Universidad de La Laguna

Escuela superior de Ingeniería y Tecnología
Grado en Ingeniería Informática
Administración y Diseño de Bases de Datos

Informe final del proyecto

Miembros del equipo:

- Daniel Nuez Wehbe (alu0100881165@ull.edu.es)
- Carlos Fernández Donate (alu0100844665@ull.edu.es)

C/ Padre Herrera s/n
38207 La Laguna
Santa Cruz de Tenerife. España

T: 900 43 25 26

ull.es

ÍNDICE

Introducción

Este documento recoge las diferentes fases de desarrollo del proyecto final de la asignatura, desde el supuesto práctico y análisis de requisitos, hasta la implementación y prueba de la base de datos; y las conclusiones del mismo.

Supuesto práctico

El Ministerio de Sanidad de España, quiere implementar una base de datos que permita mantener una gestión interna de los hospitales del país. En un principio, se implementará dicha base de datos en el Hospital Universitario de Canarias (en Tenerife), para probar el funcionamiento de la misma y, posteriormente, implementarla en el resto de hospitales.

Actualmente, algunos procedimientos de este hospital se realizan en papel, aunque la mayoría se han informatizado. Una de las principales tareas será la informatización total de todos los procedimientos del mismo, y que se cree un histórico de los mismos.

Los trabajadores del hospital se dividen principalmente en: doctores, enfermeras y recepcionistas. Existen más empleados en el hospital, pero la base de datos ha de centrarse en estos.

De los trabajadores se quiere conocer la siguiente información: DNI, nombre y apellidos, email, dirección, sexo, salario y teléfono. Además, se quiere conocer si el doctor es permanente, de prácticas o visitante.

Los recepcionistas serán los encargados de mantener un registro de los pacientes que acuden al hospital. El registro deberá describir el motivo del paciente para acudir al hospital y quien va a atenderle. Se deberán almacenar por ejemplo citas programadas, intervenciones, estancias, etc. Además, serán los encargados de dar de alta en la base de datos del hospital a los pacientes nuevos.

Se quiere centralizar la gestión de las diferentes habitaciones del hospital (dormitorios, salas de intervenciones, consultas de doctores y salas de enfermería). Cada habitación ha de ser gestionada al menos por una enfermera, y en los dormitorios tienen una capacidad de 2 personas.

De los pacientes del hospital, se quieren conocer sus datos básicos como su nombre completo, número de la seguridad social, sexo, teléfono de contacto, dirección, contacto de emergencia, médico de cabecera, y si es necesario, el

número de seguro privado. Por último, se deberá de conocer qué enfermedades ha padecido el paciente a lo largo de los años.

Un único doctor, será asignado a cada paciente, convirtiéndose en su médico de cabecera. Pero otros doctores pueden atender a los pacientes, ya sea por una interconsulta con algún especialista, o porque el médico de cabecera no pueda atender a alguno de sus pacientes por algún motivo de fuerza mayor. Por tanto, será necesario conocer qué médico atiende a cada paciente. En el caso en el que un doctor realice una intervención a un paciente, el mismo doctor que lo operó está obligado a realizar como mínimo una revisión después de la operación para evaluar la evolución del paciente.

Los pacientes que acudan al hospital y lo necesiten, serán asignados a una habitación, en las que permanecerán durante un periodo de tiempo determinado. Solo un doctor podrá mandar el ingreso de un paciente al hospital.

Un paciente puede acudir a una cita con una enfermera en caso de necesitar una cura, vacuna, etc.

Una vez atendidos, a los pacientes se les podrá mandar un tratamiento para mejorar su estado de salud. Se quiere conocer el diagnóstico, y que medicamentos se deben utilizar. Además, se quieren almacenar todos los tratamientos que un paciente necesite a lo largo del tiempo.

Análisis de requisitos

De todos los pacientes a los que el hospital atienda se debe almacenar la siguiente información: CIP, nombre y apellidos, sexo, dirección, número de teléfono, contacto de emergencia, médico de cabecera y número de seguro privado si es necesario.

Existen 3 tipos de empleado: doctor, recepcionista y enfermera.

De todos los empleados del hospital se debe almacenar la siguiente información: dni, nombre completo, dirección, sexo, salario, número de contacto y correo electrónico.

Los doctores del hospital pueden ser de tres tipos: permanente, en prácticas y visitante.

El registro de los pacientes del hospital se realizará mediantes citas programadas.

Estas citas deberán tener la información de qué paciente acude a ver a qué doctor, una breve descripción del motivo, y la fecha de la misma.

Los pacientes solicitarán las citas, que serán gestionadas por los recepcionistas.

Los recepcionistas son los encargados de registrar cada nuevo paciente.

Las citas podrán ser de 3 tipos: consulta, intervención y enfermería.

Los doctores atienden las citas de consulta (únicamente con los pacientes para los que sea médico de cabecera) e intervención (el doctor que haga una intervención está obligado a realizar una revisión de la misma).

Los doctores al finalizar una consulta o intervención, deberán realizar un diagnóstico para el paciente, con una breve descripción y la fecha en la que se realizó.

Este diagnóstico asignará a los pacientes un tratamiento de uno o varios medicamentos.

Las enfermeras atienden las citas de enfermería, y pueden participar en las intervenciones.

Las habitaciones serán gestionadas por una o más enfermeras.

Existen 4 tipos de habitaciones: dormitorios, salas de intervención, consultas y salas de enfermería.

Los dormitorios pueden tener hasta 2 pacientes asignados al mismo tiempo.

Modelo Conceptual

Se incluye una imagen del modelo conceptual[1], pero para poder verlo en profundidad se puede visualizar en este enlace mejor.

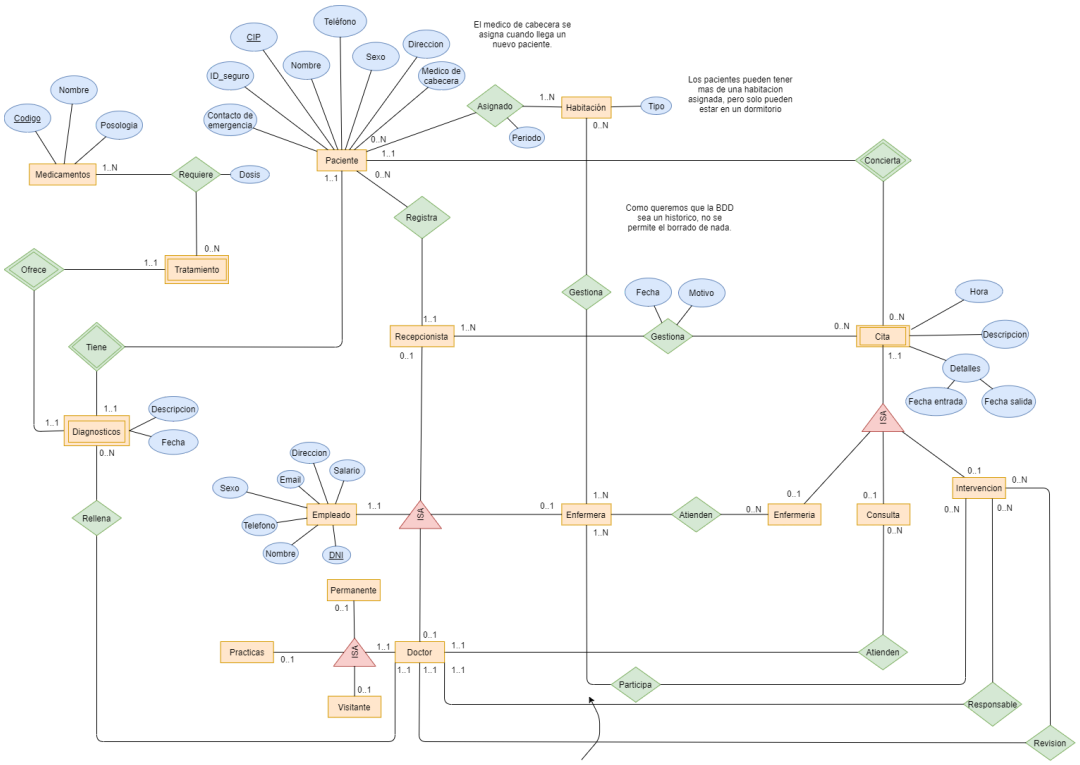

Figura 1: Modelo conceptual. Entidad Relación.

