CODER HOUSE

Curso: Lenguaje SQL

Proyecto: Confección de una BASE DE DATOS en SQL para una Clínica

Alumno: Víctor Facundo Espíndola

Docente: Sebastian Andres Quezadada

Tutor: Yoelys Figueredo Padrón

Contenido

Contenido

1. Introducción

El presente informe corresponde al Proyecto Final del curso de SQL dictado en CoderHouse. En el mismo se desarrollará la confección de una base de datos de una centro médico, esto es un factor fundamental para la rapidez con la cual se accede a la información y la posibilidad de recopilar datos clínicos y gestionar la información de los paciente, doctores, consultorios y turnos disponibles.

2. Objetivo

El objetivo del proyecto final será la confección de una base de datos relacional en SQL para un centro médico.

3. Modelo de negocio

Se ha elegido este este proyecto ya que sabíamos que en un centro médico hay una gran cantidad de información que debe ser manipulada de forma eficiente y veloz para generar consultas a pacientes en diversas especialidades y sus respectivos doctores. Por eso con esta base de datos se dará una mayor facilidad en esa búsqueda, agilizará cualquier consulta y pondrá a disposición todos datos de los doctores, consultorio, pacientes e historiales médicos.

4. Descripción de Tablas

A continuación, se muestran las tablas que serán desarrolladas luego en SQL, con una breve introducción de los datos que se alojarán en ellas.

Tabla: ***Consultas***

Descripción: esta tabla contiene información de casi todas las tablas creadas, ya que recopila datos del paciente, doctor, consultorio e historial clínico.

Field	Type	Null	Key	Default	Extra
idConsultas	int	NO	PRI	NULL	auto_increment
fecha	date	NO		NULL	
receta	varchar(255)	NO		NULL	
idDoctor	int	NO	MUL	NULL	
idPacientes	int	NO	MUL	NULL	
idConsultorio	int	NO	MUL	NULL	
idHClinico	int	NO	MUL	NULL	

Tabla: ***Doctor***

Descripción: esta tabla contiene información estratégica de cada doctor como su matrícula, especialidad y pacientes asignados.

	Field	Type	Null	Key	Default	Extra
▶	idDoctor	int	NO	PRI	NULL	auto_increment
	nombre	varchar(255)	NO		NULL	
	apellido	varchar(255)	NO		NULL	
	fechaNacimiento	date	NO		NULL	
	matricula	varchar(255)	NO		NULL	
	dni	int	NO		NULL	
	idEspecialidad	int	NO	MUL	NULL	
	idPacientes	int	NO	MUL	NULL	

Tabla: ***Pacientes***

Descripción: esta tabla contiene información estratégica de cada uno de los pacientes que tiene el centro médico. Como datos personales, datos de contacto e historial clínico.

	Field	Type	Null	Key	Default	Extra
▶	idPacientes	int	NO	PRI	NULL	auto_increment
	nombre	varchar(255)	NO		NULL	
	apellido	varchar(255)	NO		NULL	
	genero	varchar(1)	NO		NULL	
	dni	int	NO		NULL	
	idCPacientes	int	NO	MUL	NULL	
	idHClinico	int	NO	MUL	NULL	

Tabla: ***Contacto Pacientes***

Descripción: esta tabla contiene información de contacto de cada uno de los pacientes, como el email y teléfono.

	Field	Type	Null	Key	Default	Extra
▶	idCPacientes	int	NO	PRI	NULL	auto_increment
	email	varchar(255)	YES	UNI	NULL	
	telefono	varchar(255)	YES		NULL	

Tabla: ***Consultorio***

Descripción: esta tabla contiene información de contacto y dirección de cada consultorio del centro medico.

	Field	Type	Null	Key	Default	Extra
▶	idConsultorio	int	NO	PRI	NULL	auto_increment
	direccion	varchar(255)	YES		NULL	
	cp	int	NO		NULL	
	telefono	varchar(100)	NO		NULL	
	mail	varchar(200)	NO		NULL	

Tabla: ***Historial Clínico***

Descripción: esta tabla contiene información del historial clínico de cada paciente.

