desde}}&hasta={{hasta}}
- **Pre-request:** define fechas **desde** (p.ej. primer día del mes) y **hasta** (hoy).
- **Test:** verifica status **200** y que el array tenga ≤ 20 elementos con las tres propiedades.