Existen 4 restricciones semánticas:

- El médico de cabecera se asigna cuando llega un nuevo paciente.
- Este médico de cabecera debe atender las citas de consulta de los pacientes que tenga asignados.
- Los pacientes pueden tener más de una habitación asignada, pero solo pueden estar en un dormitorio.
- Como queremos que la BDD sea un histórico, no se permite el borrado de nada.

Modelo Lógico

Se incluye una imagen del modelo lógico[2], pero para poder verlo en profundidad se puede visualizar en este enlace mejor.

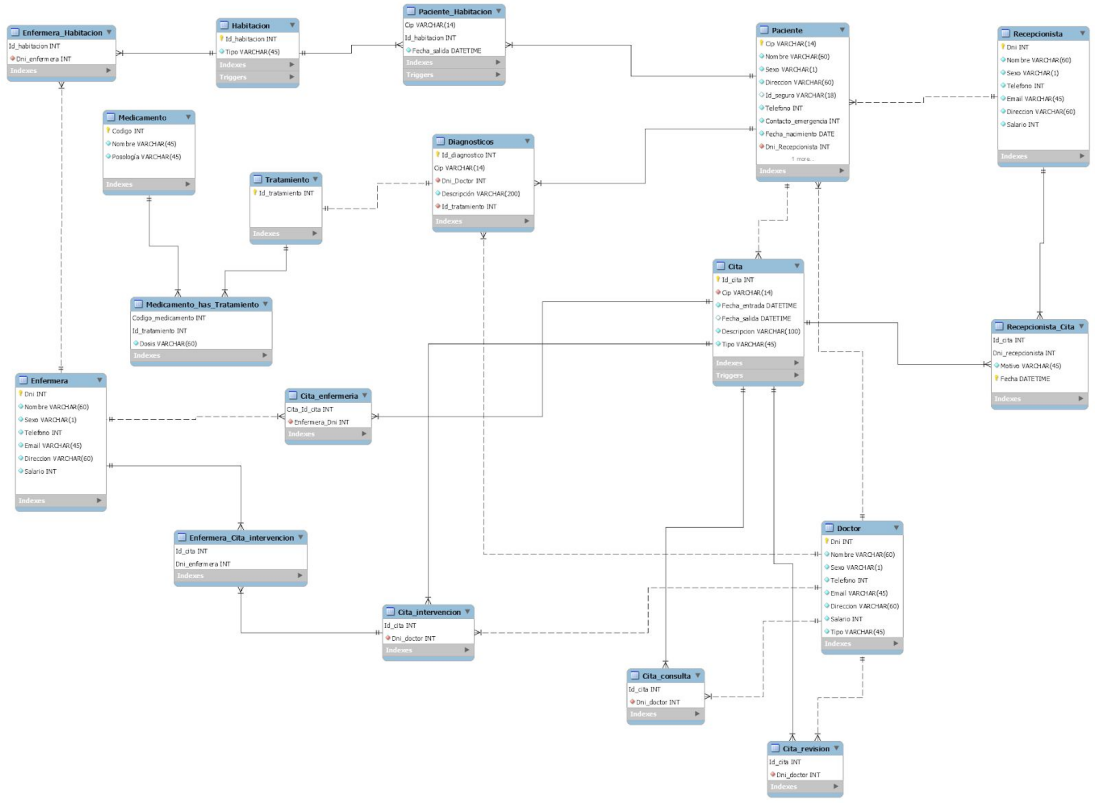

Figura 2: Modelo lógico. Captura del diagrama en MySQL Workbench.

A continuación se explicarán cada una de las relaciones, sus atributos, claves primarias, claves ajenas y dominios[3]:

1. Paciente. Recoge los datos relativos a las personas que acuden al hospital para ser atendidas, además de registrar qué recepcionista les registró por primera vez y quién es su médico de cabecera. Sus atributos son:
    a. Cip: Es el Código de Identificación Personal[4] en el Sistema de Información de Tarjeta Sanitaria. Como identifica a cada persona de manera única, se decidió utilizar este código como clave primaria. El código se representa mediante números y letras, con un VARCHAR de 14 caracteres.
    b. Nombre: Almacena el nombre completo del paciente. VARCHAR de 60 caracteres.
    c. Sexo: Se representa mediante la letra H|M, según el sexo de la persona.
    d. Dirección: Se recogen direcciones de hasta 60 caracteres. Es un VARCHAR.

e. Id_seguro: En caso de que el paciente tenga un seguro privado, se guardará el número identificatorio[5] del mismo, con una longitud máxima de 18 caracteres. Es un VARCHAR.
f. Teléfono: Almacena el número de contacto del paciente. Es de tipo INT.
g. Contacto_emergencia: En caso de que le ocurra algún accidente a un paciente, se tiene registrado un teléfono de emergencia al que contactar. Es de tipo INT.
h. Fecha_nacimiento: Para conocer la edad del paciente, se almacena su fecha de nacimiento en un campo DATE.
i. Dni_recepcionista: Se lleva control sobre qué persona registró a cada paciente. Es una clave ajena que hace referencia a la relación Recepcionista.
j. Dni_doctor_cabecera: Se quiere conocer quién es el médico de cabecera de cada paciente, por tanto, se asignará de manera aleatoria un doctor para ello. Cada vez que el paciente concierte una cita de tipo "Consulta", se le asignará una cita con su médico de cabecera. Es una clave ajena que hace referencia a la relación Doctor. Todo esto será controlado a través de un trigger[6].

Todos los atributos de la relación menos Id_seguro yt Dni_doctor_cabecera, están marcados como NOT NULL[7], ya que toda la información relativa a los pacientes tiene que estar rellenada obligatoriamente. El Id_ Seguro sólo se utilizará en el caso de que el paciente tenga seguro privado. Por otro lado, el Dni_doctor_cabecera se asignará automáticamente, sin que se tenga que especificar a la hora de añadir un usuario. Además, las dos claves ajenas están configuradas para que no se permita la eliminación de ningún doctor o recepcionista, y en caso de que se actualice el Dni de alguno de ellos, este cambio se propagará a esta relación.