Field	Type	Null	Key	Default	Extra
idHClinico	int	NO	PRI	NULL	auto_increment
edad	int	NO		NULL	
altura	int	NO		NULL	
peso	int	NO		NULL	
tipo_sangre	varchar(20)	YES		NULL	
enfermedades	varchar(255)	YES		NULL	

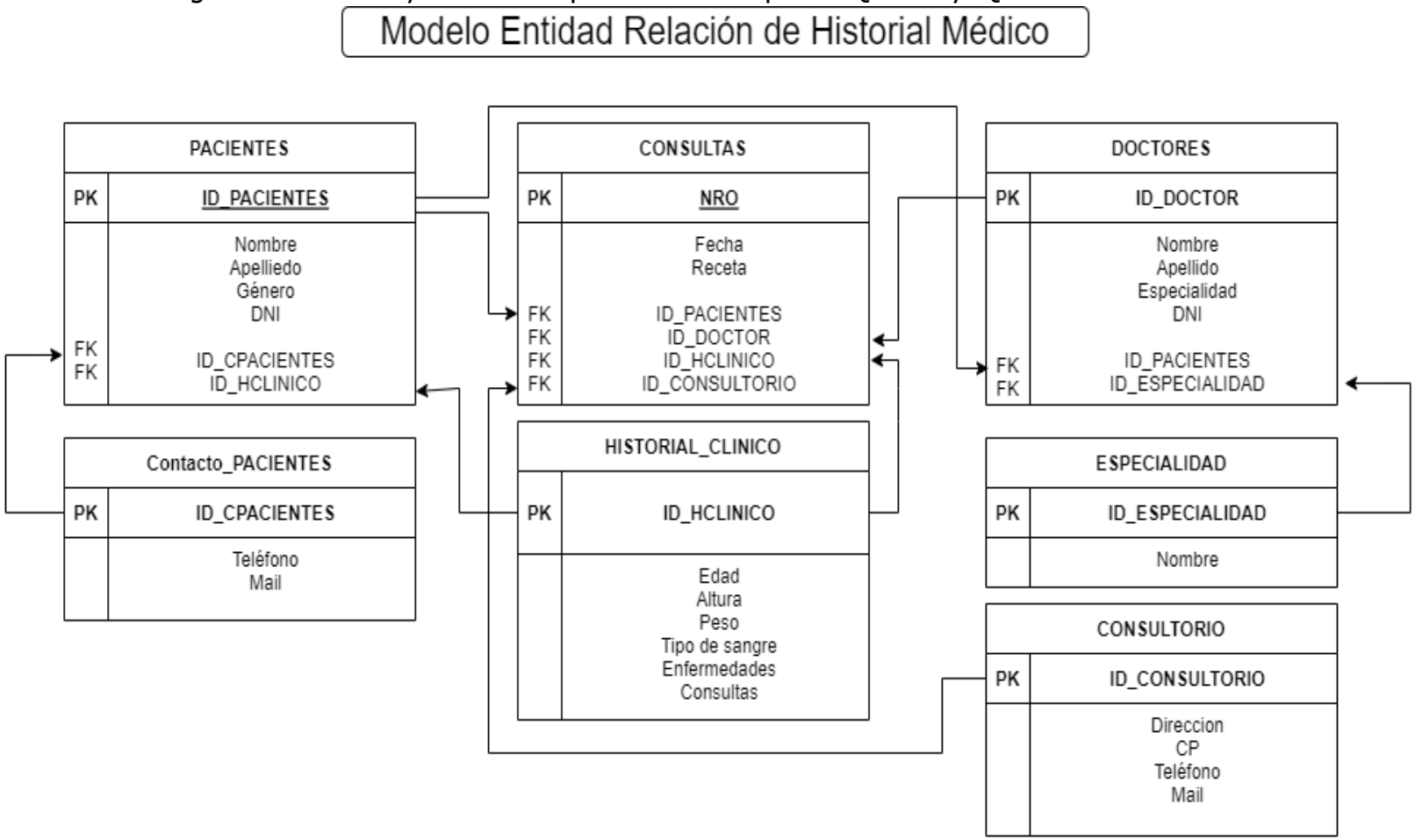
Tabla: ***Especialidad***

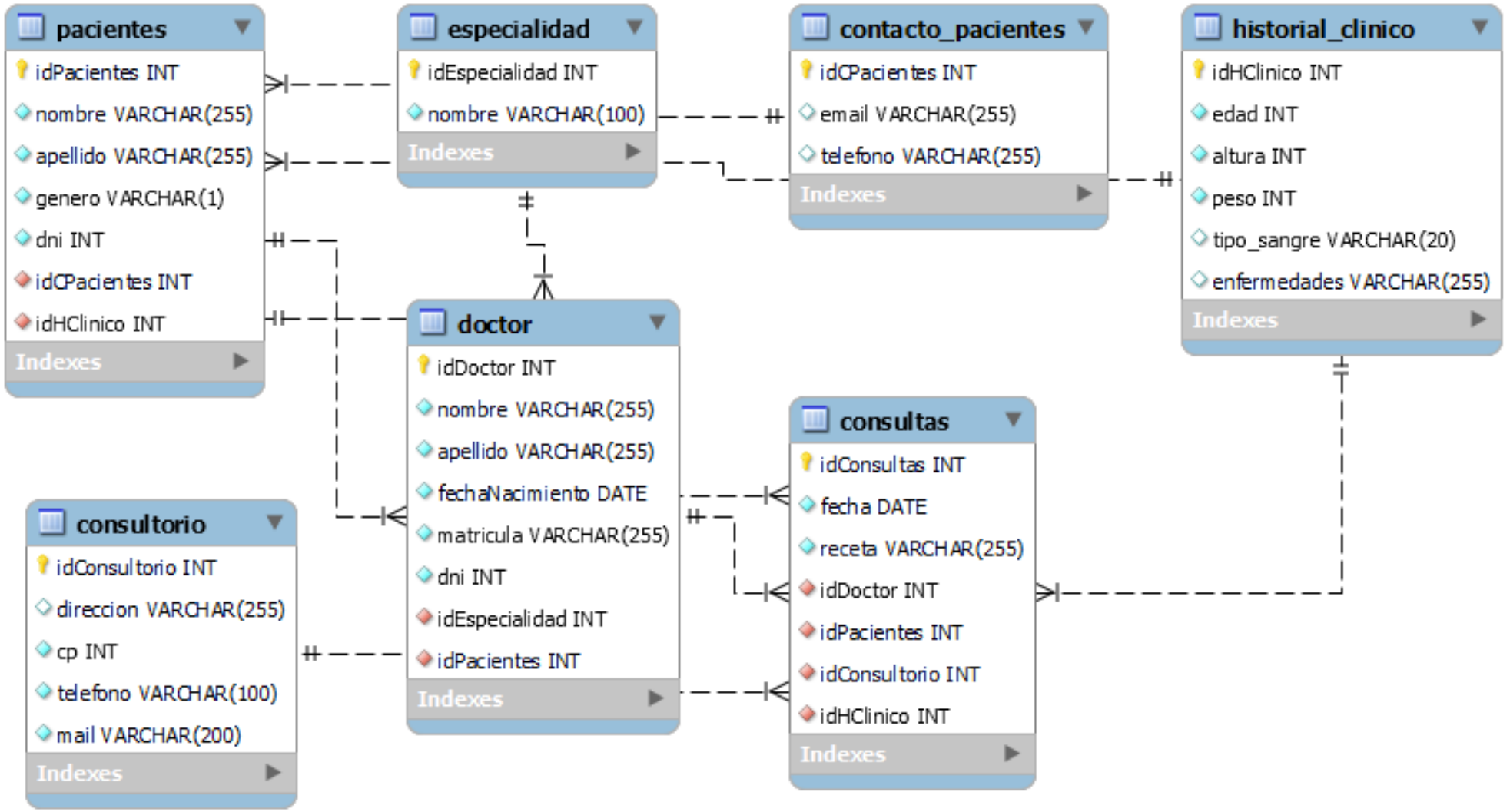
Descripción: esta tabla contiene información de las distintas especialidades médicas del centro medico.

