Nombre: Shaiel Mateo Jimenez Posada
Código: 202323846
Asignatura: Sistemas Transaccionales

Documentación de Requerimientos Funcionales y de Consulta

Requerimientos Funcionales (RF1 – RF9)

RF1: Registrar IPS

```
@Modifying
    @Transactional
    @Query(value = "INSERT INTO ips (NIT, nombre, tipo, direccion, telefono) VALUES (:nit, :nombre, :tipo,
:direccion, :telefono)", nativeQuery = true)
    void insertarIps(@Param("nit") Integer nit, @Param("nombre") String nombre, @Param("tipo") String tipo,
@Param("direccion") String direccion, @Param("telefono") Integer telefono);
```

Repository: IpsRepository
Permite registrar una IPS contratada por la EPS incluyendo su NIT, nombre, dirección, teléfono y tipo.

RF2: Registrar un Servicio de Salud

```
    @Modifying
    @Transactional
    @Query(value = "INSERT INTO servicio_salud (idservicio, tiposervicio, descripcion, servicio_salud_id)
VALUES (:idservicio, :tiposervicio, :descripcion, :servicio_salud_id)", nativeQuery = true)
    void insertarServicio(@Param("idservicio") Integer idservicio, @Param("tiposervicio") String tiposervicio,
@Param("descripcion") String descripcion, @Param("servicio_salud_id") Integer servicio_salud_id);
```

Repository: ServicioSaludRepository
Permite registrar un nuevo tipo de servicio de salud que la EPS puede ofrecer a los afiliados.

RF3: Asignar un Servicio de Salud a una IPS

```
    @Modifying
    @Transactional
    @Query(value = "INSERT INTO directorio_servicio (ips_nit, serv_salud_serv_salud_id) VALUES (:id_ips,
:id_serviciosalud)", nativeQuery = true)
    void insertarDirectorioServicio(@Param("id_ips") Integer id_ips, @Param("id_serviciosalud") Integer
id_serviciosalud);
```

Repository: DirectorioServicioRepository
Relaciona servicios de salud previamente registrados con una IPS existente para que los pueda prestar.

RF4: Registrar Médico

```
    @Modifying
    @Transactional
    @Query(value = "INSERT INTO medico (numregistromedico, numerodocumento, tipodocumento,
especialidad, nombre) VALUES (:numRegistroMedico, :numeroDocumento, :tipoDocumento, :especialidad,
:nombre)", nativeQuery = true)
    void insertarMedico(@Param("numRegistroMedico") Integer numRegistroMedico,
@Param("numeroDocumento") Integer numeroDocumento, @Param("tipoDocumento") String
tipoDocumento, @Param("especialidad") String especialidad, @Param("nombre") String nombre);
```

Repository: MedicoRepository
Registra la información de médicos que pueden prestar servicios de salud en una o varias IPS.

RF5: Registrar Afiliado

```
    @Modifying
    @Transactional
    @Query(value="INSERT INTO usuario (tipodocumento, numdocumento, nombre, fechanacimiento,
telefono, tipocontribucion, usuario_id) VALUES (:tipodocumento, :numdocumento, :nombre,
:fechanacimiento, :telefono, :tipocontribucion, :usuarioid)", nativeQuery = true)
    void insertarUsuario(@Param("tipodocumento") String tipodocumento, @Param("numdocumento")
Integer numdocumento, @Param("nombre") String nombre, @Param("fechanacimiento") LocalDate
```

```
fechanacimiento, @Param("telefono") Integer telefono, @Param("tipocontribucion") String tipocontribucion,
@Param("usuarioid") Integer usuarioid);
```

Repository: UsuarioRepository
Registra la información de un afiliado, incluyendo tipo (contribuyente o beneficiario) y parentesco si aplica.