Esta relación tiene un trigger, que se encarga de asignar a cada nuevo paciente un doctor aleatoriamente. El código del mismo es:

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Paciente_BEFORE_INSERT` BEFORE INSERT ON `Paciente` FOR EACH ROW
BEGIN
    IF(NEW.Dni_doctor_cabecera IS NULL) THEN
        SET NEW.Dni_doctor_cabecera = (SELECT Dni FROM Doctor
                                       ORDER BY RAND()
                                       LIMIT 1);
    END IF;
END
```

Figura 3: Trigger de Paciente.

2. Recepcionista, Doctor y Enfermera. En este caso, hablamos de 3 relaciones a la vez, ya que en el modelo Entidad Relación, cada una de ellas heredaba de un tipo ISA[8] los atributos de la una relación llamada empleado. Se optó por implementar este tipo ISA en el modelo lógico creando 3 relaciones independientes, que contienen los atributos de la relación empleado, ya que facilitaba mucho el control de las mismas. Los atributos de las 3 relaciones son:
    a. Dni: Sirve como identificador único, por tanto hará de clave primaria. Es un INT.
    b. Nombre: Almacena el nombre completo del empleado. Tiene un tamaño máximo de 60 caracteres. Es un VARCHAR.
    c. Sexo: Se representa mediante la letra H|M, según el sexo de la persona.
    d. Teléfono: Almacena el número de contacto del empleado. Es de tipo INT.
    e. Email: Dirección de correo electrónico del empleado. Se aceptan emails de hasta 45 caracteres de longitud. Es de tipo VARCHAR.
    f. Dirección: Se recogen direcciones de hasta 60 caracteres. Es un VARCHAR.
    g. Salario: Se almacena también el sueldo mensual de cada empleado. Es un INT.

    En la relación Doctor existe un atributo más, que los otras dos no tienes. Este atributo es:

    - Tipo: Este atributo es necesario ya que existen 3 clases de doctores: permanentes, visitantes y en prácticas. Según el supuesto práctico esto no significa ningún tipo de restricción a la hora de atender a los pacientes, pero se quiere conocer qué tipo de doctor es cada uno. Es un VARCHAR de tamaño 10.

    Todos los atributos de la relación están marcados como NOT NULL, ya que toda la información relativa a los empleados tiene que estar rellenada obligatoriamente. No existe ningún trigger ni ninguna clave ajena en estas relaciones.

3. Cita. Como su nombre indica, almacena la información relativa a las citas que conciertan los pacientes. Sus atributos son:
    a. Id_cita: Código que servirá para diferenciar unas citas de otras. Se utilizará como clave primaria. Es un número entero incremental, es decir, la primera cita tendrá el identificador 1, la segunda el 2, etc. Es de tipo INT.

b. Cip: Código de Identificación Personal en el Sistema de Información de Tarjeta Sanitaria. Es una clave ajena, que hace referencia a la relación paciente.
c. Fecha_entrada: Es la fecha y hora a la que el paciente tiene su cita. Es de tipo DATETIME[9].
d. Fecha_salida: En caso de que fuera necesario ingresar a un paciente en el hospital, una vez le den el alta, se registrará la fecha y hora de su salida del hospital. Es de tipo DATETIME.
e. Descripción: Recoge el motivo por el que el paciente concertó la cita, teniendo una longitud máxima de 100 caracteres. Es un VARCHAR.
f. Tipo: Existen 4 tipos de cita: consulta, intervención, enfermería y revisión. Es un VARCHAR de 12 caracteres.

Todos los atributos de la relación menos Fecha_salida, están marcados como NOT NULL, ya que la fecha de salida debe ser rellenada mediante un UPDATE[10], una vez el paciente reciba el alta médica. Además, la clave ajena está configurada para que no se permita la eliminación de ningún pacientes, y en caso de que se actualice el Cip de alguno de ellos, este cambio se propagará a esta relación.

Esta relación tiene 2 triggers, uno Before Insert, y otro After Insert.

En el Before Insert, se comprueba que el usuario que está insertando sea un recepcionista, o el usuario root.

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Cita_BEFORE_INSERT` BEFORE INSERT ON `Cita` FOR EACH ROW
BEGIN
    IF ((select NOT EXISTS (select * from Recepcionista where Dni=(SELECT substring_index(USER(), '@', 1))))
        AND (SELECT substring_index(USER(), '@', 1) != 'root'))
    THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT ='El usuario no es un recepcionista';
    END IF;
END
```

Figura 4: Trigger de cita antes de insertar.

Por otro lado, en el After Insert, se controla qué tipo de cita está siendo insertada, y se inserta en la tabla correspondiente. En el caso de la cita de tipo intervención, se debe añadir a Cita_intervencion, Cita_revision y Enfermera_Cita_intervencion, el atributo de Id_cita, en las tablas de intervención y revisión el dni del doctor responsable, y en la tabla de las enfermeras, el dni de la enfermera que participa en la intervención. Y finalmente, siempre que se crea una cita nueva se debe de añadir también en la tabla Recepcionista_Cita qué cita fue creada, por quién y cuándo. Se coge el dni del recepcionista que esté añadiendo la cita. Por defecto, cuando

se crea una cita de tipo revisión, esta tiene fecha para una semana después de la intervención.

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Cita_AFTER_INSERT` AFTER INSERT ON `Cita` FOR EACH ROW
BEGIN
    IF (NEW.Tipo='Enfermeria')
    THEN
        INSERT INTO Cita_enfermeria(Id_cita, Dni_enfermera) VALUES(NEW.Id_cita, (SELECT Dni FROM Enfermera
                                                                                 ORDER BY RAND()
                                                                                 LIMIT 1));
    ELSEIF (NEW.Tipo='Intervencion')
    THEN
        INSERT INTO Cita_intervencion(Id_cita, Dni_doctor) VALUES(NEW.Id_cita, (SELECT Dni FROM Doctor
                                                                                ORDER BY RAND()
                                                                                LIMIT 1));
        INSERT INTO Enfermera_Cita_intervencion(Id_cita, Dni_enfermera) VALUES(NEW.Id_cita, (SELECT Dni FROM Enfermera
                                                                                             ORDER BY RAND()
                                                                                             LIMIT 1));
        INSERT INTO Cita_revision(Id_cita, Dni_doctor) VALUES(NEW.Id_cita, (SELECT Dni_doctor FROM Cita_intervencion
                                                                            WHERE Id_cita=NEW.Id_cita));
    ElSEIF (NEW.Tipo='Consulta')
    THEN
        INSERT INTO Cita_consulta(Id_cita, Dni_doctor) VALUES(NEW.Id_cita, (SELECT Dni FROM Doctor
                                                                            WHERE Dni = (SELECT Dni_doctor_cabecera
                                                                                         FROM Paciente
                                                                                         WHERE Cip=NEW.Cip)));
    END IF;
    IF(SELECT substring_index(USER(), '@', 1) = 'root')
    THEN
        INSERT INTO Recepcionista_Cita(Id_cita,Dni_recepcionista,Motivo,Fecha)
        VALUES(NEW.Id_cita,'11111111','Nueva cita', NOW());
    ELSE
        INSERT INTO Recepcionista_Cita(Id_cita,Dni_recepcionista,Motivo,Fecha)
        VALUES(NEW.Id_cita,(SELECT substring_index(USER(), '@', 1)),'Nueva cita', NOW());
    END IF;