Field	Type	Null	Key	Default	Extra
idEspecialidad	int	NO	PRI	NULL	auto_increment
nombre	varchar(100)	NO		NULL	

5. Diagrama Entidad – Relación

A continuación, se muestran dos imágenes del diagrama de Entidad – Relación del proyecto. La primera imagen fue realizada en la web diagrams.net y se realizó al comienzo del curso (luego se realizaron modificaciones sobre los campos de las tablas). La segunda imagen contiene el Diagrama E – R final y se realizó a partir de un Script en SQL en MySQL Workbench.




pacientes
idPacientes INT
nombre VARCHAR(255)
apellido VARCHAR(255)
genero VARCHAR(1)
dni INT
idCPacientes INT
idHClinico INT
Indexes
especialidad
idEspecialidad INT
nombre VARCHAR(100)
Indexes
contacto_pacientes
idCPacientes INT
email VARCHAR(255)
telefono VARCHAR(255)
Indexes
historial_clinico
idHClinico INT
edad INT
altura INT
peso INT
tipo_sangre VARCHAR(20)
enfermedades VARCHAR(255)
Indexes
doctor
idDoctor INT
nombre VARCHAR(255)
apellido VARCHAR(255)
fechaNacimiento DATE
matricula VARCHAR(255)
dni INT
idEspecialidad INT
idPacientes INT
Indexes
consultas
idConsultas INT
fecha DATE
receta VARCHAR(255)
idDoctor INT
idPacientes INT
idConsultorio INT
idHClinico INT
Indexes
consultorio
idConsultorio INT
direccion VARCHAR(255)
cp INT
telefono VARCHAR(100)
mail VARCHAR(200)
Indexes

6. Creación de tablas en SQL

En este apartado se procedió a la creación de las tablas del proyecto. En estecaso fueron 7 tablas. Se adjunta en el documento un Script SQL con la creaciónde estas.

```sql
-- CREACION DE LA BASE DE DATOS

CREATE DATABASE IF NOT EXISTS proyecto_HISTORIAL_medico;
USE proyecto_HISTORIAL_medico;
-- CREACION DE LAS ENTIDADES


CREATE TABLE IF NOT EXISTS especialidad(
idEspecialidad INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
nombre VARCHAR(100) NOT NULL
);

CREATE TABLE IF NOT EXISTS historial_clinico(
idHClinico INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
edad INT NOT NULL,
altura INT NOT NULL,
peso INT NOT NULL,
tipo_sangre VARCHAR (20),
enfermedades VARCHAR (255)
);

CREATE TABLE IF NOT EXISTS consultorio(
idConsultorio INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
direccion VARCHAR (255),
cp INT NOT NULL,
telefono VARCHAR (100) NOT NULL,
mail VARCHAR (200) NOT NULL
);

CREATE TABLE IF NOT EXISTS contacto_pacientes(
idCPacientes INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
email VARCHAR(255) UNIQUE,
telefono VARCHAR(255)
);
```

```sql
CREATE TABLE IF NOT EXISTS pacientes(
idPacientes INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
nombre VARCHAR(255) NOT NULL,
apellido VARCHAR(255) NOT NULL,
genero VARCHAR (1) NOT NULL,
dni INT NOT NULL,
idCPacientes INT NOT NULL,
idHClinico INT NOT NULL,
FOREIGN KEY (idCPacientes) REFERENCES contacto_pacientes(idCPacientes),
FOREIGN KEY (idHClinico) REFERENCES historial_clinico(idHClinico)
);

CREATE TABLE IF NOT EXISTS doctor(
idDoctor INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
nombre VARCHAR(255) NOT NULL,
apellido VARCHAR(255) NOT NULL,
fechaNacimiento DATE NOT NULL,
matricula VARCHAR(255) NOT NULL,
dni INT NOT NULL,
idEspecialidad INT NOT NULL,
idPacientes INT NOT NULL,
FOREIGN KEY (idEspecialidad) REFERENCES especialidad(idEspecialidad),
FOREIGN KEY (idPacientes) REFERENCES pacientes(idPacientes)
);

CREATE TABLE IF NOT EXISTS consultas(
idConsultas INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
fecha DATE NOT NULL,
receta VARCHAR(255) NOT NULL,
idDoctor INT NOT NULL,
idPacientes INT NOT NULL,
idConsultorio INT NOT NULL,
idHClinico INT NOT NULL,
FOREIGN KEY (idDoctor) REFERENCES doctor(idDoctor),
FOREIGN KEY (idPacientes) REFERENCES pacientes(idPacientes),
FOREIGN KEY (idConsultorio) REFERENCES consultorio(idConsultorio),
FOREIGN KEY (idHClinico) REFERENCES historial_clinico(idHClinico)
);
```