RF6: Registrar una Orden de Servicio de Salud

```
    @Modifying
    @Transactional
    @Query(value = "INSERT INTO orden (idorden, medico_numregistromedico, usuario_usuario_id,
fechaemision, estado) VALUES (:idorden, :numregistromedico, :usuarioid, :fechaemision, :estado)",
nativeQuery = true)
    void insertarOrden(@Param("idorden") Integer idorden, @Param("numregistromedico") Integer
numregistromedico, @Param("usuarioid") Integer usuarioid, @Param("fechaemision") LocalDate
fechaemision, @Param("estado") String estado);
```

Repository: OrdenRepository
Permite que un médico prescriba uno o más servicios de salud a un afiliado, asociando médico, afiliado y servicios.

RF7: Agendar un Servicio de Salud por parte del Afiliado

```
    @Modifying
    @Transactional
    @Query(value = " UPDATE agenda SET fecha = :fecha, disponibilidad = :disponibilidad,
directorio_servicio_nit = :directorio_servicio_nit, directorio_servicio_serv_id = :directorio_servicio_serv_id,
directorio_medico_nit = :directorio_medico_nit, directorio_medico_regmed = :directorio_medico_regmed,
servicio_medico_regmed = :servicio_medico_regmed, servicio_medico_serv_id = :servicio_medico_serv_id
WHERE id = :id AND disponibilidad = 'Disponible' ", nativeQuery = true)
    void actualizarAgendaPorDisponibilidad(@Param("fecha") LocalDate fecha, @Param("disponibilidad")
String disponibilidad, @Param("directorio_servicio_nit") Integer directorio_servicio_nit,
@Param("directorio_servicio_serv_id") Integer directorio_servicio_serv_id, @Param("directorio_medico_nit")
Integer directorio_medico_nit, @Param("directorio_medico_regmed") Integer directorio_medico_regmed,
@Param("servicio_medico_regmed") Integer servicio_medico_regmed, @Param("servicio_medico_serv_id")
Integer servicio_medico_serv_id, @Param("id") Integer id);
```

Repository: AgendaRepository
Permite consultar disponibilidad y reservar una cita según una orden médica válida (si aplica).

RF8: Registrar la Prestación de un Servicio

```
@Modifying
   @Transactional
   @Query(value="INSERT INTO HISTORIAL_CITAS (IDCITA, IPS_NIT) VALUES (:id_cita, :nit_ips)",
nativeQuery = true)
   void insertarHistorialCita(@Param("id_cita")Integer id_cita,@Param("nit_ips") Integer nit_ips);
```

Repository: CitaRepository
Registra que un servicio fue efectivamente prestado a un afiliado en una fecha y hora específicas.

RF9: Agendar un Servicio de Salud por parte del Afiliado (Transaccional)

```
   @Modifying
   @Transactional
   @Query(value = " UPDATE agenda SET fecha = :fecha, disponibilidad = :disponibilidad,
directorio_servicio_nit = :directorio_servicio_nit, directorio_servicio_serv_id = :directorio_servicio_serv_id,
directorio_medico_nit = :directorio_medico_nit, directorio_medico_regmed = :directorio_medico_regmed,
servicio_medico_regmed = :servicio_medico_regmed, servicio_medico_serv_id = :servicio_medico_serv_id
WHERE id = :id AND disponibilidad = 'Disponible' ", nativeQuery = true)
   void actualizarAgendaPorDisponibilidad(@Param("fecha") LocalDate fecha, @Param("disponibilidad")
String disponibilidad, @Param("directorio_servicio_nit") Integer directorio_servicio_nit,
@Param("directorio_servicio_serv_id") Integer directorio_servicio_serv_id, @Param("directorio_medico_nit")
Integer directorio_medico_nit, @Param("directorio_medico_regmed") Integer directorio_medico_regmed,
@Param("servicio_medico_regmed") Integer servicio_medico_regmed, @Param("servicio_medico_serv_id")
Integer servicio_medico_serv_id, @Param("id") Integer id);
```

Repository: AgendaRepository
Registra el agendamiento de un servicio a un afiliado en una fecha y hora específicas.