END
```

Figura 5: Trigger de cita después de insertar.

4. Recepcionista_cita. Cada vez que algún recepcionista realice algún cambio en las citas, este quedará registrado en esta relación. Sus atributos son:
    a. Id_cita: Identificador de la cita. Es parte de la clave primaria y es clave ajena.
    b. Dni_recepcionista: Identificador del recepcionista. Es parte de la clave primaria y es clave ajena.
    c. Motivo: El recepcionista puede crear, modificar o cancelar una cita, y al hacerlo deberá reflejar el qué ha hecho. Es un VARCHAR de 45 caracteres.

d. Fecha: Momento en el que se creó, modificó o canceló una cita. Se almacena tanto la fecha como la hora. Es parte de la clave primaria. Es un DATETIME.

Todos los atributos de la relación están marcados como NOT NULL. Además, las dos claves ajenas están configuradas para que no se permita la eliminación de ninguna cita o recepcionista, y en caso de que se actualice el Id_cita o Dni de alguno de ellos, este cambio se propagará a esta relación. Se ha optado por elegir una clave compuesta por 3 atributos, ya que se podría dar el caso en el que un mismo recepcionista modifique varias veces en un mismo día una misma cita.

5. Cita_consulta. Esta relación es utilizada para conocer qué doctores tienen que atender las citas que son de tipo consulta. Sus atributos son:
    a. Id_cita: Identificador de la cita. Es la clave primaria y es clave ajena.
    b. Dni_doctor: Identificador del doctor. Es clave ajena.

    Las dos claves ajenas están configuradas para que no se permita la eliminación de ninguna cita o doctor, y en caso de que se actualice el Id_cita o Dni de alguno de ellos, este cambio se propagará a esta relación. En este tipo de cita, solo podrá pasar cita un doctor, por tanto la única clave primaria es Id_cita. Como se explica en el Entidad Relación, los doctores deben atender a los pacientes que tienen asignados en este tipo de cita. Esto se controla desde el trigger de la tabla cita.

6. Cita_enfermeria. Se recogen las enfermeras que están asignadas a las citas de tipo enfermería. Sus atributos son:
    a. Id_cita: Identificador de la cita. Es la clave primaria y es clave ajena.
    b. Dni_enfermera: Identificador de la enfermera. Es clave ajena.

    Las dos claves ajenas están configuradas para que no se permita la eliminación de ninguna cita o enfermera, y en caso de que se actualice el Id_cita o Dni de alguno de ellos, este cambio se propagará a esta relación. Las citas de enfermería sólo podrán ser atendidas por una enfermera, por tanto solo es clave primaria Id_cita. La asignación de enfermeras se controla con un trigger en la tabla cita.

7. Cita_intervencion. Almacena los doctores que se asignan para una operación. Sus atributos son:
    a. Id_cita: Identificador de la cita. Es la clave primaria y es clave ajena.
    b. Dni_doctor: Identificador del doctor. Es clave ajena.

    Las dos claves ajenas están configuradas para que no se permita la eliminación de ninguna cita o doctor, y en caso de que se actualice el Id_cita

o Dni de alguno de ellos, este cambio se propagará a esta relación. Las intervenciones solo tienen un único doctor como responsable, por tanto Id_cita es la única clave primaria. Los doctores se asignan mediante un trigger en la tabla cita.

8. Enfermera_cita_intervencion. Recoge a las enfermeras que participan en una intervención. Sus atributos son:
    a. Id_cita: Identificador de la cita. Es parte de la clave primaria y es clave ajena.
    b. Dni_enfermera: Identificador de la enfermera. Es parte de la clave primaria y es clave ajena.

    Las dos claves ajenas están configuradas para que no se permita la eliminación de ninguna cita o enfermera, y en caso de que se actualice el Id_cita o Dni de alguno de ellos, este cambio se propagará a esta relación. En este caso, sí podemos tener varias enfermeras participando en una misma intervención, por tanto ambos atributos son clave primaria. Se controla la asignación de las enfermeras desde un trigger en la tabla cita.

9. Cita_revision. Los doctores que realicen una intervención están obligados a realizar una revisión al mismo paciente al que operaron. En esta tabla se recoge a los doctores y las citas de este tipo. Sus atributos son:
    a. Id_cita: Identificador de la cita. Es parte de la clave primaria y es clave ajena.
    b. Dni_doctor: Identificador del doctor. Es clave ajena.

    Las dos claves ajenas están configuradas para que no se permita la eliminación de ninguna cita o doctor, y en caso de que se actualice el Id_cita o Dni de alguno de ellos, este cambio se propagará a esta relación. La revisión deberá ser llevada a cabo por el mismo doctor que realizó la intervención, por tanto la única clave primaria es Id_cita. Esto se controla mediante un trigger en la relación cita.

10. Habitacion. Esta relación identifica y diferencia las habitaciones del hospital. Sus atributos son:
    a. Id_habitacion: Código que servirá para diferenciar a las habitaciones entre ellas. Se utilizará como clave primaria. Es un número entero incremental, es decir, la primera cita tendrá el identificador 1, la segunda el 2, etc. Es de tipo INT. Es la clave primaria.
    b. Tipo: Actualmente se contemplan 3 tipos de habitación diferentes: dormitorio, quirófano y enfermería. Es un VARCHAR de 10 caracteres.

El atributo Tipo, está marcado como NOT NULL, ya que es importante conocer qué tipo de habitación es cada una para posteriormente poder asignarlas a pacientes.

Existe un trigger en esta relación, el cual una vez se inserta una nueva habitación, le asigna automáticamente, y de manera aleatoria, una enfermera para gestionar la habitación. Esta asignación se produce realizando un insert en la tabla Enfermera_habitacion.

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Habitacion_AFTER_INSERT` AFTER INSERT ON `Habitacion` FOR EACH ROW
BEGIN
    INSERT INTO Enfermera_Habitacion(Id_habitacion, Dni_enfermera)
    VALUES(NEW.Id_habitacion, (SELECT Dni FROM Enfermera ORDER BY RAND() LIMIT 1));
END
```

Figura 6: Trigger de habitación después de insertar.

11. Enfermera_habitacion. Recoge qué enfermeras se encargan de qué habitaciones. Sus atributos son:
    a. Id_habitacion: Código que servirá para diferenciar a las habitaciones entre ellas. Se utilizará como parte de la clave primaria.
    b. Dni_enfermera: Identificador de la enfermera. Es parte de la clave primaria y es clave ajena.

    Ambos atributos son clave primaria, ya que puede las habitaciones deben estar gestionadas por al menos una enfermera. Las clave ajenas está configurada para que no se permita la eliminación de ninguna enfermera, y en caso de que se actualice el Dni de alguno de ellos, este cambio se propagará a esta relación.

12. Paciente_habitacion. Recoge los datos relativos a las personas que acuden al hospital para ser atendidas, además de registrar qué recepcionista les registró por primera vez y quién es su médico de cabecera. Sus atributos son:
    a. Cip: Código de Identificación Personal en el Sistema de Información de Tarjeta Sanitaria. Es parte de la clave primaria.
    b. Id_habitacion: Código que servirá para diferenciar a las habitaciones entre ellas. Es parte de la clave primaria.
    c. Fecha_salida: Momento en el que el paciente dejará de estar asignado a una habitación. Por ejemplo en caso de una habitación de tipo quirófano, la fecha de salida correspondería con la hora aproximada a la que finalizaría la intervención. Es de tipo DATETIME. Es parte de la clave primaria.

Las clave ajena está configurada para que no se permita la eliminación de ninguna habitación, y en caso de que se actualice el Id_habitacion, este cambio se propagará a esta relación. Se ha optado por incluir a la Fecha_salida como clave primaria, ya que puede darse el caso de que un paciente esté asignado a una misma habitación, pero en un momento diferente del tiempo, y queremos que la base de datos sea un histórico.