7. Inserción de Datos

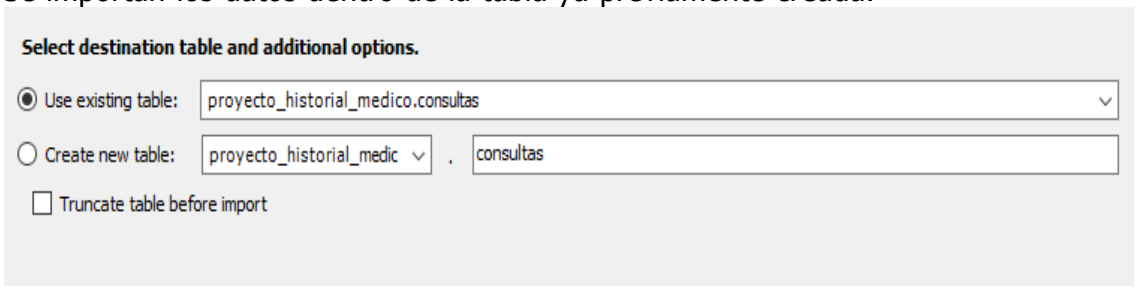
Se procedió a realizar la inserción de datos dentro de las tablas mediante importación de archivos CSV. Se adjunta en el documento un Script SQL con la inserción de los registros en las tablas.

Los datos que se incorporan al proyecto corresponden a datos de pacientes, doctores, consultorios, especialidades, historiales clínicos, consultas, etc

El procesoque se realizó fue el siguiente:

1. Creación de la tabla en SQL
2. En MySQL se ejectuta: SELECT * FROM tabla_a_importar;
3. En el Result Grid se selecciona la opción de importar datos y se elige la rutadonde se encuentra el archivo.

4. Se importan los datos dentro de la tabla ya previamente creada.



5. Se verifica que los tipos de datos de la tabla coincidan con los datosimportados.
6. Se realiza la importación.

8. Vistas en SQL

Una vista es una tabla virtual que se genera a partir de la ejecución de una o másconsultas SQL, aplicada sobre una o más tablas.

En este apartado se realizaron diversas Vistas de consultas a la base de datos delproyecto. Se adjunta en el documento un Script SQL con las siguientes vistas:

- Vista Listado_Pacientes: Trae los datos básicos de cada paciente, la vista está ordenada por Apellido

```
create view Listado_Pacientes as
select CONCAT_WS(' ', p.apellido, p.nombre) as NyA, genero, dni
from pacientes as p
order by p.apellido;
```

- Vista Contacto_de_Pacientes: Trae los datos de contacto de cada paciente, la vista está ordenada por Apellido

```
create view Contacto_de_Pacientes as
select CONCAT_WS(' ', p.apellido, p.nombre) as Nombre_Apellido, c_p.email
as Correo, c_p.telefono as Tel
from pacientes as p
join contacto_pacientes as c_p ON (p.idPacientes = c_p.idCPacientes)
order by p.apellido;
```

- Vista Datos_Clinicos_de_Pacientes: Trae los datos de clínicos de cada paciente, la vista está ordenada por Apellido

```
create view Datos_Clinicos_de_Pacientes as
select CONCAT_WS(' ', p.apellido, p.nombre) as Nombre_Apellido,
hc_p.edad as Edad ,hc_p.altura as Altura ,hc_p.peso ,hc_p.tipo_sangre
as Sangre ,hc_p.enfermedades as Enfermedades
from pacientes as p
join historial_clinico as hc_p ON (p.idHClinico = hc_p.idHClinico)
order by p.apellido;
```