Requerimientos Funcionales de Consulta (RFC1 - RFC5)

RFC1: Consultar la agenda de disponibilidad en las siguientes 4 semanas

```
@Query(value = "SELECT * FROM agenda WHERE disponibilidad = 'Disponible' AND
directorio_servicio_serv_id = :idServicio AND fecha BETWEEN :startDate AND :endDate", nativeQuery =
true)
    Collection<Agenda> darAgendasPorRangoDeFechasYServicio(@Param("idServicio") Integer idServicio,
@Param("startDate") LocalDate startDate, @Param("endDate") LocalDate endDate);
```

Repository: AgendaRepository
Devuelve los horarios disponibles de un servicio de salud durante las próximas 4 semanas.

RFC2: Mostrar los 20 servicios más solicitados

```
@Query(value="SELECT SERVICIO_SALUD.tipoServicio, count(DIRECTORIO_SERVICIO.idServicio) as
veces_agendado " +
          "FROM DIRECTORIO_SERVICIO"+
          "INNER JOIN SERVICIO_SALUD ON DIRECTORIO_SERVICIO.ID_SERVICIOSALUD =
SERVICIO_SALUD.ID " +
          "INNER JOIN CITA ON DIRECTORIO_SERVICIO.idCita=CITA.id " +
          "WHERE CITA.fecha BETWEEN :fechaInicial AND :fechaFinal  " +
          "GROUP BY SERVICIO_SALUD.tipoServicio, SERVICIO_SALUD.ID ORDER BY  veces_agendado
DESC " +
          "FETCH FIRST 20 ROWS ONLY"
          , nativeQuery = true)
    Collection<RespuestaDar20ServiciosSalud> dar20ServiciosSalud(@Param ("fechaInicial") String
fechaInicial, @Param ("fechaFinal") String fechaFinal);
```

Repository: DirectorioServicioRepository
Devuelve los 20 servicios que más veces han sido agendados en un periodo de tiempo.

RFC3: Mostrar el índice de uso de cada servicio

```
@Query(value="SELECT Servicio_Orden.idServicio," +
          "ROUND( COUNT(CITA.idServicio) /SELECT COUNT(*) FROM CITA WHERE fecha BETWEEN
:fechaIn AND :fechaOut) , 2) AS indice_uso" +
```

```
        "FROM Servicio_Orden " +
        "INNER JOIN CITA ON CITA.idServicio = Servicio_Orden.idServicio " +
        "WHERE CITA.fecha BETWEEN :fechaIn AND :fechaOut" +
        "GROUP BY Servicio_Orden.idServicio" +
        "ORDER BY indice_uso DESC",
        nativeQuery = true)
  Collection<RespuestaMostrarIndiceUsoServicios> mostrarIndiceUsoServicios(@Param("fechaIn") String
fechaIn, @Param("fechaOut") String fechaOut);
```

Repository: DirectorioServicioRepository
Calcula un índice de uso para cada servicio comparando servicios disponibles y usados.

RFC4: Mostrar utilización de servicios por un afiliado

```
@Query(value = "SELECT SERVICIO_SALUD.tipoServicio," +
        "CITA.fecha,MEDICO.nombre, IPS.nombre " +
        "FROM CITA " +
        "INNER JOIN DIRECTORIO_SERVICIO ON DIRECTORIO_SERVICIO.id_cita = CITA.idCita" +
        "INNER JOIN SERVICIO_SALUD ON SERVICIO_SALUD.idServicio =
DIRECTORIO_SERVICIO.id_servicio_salud" +
        "INNER JOIN DIRECTORIO_MEDICO ON DIRECTORIO_MEDICO.nit_ips =
DIRECTORIO_SERVICIO.id_ips" +
        "INNER JOIN MEDICO ON MEDICO.numRegistroMedico =
DIRECTORIO_MEDICO._num_registro_medico" +
        "INNER JOIN IPS ON IPS.id = CITA.ips" +
        "INNER JOIN USUARIO ON USUARIO.numDocumento = CITA.numDocumento " +
        "WHERE USUARIO.numDocumento= :numDocumento AND CITA.fecha BETWEEN :fechaInicio
AND :fechaFinal;"
        , nativeQuery = true)
  Collection<Respuesta> darCitasPorFecha(@Param("fechaInicio") Date fechaInicio,
@Param("fechaFinal") Date fechaFinal,@Param("numDocumento") Long numDocumento);
```

Repository: CitaRepository
Lista los servicios usados por un afiliado en un rango de fechas.