Existe un trigger que controla por un lado, que al asignar un paciente a un dormitorio, no esté ya asignado a otro; y por otro lado, que no se asigne a un paciente un dormitorio que esté lleno, es decir, en la que ya hayan 2 personas asignadas. Se tiene en cuenta la fecha en la que el paciente abandona la habitación.

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Paciente_Habitacion_BEFORE_INSERT` BEFORE INSERT ON `Paciente_Habitacion` FOR EACH ROW
BEGIN
    /** Trigger que controle que el paciente que se está insertando no tenga asignada una habitación ya del mismo tipo **/
    IF( SELECT Cip
        FROM Paciente_Habitacion
        WHERE Cip=New.Cip AND Fecha_salida > NOW()
        AND Id_habitacion IN (SELECT Id_habitacion
                              FROM Habitacion
                              WHERE Tipo='Dormitorio'))
    THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT ='El paciente ya tiene un dormitorio asignado.';

    /** Trigger que controle que el paciente que se está insertando no tenga asignada una habitación ya del mismo tipo **/
    ELSEIF(SELECT COUNT(*) AS cnt
         FROM Paciente_Habitacion
         GROUP BY Id_habitacion
         HAVING cnt=2 AND Id_habitacion=NEW.Id_habitacion
         AND Id_habitacion IN (SELECT Id_habitacion
                               FROM Habitacion
                               WHERE Tipo='Dormitorio'))
    THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT ='El dormitorio ya está lleno.';
    END IF;
END
```

Figura 7: Trigger de Paciente_habitacion antes de insertar.

13. Diagnostico. Almacena los diagnósticos que rellenan los doctores a los pacientes. Sus atributos son:
    a. Id_diagnostico: Código que servirá para diferenciar a los diagnósticos. Se utilizará como clave primaria.
    b. Cip: Código de Identificación Personal en el Sistema de Información de Tarjeta Sanitaria. Es clave ajena
    c. Dni_doctor: Identificador de la enfermera. Es clave ajena.
    d. Descripcion: Describe el diagnóstico que el doctor asigna al paciente. Es un VARCHAR de 200 caracteres.

e. Fecha: Fecha y hora en la que se crea el diagnóstico. Es de tipo DATETIME.

Todos los atributos de la relación están marcados como NOT NULL. Además, las dos claves ajenas están configuradas para que no se permita la eliminación de ningún doctor o paciente, y en caso de que se actualice el Dni o Cip de alguno de ellos, este cambio se propagará a esta relación.

Tiene un trigger que se encarga de controlar que las personas que inserten en la relación sean doctores.

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Diagnostico_BEFORE_INSERT` BEFORE INSERT ON `Diagnostico` FOR EACH ROW
BEGIN
    IF ((select NOT EXISTS (select * from Doctor where Dni=(SELECT substring_index(USER(), '@', 1))))
         AND (SELECT substring_index(USER(), '@', 1) != 'root'))
    THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT ='El usuario no es un doctor';
    END IF;
END
```

Figura 8: Trigger de Diagnóstico antes de insertar.

14. Tratamiento. Los diagnósticos asignan un tratamiento a los pacientes. Esta relación sirve para relacionar los diagnósticos con el tratamiento que asignan al paciente. Sus atributos son:
    a. Id_tratamiento: Código que servirá para diferenciarlos tratamientos. Es un número entero incremental, es decir, la primera cita tendrá el identificador 1, la segunda el 2, etc. Es de tipo INT. Es parte de la clave primaria.
    b. Id_diagnostico: Código que servirá para diferenciar a los diagnósticos. Es parte de la clave primaria y es clave ajena.

    La clave ajena está configurada para que no se permita la eliminación de ningún diagnóstico, y en caso de que se actualice el Id_diagnostico de alguno, este cambio se propagará a esta relación.

15. Medicamento. Contiene los medicamentos que son asignados en los tratamientos. Sus atributos son:
    a. Codigo: Código identificatorio del medicamento. Corresponde al PC[11] del envase. Es de tipo INT. Es la clave primaria.
    b. Nombre: Corresponde al nombre del medicamento. Es un VARCHAR de 45 caracteres.

c. Posologia[12]: Almacena la dosis de los medicamentos, tanto la cantidad de medicamento como el intervalo de tiempo entre las administraciones sucesivas. Es un VARCHAR de 45 caracteres.

Todos los atributos de la relación, están marcados como NOT NULL, ya que toda la información relativa a los medicamentos tiene que estar rellenada obligatoriamente.

16. Medicamento_tratamiento. Es una relación intermedia entre Medicamento y Tratamiento, que se encarga de especificar qué medicamentos se asignan a cada tratamiento, y la dosis del mismo . Sus atributos son:
    a. Codigo_medicamento: Código identificatorio del medicamento. Corresponde al PC del envase. Es parte de la clave primaria y es clave ajena.
    b. Id_tratamiento: Código que servirá para diferenciarlos tratamientos. Es parte de la clave primaria y es clave ajena.
    c. Dosis: Recoge la dosis de cada medicamento asignado a cada tratamiento. Es un VARCHAR de 60 caracteres.

    El atributo dosis está marcado como NOT NULL, ya que especificar cómo se debe consumir el medicamento, es obligatorio. Además, las dos claves ajenas están configuradas para que no se permita la eliminación de ningún medicamento o tratamiento, y en caso de que se actualice el Codigo_medicamento o el Id_tratamiento de alguno, este cambio se propagará a esta relación.