- Vista Consultas_de_Pacientes: Trae los datos de las consultas que tuvo cada paciente, la vista está ordenada por Numero de Consulta

```
create view Consultas_de_Pacientes as
select c.idConsultas as Numero_De_Gestion, CONCAT_WS(' ', p.nombre, p.apellido)
as Nombre_De_Paciente, c.fecha as Fecha, c.receta as Receta,
CONCAT_WS(' ', d.nombre, d.apellido) as Doctor, a.direccion as Lugar
from consultas as c
join pacientes as p ON (c.idPacientes = p.idPacientes)
join doctor as d ON (c.idDoctor = d.idDoctor)
join consultorio as a  ON (c.idConsultorio = a.idConsultorio)
order by idConsultas;
```

- Vista Listado de Doctores: Trae los datos de todos los doctores, la vista está ordenada por Apellido

```
create view Listado_Doctores as
select CONCAT_WS(' ', d.apellido, d.nombre) as Nombre_De_Doctor,
 d.fechaNacimiento as Fecha_De_Nacimiento, d.matricula as Matricula,
 e.nombre as Especialidad, CONCAT_WS(' ', p.nombre, p.apellido) as Pacientes
from doctor as d
join especialidad as e ON (d.idEspecialidad = e.idEspecialidad)
join pacientes as p  ON (d.idPacientes = p.idPacientes)
order by d.apellido ;
```

- Vista Listado de Especialidades: Trae los datos de todas las especialidades de la Clinicia, la vista está ordenada por Nombre de Especialidad

```
create view Listado_Especialidades as
select e.nombre as Especialidad , CONCAT_WS(' ', d.apellido, d.nombre)
as Nombre_De_Doctor
from especialidad as e
join doctor as d ON (d.idEspecialidad = e.idEspecialidad)
order by e.nombre;
```

9. Funciones en SQL

Las funciones personalizadas o almacenadas de Mysql permiten procesar y manipular datos de forma procedural y eficiente. Dichos datos son enviados a travésde uno o más parámetros, al momento de invocar la función, y devueltos como resultado por esta misma.

Se adjunta en el documento un Script SQL con las siguientes funciones.

- Función contador_tiposdesangre: Con ésta función podemos hacer un recuento rápido del total de pacientes que tienen cada uno de los 4 tipos de sangre (A, B, AB, O)

```
DELIMITER //
CREATE FUNCTION contador_tiposdesangre (tipo VARCHAR(3)) RETURNS int(2)
deterministic
  BEGIN
    return (SELECT COUNT(*) FROM historial_clinico WHERE tipo_sangre = tipo);
  END;//
```

- Función Calculo_del_IMC: Con ésta función podemos calcular el índice de masa corporal de cada paciente, indicando que tipo de Peso tiene (Inferior, Normal, superior, obesidad)

```
DELIMITER //
CREATE FUNCTION Calculo_del_IMC (id_Paciente int (3)) RETURNS VARCHAR(60)
deterministic
  BEGIN
    declare IMC INT(2);
    declare p int (3);
    declare h int (3);
    declare mensaje varchar(60);
    set p = ( SELECT peso from historial_clinico WHERE idHClinico = id_Paciente );
    set h = ( SELECT altura from historial_clinico WHERE idHClinico = id_Paciente );
    set IMC = p/(POWER(2,h/100));
    case
        WHEN IMC>=28 then
            set mensaje = "El paciente tiene OBESIDAD";
        when IMC<28 AND IMC>=23 then
            set mensaje = "El paciente tiene Peso Superior al Normal";
        WHEN IMC<23 AND IMC>=19 then
            set mensaje = "El paciente tiene Peso Normal";
        WHEN IMC<19 then
            set mensaje = "El paciente tiene Peso Inferior al Normal";
    end case;
    return (mensaje);
  END;//
```

10. Procedimientos almacenados en SQL

Un Procedimiento Almacenado o Stored Procedure es un programa almacenado físicamente en una base de datos, creado para cumplir tareas específicas. Permitetambién establecer niveles de seguridad y manipular operaciones complejas o extensas del lado del servidor, evitando un ida y vuelta de datos que termine sobrecargando una red o servidor.

Se adjunta en el documento un Script SQL con los siguientes Procedimiento Almacenados.