RFC5: Consultar agenda de disponibilidad (Serializable)

```
@Query(value = "SELECT s.tiposervicio AS nombreServicio, " +
                "a.fecha AS fechaDisponibilidad, " +
                "m.nombre AS nombreMedico " +
                "FROM agenda a " +
                "JOIN servicios_medico sm ON a.servicio_medico_serv_id = sm.servicio_servicio_id " +
                "JOIN medico m ON sm.medico_numregistromedico = m.numregistromedico " +
                "JOIN servicio_salud s ON sm.servicio_servicio_id = s.idservicio " +
                "WHERE disponibilidad = 'Disponible' AND (:idServicio IS NULL OR s.idservicio = :idServicio) "
+
                "AND (:idMedico IS NULL OR m.numregistromedico = :idMedico) " +
                "AND a.fecha BETWEEN :startDate AND :endDate " +
                "ORDER BY a.fecha ASC FOR UPDATE",
        nativeQuery = true)

   Collection<RespuestaDisponibilidadServicio> consultarDisponibilidadSerializable(
      @Param("idServicio") Integer idServicio,
      @Param("startDate") LocalDate startDate,
      @Param("endDate") LocalDate endDate,
      @Param("idMedico") Integer idMedico
   );
```

Repository: AgendaRepository
Consulta con nivel SERIALIZABLE, bloqueando filas y simulando concurrencia con delay de 30 segundos.

RFC6: Consultar agenda de disponibilidad (Read Committed)

```
@Query(value = "SELECT s.tiposervicio AS nombreServicio, " +
                "a.fecha AS fechaDisponibilidad, " +
                "m.nombre AS nombreMedico " +
                "FROM agenda a " +
                "JOIN servicios_medico sm ON a.servicio_medico_serv_id = sm.servicio_servicio_id " +
                "JOIN medico m ON sm.medico_numregistromedico = m.numregistromedico " +
                "JOIN servicio_salud s ON sm.servicio_servicio_id = s.idservicio " +
                "WHERE disponibilidad = 'Disponible' AND (:idServicio IS NULL OR s.idservicio = :idServicio) "
+
```

```
            "AND (:idMedico IS NULL OR m.numregistromedico = :idMedico) " +
            "AND a.fecha BETWEEN :startDate AND :endDate " +
            "ORDER BY a.fecha ASC",
        nativeQuery = true)

Collection<RespuestaDisponibilidadServicio> consultarDisponibilidadReadCommitted(
    @Param("idServicio") Integer idServicio,
    @Param("startDate") LocalDate startDate,
    @Param("endDate") LocalDate endDate,
    @Param("idMedico") Integer idMedico
);
```

Repository: AgendaRepository
Consulta con el nivel READ COMMITTED, leyendo filas y simulando concurrencia con un delay de 30 segundos.

Escenarios de Prueba de Concurrencia

Escenario de Prueba de Concurrencia 1

Se ejecuta RFC5 con nivel de aislamiento SERIALIZABLE, luego se ejecuta RF9 durante los 30 segundos de espera. RF9 queda bloqueado hasta que RFC5 libera el recurso, demostrando que los datos leídos no son alterados durante la transacción.
El componente que implementa RF9 quedó bloqueado esperando que finalizara la transacción de RFC5. Esto se debe a que ambas transacciones acceden a la misma fila en la tabla agenda usando SELECT ... FOR UPDATE. La base de datos, cumpliendo con las reglas del aislamiento SERIALIZABLE, impide que RF9 acceda a los datos hasta que RFC5 finalice su ejecución y libere los bloqueos.