Modelo Objeto Relacional

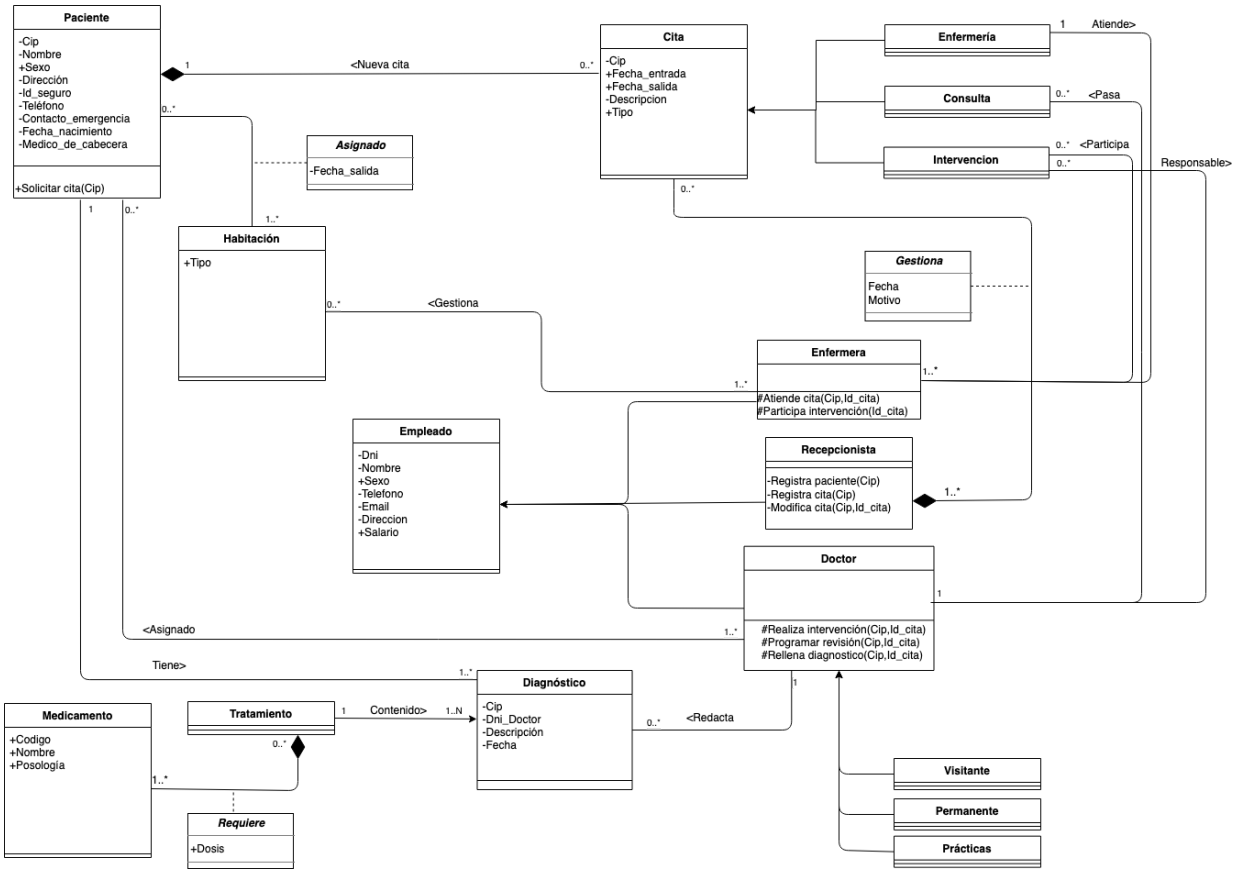

Figura 9: Modelo Objeto Relacional. Uml.

La figura 3 es el resultado de aplicar las transformaciones necesarias al diagrama entidad relación, para obtener un diagrama de clases Uml. Si desea ver el diagrama Uml original puede acceder a él a través de este link.

Las transformaciones más reseñables son las del tipo agregación por composición por ejemplo la que hay entre Medicamento y Tratamiento donde el todo es el tratamiento y la parte es el medicamento y la dosis correspondiente.

```
CREATE TYPE Paciente AS
(Cip                    VARCHAR(14),
 Nombre                 VARCHAR(60),
 Sexo                   VARCHAR(1),
 Dirección              VARCHAR(60),
 Id_seguro              VARCHAR(18),
 Teléfono               INTEGER,
 Contacto_emergencia    INTEGER,
 Fecha_nacimiento       DATE,
 Dni_doctor_cabecera    REF(Doctor),
 Recepcionista          REF(Recepcionista),
 Solicitar_cita         METHOD(Cip));

CREATE TABLE Tabla_Pacientes OF Paciente
(Cip PRIMARY KEY,
 Nombre NOT NULL
 Sexo NOT NULL,
 Dirección NOT NULL,
 Telefono NOT NULL,
 Contacto_emergencia NOT NULL,
 Fecha_nacimiento NOT NULL,
 Dni_doctor_cabecera NOT NULL,
 Recepcionista NOT NULL);
```

```
CREATE TYPE T_Gestiona AS
(Fecha          DATETIME,
 Motivo         VARCHAR(45),
 Recepcionista  REF(Recepcionista));

CREATE TYPE Cita AS
(Id_cita        INTEGER,
 Cip            REF(Paciente),
 Fecha_entrada  DATETIME,
 Fecha_salida   DATETIME,
 Descripción    VARCHAR(45),
 Tipo           VARCHAR(45),
 Gestionada_Por T_gestiona MULTISET);

CREATE TABLE Tabla_Citas OF Citas
(Id_cita PRIMARY KEY,
 Cip NOT NULL,
 Fecha_entrada NOT NULL,
 Descripción NOT NULL,
 Tipo NOT NULL,
 Gestionada_por NOT NULL);
```

```
CREATE TYPE Cita_Intervención AS
(Dni_doctor     REF(Doctor)
 Enfermeras     REF(Enfermera) MULTISET);

CREATE TABLE Tabla_Citas_Intervención OF Cita_Intervención UNDER
Cita
(Dni_doctor NOT NULL,
 Enfermeras NOT NULL);
```

```
CREATE TYPE Cita_Enfermería AS
(Dni_enfermera  REF(Enfermera));

CREATE TABLE Tabla_Citas_Enfermería OF Cita_Enfermeria UNDER Cita
(Dni_doctor NOT NULL);
```

```
CREATE TYPE Cita_Consulta AS
(Dni_doctor     REF(Doctor));

CREATE TABLE Tabla_Citas_Consulta OF Cita_Consultas UNDER Cita
(Dni_doctor NOT NULL);
```

```
CREATE TYPE Enfermera AS
(Atender_cita           METHOD(Cip,Id_cita),
 Participa_intervención METHOD(Id_cita));
 CREATE TABLE Tabla_Enfermeras OF Enfermera UNDER Empleado;
```

```
CREATE TYPE Recepcionista AS
(Registra_paciente      METHOD(Cip),
 Registra_cita          METHOD(Cip),
 Modifica_cita          METHOD(Cip,Id_cita));
 CREATE TABLE Tabla_Recepcionistas OF Recepcionista UNDER Empleado;
```

```
CREATE TYPE Doctor AS
(Realiza_intervención   METHOD(Cip,Id_cita),
 Programar_revisión     METHOD(Cip,Id_cita),
 Rellenar_diagnostico   METHOD(Cip,Id_cita));
 CREATE TABLE Tabla_Doctores OF Doctor UNDER Empleado;
```

```
CREATE TYPE Permanente AS();
CREATE TABLE Tabla_Doctores_Permanentes OF Permanente UNDER Doctor;
```

```
CREATE TYPE Visitante AS();
CREATE TABLE Tabla_Doctores_Visitantes OF Visitante UNDER Doctor;
```

```
CREATE TYPE Prácticas AS();
CREATE TABLE Tabla_Doctores_Prácticas OF Prácticas UNDER Doctor;
```

```
CREATE TYPE Empleado AS
(Dni        INTEGER,
 Nombre     VARCHAR(60),
 Sexo       VARCHAR(1),
 Telefono   INTEGER,
 Email      VARCHAR(45),
 Dirección  VARCHAR(60),
 Salario    INTEGER);

CREATE TABLE Tabla_Empleados OF Empleado
(Dni PRIMARY KEY,
 Nombre NOT NULL,
 Sexo NOT NULL,
 Telefono NOT NULL,
 Email NOT NULL,
 Dirección NOT NULL,
 Salario NOT NULL);
```

```
CREATE TYPE Medicamento AS
(Codigo         INTEGER,
 Nombre         VARCHAR(45),
 Posología      VARCHAR(45));

CREATE TABLE Tabla_Medicamentos OF Medicamento
(Codigo PRIMARY KEY,
 Nombre NOT NULL,
 Posología NOT NULL);
```

```
CREATE TYPE Tratamiento AS
(Medicamento        REF(Medicamento) MULTISET,
 Dosis              VARCHAR(60) MULTISET,
 Justificado_por    REF(Diagnostico));
CREATE TABLE Tabla_Tratamientos OF Tratamiento;
```

```
CREATE TYPE Diagnóstico AS
(Id_diagnóstico     INTEGER,
 Cip                REF(Paciente),
 Dni_doctor         REF(Doctor),
 Descripción        VARCHAR(200),
 Fecha              DATETIME);

CREATE TABLE Tabla_Diagnosticos OF Diagnóstico
(Id_diagnóstico PRIMARY KEY,
 Cip PRIMARY KEY,
 Dni_doctor NOT NULL,
 Descripción NOT NULL,
 Fecha NOT NULL);
```

```
CREATE TYPE T_Asignada AS
(Fecha_salida       DATETIME
 Cip                REF(Paciente));

CREATE TYPE Habitación AS
(Id_habitación      INTEGER,
 Tipo               VARCHAR(45)
 Gestionada_Por     REF(Enfermera));

CREATE TABLE Tabla_habitaciones OF Habitación
(Id_habitación PRIMARY KEY,
 Tipo NOT NULL,
 Gestionada_Por NOT NULL);
```

Figura 10: Modelo Objeto Relacional. SQL 2003.