Procedimiento Almacenado 1: Procedimiento almacenado que CARGA NUEVOS PACIENTES

```
DROP PROCEDURE IF EXISTS `create_paciente`
DELIMITER //
CREATE PROCEDURE `create_paciente` (
    IN id INT, IN nombre_new VARCHAR(255), IN apellido_new VARCHAR(255),
    IN genero_new VARCHAR(1), IN dni_new INT, IN prepaga_new VARCHAR(255),
    IN email_new VARCHAR(255), IN telefono_new VARCHAR(255), IN edad_new INT,
    IN altura_new INT, IN peso_new INT, IN tipo_sangre_new VARCHAR (20),
    IN enfermedades_new VARCHAR (255))
    BEGIN
        DECLARE existe_paciente INT;
        SET existe_paciente = (SELECT COUNT(*) FROM pacientes WHERE dni=dni_new);
        IF existe_paciente = 0 THEN
            SET FOREIGN_KEY_CHECKS=0;
            INSERT INTO pacientes
            VALUES (id,nombre_new,apellido_new ,genero_new ,dni_new ,prepaga_new,id, id );
            INSERT INTO contacto_pacientes
            VALUES (id,email_new,telefono_new);
            INSERT INTO historial_clinico
            VALUES (id,edad_new,altura_new,peso_new,tipo_sangre_new,enfermedades_new);
        END IF;
END//
```

Procedimiento Almacenado 2: Procedimiento almacenado que actualiza la prepaga del paciente

```
DROP PROCEDURE IF EXISTS `update_prepaga`
DELIMITER //
CREATE PROCEDURE `update_prepaga` (IN id_paciente INT,IN nombre_prepaga CHAR(20))
BEGIN
   update pacientes set prepaga = nombre_prepaga where idPacientes = id_paciente;
END//
```

Procedimiento Almacenado 3: Procedimiento almacenado que actualiza la los datos del paciente

```
DROP PROCEDURE IF EXISTS `update_paciente`
DELIMITER //
CREATE PROCEDURE `update_paciente` (IN id_paciente INT, IN nombre_nuevo VARCHAR(255),
IN apellido_nuevo VARCHAR(255), IN genero_nuevo VARCHAR (1), IN nombre_prepaga VARCHAR(255))
BEGIN
   IF nombre_nuevo != '' THEN
   update pacientes set nombre = nombre_nuevo where idPacientes = id_paciente;
   END IF;
   IF apellido_nuevo != '' THEN
   update pacientes set apellido = apellido_nuevo where idPacientes = id_paciente;
   END IF;
   IF genero_nuevo != '' THEN
   update pacientes set genero = genero_nuevo where idPacientes = id_paciente;
   END IF;
   IF nombre_prepaga != '' THEN
   update pacientes set prepaga = nombre_prepaga where idPacientes = id_paciente;
   END IF;
END//
```

11. Triggers en SQL

Un Trigger puede definirse como una aplicación o programa almacenado en el servidor (de base de datos) creado para ejecutarse de forma automática, cuandouno o más eventos específicos ocurren en la base de datos.

Se adjunta en el documento un Script SQL con los siguientes Triggers.

- Trigger 1: Deja el registro de la creación de un paciente nuevo en la Bitácora

```
DELIMITER //
CREATE trigger `bitacora_insert`
after insert on pacientes
for each row
BEGIN
   insert into bitacora (accion, fecha, id_insertado, usuario)
   values ('create', now(), new.idPacientes , SYSTEM_USER());
END//
```

Trigger 2: Deja el registro de la actualización de un paciente en la Bitácora

```
DELIMITER //
CREATE trigger `bitacora_update`
after update on pacientes
for each row
BEGIN
   insert into bitacora (accion, fecha, id_insertado, usuario)
   values ('update', now(), new.idPacientes , SYSTEM_USER());
END//
```

Trigger 3: Deja el registro de la eliminación de un paciente en la Bitácora

```
DELIMITER //
CREATE trigger `bitacora_delete`
before delete on pacientes
for each row
BEGIN
   insert into bitacora (accion, fecha, id_insertado, usuario)
   values ('delete', now(), old.idPacientes , SYSTEM_USER());
END//
```

Tabla Bitácora, sirve de registro de acciones (créate, update o delete) en la tabla Pacientes

```
DROP TABLE IF EXISTS `bitacora`;
CREATE TABLE IF NOT EXISTS `bitacora`(
idB int auto_increment not null primary key,
accion varchar (30),
fecha datetime,
id_insertado int,
usuario varchar(30)
);
```