Este comportamiento garantiza que no ocurran lecturas no repetibles, sucias o fantasmas, y que ambas transacciones se ejecuten como si fueran en serie.

El resultado devuelto por RFC5 corresponde al estado de la agenda al momento de iniciar su transacción, antes de que RF9 intentara agendar el servicio.

Dado que la transacción de RFC5 mantiene un snapshot aislado del estado de los datos y bloquea las filas leídas, no ve ninguna modificación realizada por RF9. Por lo tanto la orden de servicio ingresada por RF9 no aparece en los resultados de RFC5, mientras que el RFC5 retorna un conjunto de datos “congelado”, consistente, que no incluye ninguna agenda que haya sido modificada por RF9, ya que RF9 ni siquiera había podido ejecutarse completamente en ese momento.

Tiempo	Acción
0s	El usuario A ejecuta RFC5. La consulta se ejecuta bajo aislamiento SERIALIZABLE e incluye una pausa artificial de 30 segundos (Thread.sleep) para simular procesamiento prolongado. La consulta usa SELECT ... FOR UPDATE, lo que implica que bloquea las filas leídas en la tabla agenda.
10s	Durante la pausa, el usuario B intenta ejecutar RF9, que intenta reservar una agenda específica previamente consultada. Esta transacción también tiene aislamiento SERIALIZABLE y accede (con FOR UPDATE) a la misma fila de agenda bloqueada por RFC5.
10s–30s	Como RF9 intenta acceder a una fila bloqueada por RFC5, la base de datos lo bloquea y lo deja en espera hasta que RFC5 termine y libere el candado.
30s	RFC5 finaliza su consulta, libera el candado sobre la fila de agenda.
31s	RF9 retoma su ejecución, accede a la agenda, verifica la disponibilidad y agenda el servicio si aún está disponible.

Escenario de Prueba de Concurrencia 2

Se ejecuta RFC6 con nivel de aislamiento READ COMMITTED, luego se ejecuta RF9 durante los 30 segundos de espera. En este caso, RF9 no queda bloqueado y puede ejecutarse inmediatamente, ya que RFC6 no bloquea las filas consultadas. Esto demuestra que el aislamiento READ COMMITTED permite la visualización de datos comprometidos por otras transacciones que ocurren durante la ejecución de una consulta, permitiendo así la aparición de lecturas fantasma.

El componente que implementa RF9 no fue bloqueado y se ejecutó con éxito durante la espera artificial en RFC6. Debido a que el nivel READ COMMITTED permite leer los datos tal como están comprometidos en la base de datos hasta el momento de lectura, RFC6 pudo haber incluido en su resultado los cambios hechos por RF9 (es decir, una agenda ya reservada).

Este comportamiento muestra que, aunque se evita leer datos sin commit (lecturas sucias), es posible que RFC6 lea datos modificados por otras transacciones antes de que finalice su propia ejecución, lo cual puede provocar lecturas no repetibles o fantasma.

El resultado devuelto por RFC6 puede incluir o no la orden ingresada por RF9, dependiendo del momento exacto en que se realice la segunda lectura después del retardo. Si RF9 comprometió su cambio antes de que RFC6 terminara el sleep y rehiciera la consulta, entonces RFC6 verá el cambio.

Tiempo	Acción
0s	El usuario A ejecuta RFC6. La consulta se ejecuta bajo aislamiento READ COMMITTED e incluye una pausa artificial de 30 segundos.
10s	Durante la pausa, el usuario B ejecuta RF9 y agenda exitosamente un servicio, ya que no encuentra bloqueo al acceder a la agenda.
10s–30s	RF9 finaliza su transacción y deja comprometido el cambio en la agenda.
30s	RFC6 finaliza su sleep, reejecuta la consulta, y puede visualizar el estado actualizado con la agenda ya reservada por RF9.
31s	RFC6 retorna los resultados, que pueden incluir la agenda modificada por RF9, si esta fue comprometida a tiempo.