La figura 4 se corresponde con una captura del diagrama SQL extensión 2003 resultado de aplicar las transformaciones necesarias al diagrama UML. Si desea visualizar este documento en profundidad puede acceder a él en el siguiente link.

Las transformaciones más reseñables son las del tipo de agregación simple a REF + MULTISET. Por ejemplo Tratamiento donde la relación por agregación con Medicamento se transforma en REF(Medicamento) MULTISET.

Para entenderlo con mayor facilidad observe con detenimiento la figura 5.

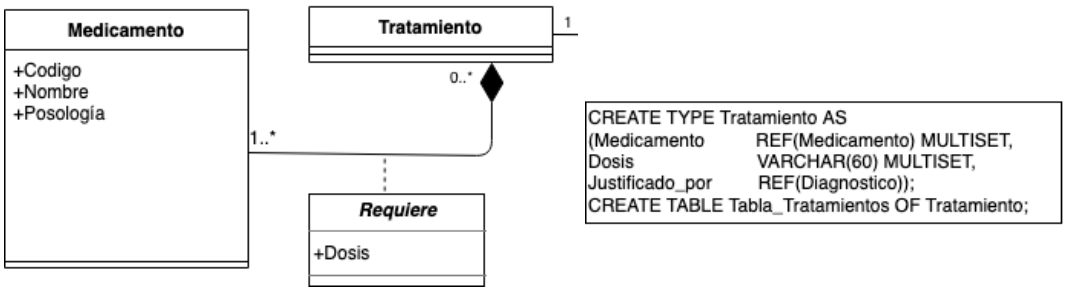

Figura 11: Modelo Objeto Relacional. SQL 2003 ejemplo transformación.

Sistema Gestor de Base de Datos

Se ha utilizado PostgreSQL como Gestor de Base de Datos[13], como se indica en el enunciado del trabajo final. Para ello se ha realizado un despliegue de una máquina virtual en el IaaS de la ULL para alojar la base de datos.

Generación de scripts y transformación

Para montar la base de datos con PostgreSQL, se exportó el modelo lógico desde el programa MySQL Workbench, el cual genera un script .sql, que contiene los comandos necesarios para la creación de tablas, triggers e inserts.

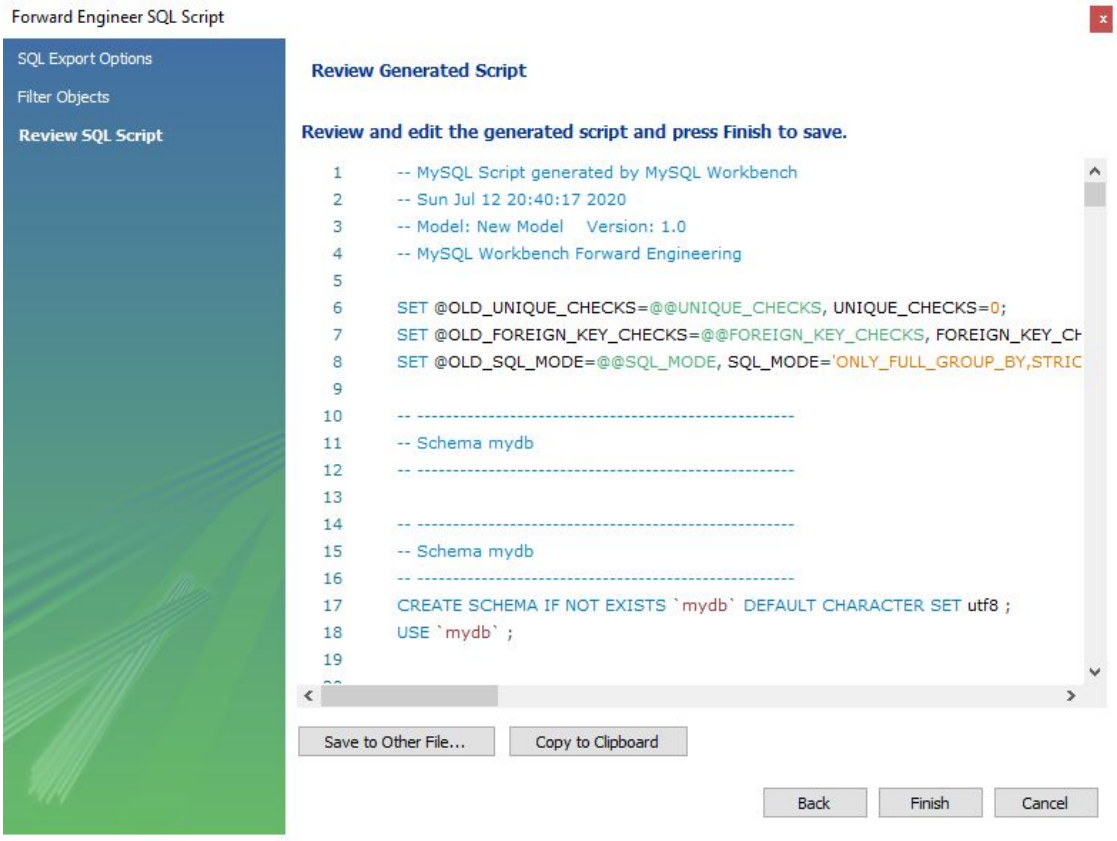

Figura 12: Resultado de generación de scripts con MySQL Workbench.

Una vez generados estos scripts, se debe ajustar la sintaxis para que sean válidos para PostgreSQL. Existen algunas herramientas online que permiten adaptar hasta cierto punto el código automáticamente, pero nosotros optamos por hacerlo de manera manual, ya que nos resultaba más sencillo y daba menos problemas. Un ejemplo de tabla en sintaxis de PostgreSQL es la siguiente:

```
CREATE TABLE IF NOT EXISTS Recepcionista (
  Dni INT NOT NULL,
  Nombre VARCHAR(60) NOT NULL,
  Sexo VARCHAR(1) NOT NULL,
  Telefono INT NOT NULL,
  Email VARCHAR(45) NOT NULL,
  Direccion VARCHAR(60) NOT NULL,
  Salario INT NOT NULL,
  PRIMARY KEY (Dni));
```

Figura 13: Tabla adaptada a la sintaxis de PostgreSQL.

Del mismo modo, se deben adaptar los triggers. Esta tarea es algo más complicada, ya que PostgreSQL utiliza lo que se conoce como Procedures[14]. Crearemos un trigger que se encargará de ejecutar una función, la cual realizará el trabajo que hacían los triggers en MySQL. Un ejemplo de transformación de trigger es el siguiente:

```
CREATE OR REPLACE FUNCTION paciente_control_habitacion() RETURNS trigger AS
$$
BEGIN
/* Trigger que controle que el paciente que se está insertando no tenga
asignada una habitación ya del mismo tipo */
        IF EXISTS (SELECT cip FROM paciente_habitacion WHERE cip=NEW.cip
                    AND Fecha_salida > NOW() AND id_habitacion IN
                    (SELECT id_habitacion FROM habitacion WHERE tipo='Dormitorio'))
                    THEN
                RAISE EXCEPTION 'El paciente ya tiene una habitación asignada';

/* Trigger que controle que la habitación no esté llena */
        ELSEIF(SELECT COUNT(*) AS cnt FROM paciente_habitacion
                WHERE Fecha_salida > NOW() GROUP BY id_habitacion
                HAVING id_habitacion=NEW.id_habitacion
                AND id_habitacion IN (SELECT id_habitacion FROM habitacion
                WHERE tipo='Dormitorio')) >= 2 THEN
                 RAISE EXCEPTION 'El dormitorio ya está lleno';
        END IF;

        RETURN NEW;

END;
$$
LANGUAGE plpgsql;

/***/

CREATE TRIGGER paciente_habitacion_before_insert
BEFORE INSERT ON paciente_habitacion
FOR EACH ROW
EXECUTE PROCEDURE paciente_control_habitacion();
```

Figura 14: Tabla adaptada a la sintaxis de PostgreSQL. La figura 7 es el trigger en MySQL

Todos los triggers, tanto de PostgreSQL como de MySQL, pueden ser visualizados en el github.

Como solo los recepcionistas pueden manipular la relación citas, y los doctores la relación diagnóstico, se ha creado un script para crear usuarios de prueba. Existe un usuario llamado "*usuario*" con permisos sobre toda la base de datos del hospital, y hará de administrador. Además, existen 3 usuarios más:

- "*r11111111*", corresponde al recepcionista con dni=11111111.
- "*d66666666*", corresponde al doctor con dni=66666666.
- "*e20202020*", corresponde al enfermera con dni=20202020.

En un principio, se intentó trabajar con usuario cuyo nombre fuera el dni directamente, sin el carácter al principio. Pero esto daba problemas, ya que

permitía crear al usuario, pero luego no se podía manipular ni para dar permisos ni para eliminarlo.

Una vez creados, se les daba permisos en las tablas que tendrían que manipular.

```
-- ------------------------------------------------------
-- Create roles with passwords
-- ------------------------------------------------------
CREATE ROLE usuario WITH LOGIN CREATEDB ENCRYPTED PASSWORD '1234';
CREATE ROLE r11111111 WITH LOGIN ENCRYPTED PASSWORD '1234';
CREATE ROLE d66666666 WITH LOGIN ENCRYPTED PASSWORD '1234';
CREATE ROLE e20202020 WITH LOGIN ENCRYPTED PASSWORD '1234';

-- ------------------------------------------------------
-- Give privileges to usuario
-- ------------------------------------------------------
GRANT CONNECT ON DATABASE hospital TO usuario;
GRANT USAGE ON SCHEMA public TO usuario;
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO usuario;
GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO usuario;

-- ------------------------------------------------------
-- Give privileges to other users
-- ------------------------------------------------------
GRANT CONNECT ON DATABASE hospital TO r11111111;
GRANT ALL PRIVILEGES ON TABLE cita TO r11111111;
GRANT ALL PRIVILEGES ON TABLE paciente TO r11111111;
GRANT ALL PRIVILEGES ON TABLE recepcionista_cita TO r11111111;
GRANT ALL PRIVILEGES ON TABLE cita_enfermeria TO r11111111;
GRANT ALL PRIVILEGES ON TABLE cita_intervencion TO r11111111;
GRANT ALL PRIVILEGES ON TABLE cita_revision TO r11111111;
GRANT ALL PRIVILEGES ON TABLE cita_consulta TO r11111111;
GRANT ALL PRIVILEGES ON TABLE enfermera_cita_intervencion TO r11111111;
GRANT SELECT ON TABLE doctor TO r11111111;
GRANT SELECT ON TABLE enfermera TO r11111111;
```

Figura 15: Creación de roles (usuarios), y asignación de permisos.

En la figura anterior, se puede ver como ejemplo la asignación de permisos al usuario "*r1111111*".

Conclusiones

La realización de este proyecto ha sido muy beneficiosa, ya que nos ha permitido enfrentarnos con el problema de diseñar e implementar una base de datos más realista, y ver cómo nos desenvolvemos ante las dificultades que van apareciendo en un proceso de desarrollo como este.

Hemos tenido algunas dificultades a la hora de realizar el modelo lógico, ya que hubieron 2 veces en las que nos faltaron indicar restricciones semánticas, por lo que se tuvo que desechar lo que se había obtenido de la transformación del Entidad Relación para corregirlo, y entonces realizar la transformación al modelo lógico correctamente.

En cuanto a la elección del sistema gestor de bases de datos, creemos que PostgreSQL es una gran herramienta, ya que es bastante cómodo y sencillo de utilizar.

Estamos satisfechos con el resultado obtenido, aunque nos hubiera gustado implementar más triggers, ya que hay muchas cosas que se pueden controlar en una base de datos como esta. Pero creemos que tampoco iban a aportar mucho más, ya que la mayoría eran muy parecidos a los que ya tenemos implementados, por lo que tampoco iban a aportar algo muy novedoso. Además, hemos creado los triggers que son más importantes a nuestro parecer.

Referencias

[1] https://es.wikipedia.org/wiki/Modelo_conceptual

[2] https://www.ibm.com/support/knowledgecenter/es/SS9UM9_9.1.2/com.ibm.datatools.logical.ui.doc/topics/clogmod.html

[3] https://es.wikipedia.org/wiki/Base_de_datos_relacional#Relaciones_(caracter%C3%ADsticas_en_com%C3%BAn)

[4] https://www.sergas.es/Tarxeta-sanitaria/Que-informaci%C3%B3n-cont%C3%A9n?idioma=es#:~:text=2.-,CIP%3A%20c%C3%B3digo%20de%20identificaci%C3%B3n%20personal%20en%20el%20Sistema%20de%20Informaci%C3%B3n,ES%20un%20dato%20opcional.

[5] https://www.asisa.es/preguntas-frecuentes/preguntas/mi-seguro-medico/tarjeta-sanitaria/261-que-informacion-proporciona-la-numeracion-de-mi-tarjeta-sanitaria

[6] https://es.wikipedia.org/wiki/Trigger_(base_de_datos)

[7] http://sql.11sql.com/sql-not-null.htm

[8] https://jorgesanchez.net/manuales/gbd/entidad-relacion.html#h25 apartado 1.5.2

[9] https://dev.mysql.com/doc/refman/8.0/en/datetime.html

[10] https://dev.mysql.com/doc/refman/8.0/en/update.html

[11] https://www.aemps.gob.es/industria/dispositivos_seguridad/FAQs/home.htm

[12] https://es.wikipedia.org/wiki/Posolog%C3%ADa

[13] https://es.wikipedia.org/wiki/Sistema_de_gesti%C3%B3n_de_bases_de_datos

[14] https://www.postgresql.org/docs/11/sql-createprocedure.html