

C.2 Laboratorio Semana 2- Angel Gabriel Ortega

Siguiendo el caso de uso de un Hospital, crear las tablas necesarias y realizar lo siguiente

Trabajando con vistas

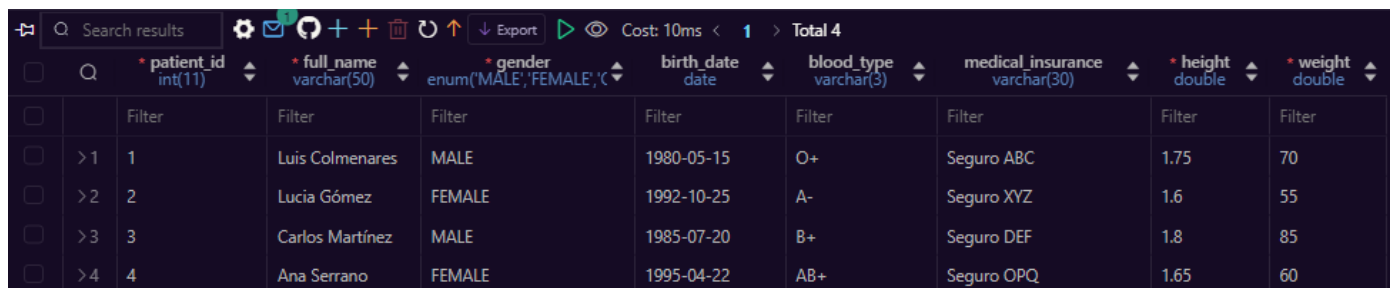
- Creación de vistas utilizando diferentes tipos de comandos

Como sabemos tenemos bastantes maneras de crear vistas dependiendo de como deseemos visualizar los datos, desde crear vistas de manera sencilla con solo hacer un **SELECT**, a continuación algunos ejemplos para poder crear vistas

Ejemplos

```
CREATE VIEW vista_paciente
AS
SELECT * FROM patient
```

Resultado



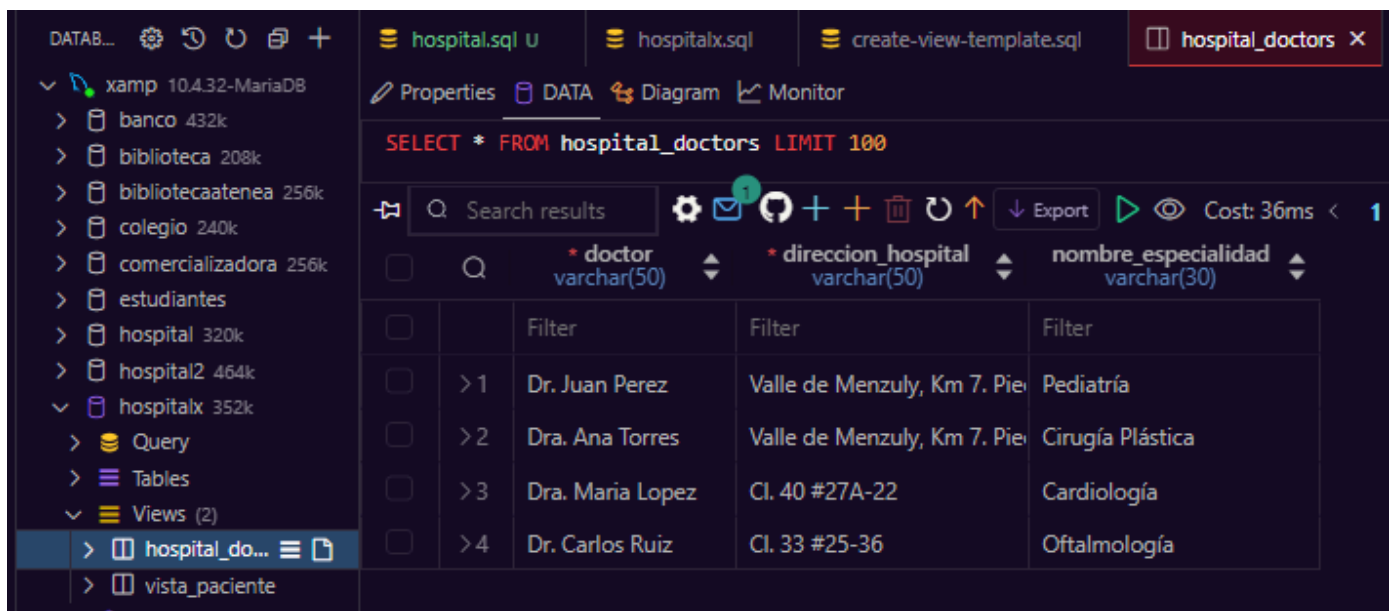
The screenshot shows a database query result with 10 columns: patient_id, full_name, gender, birth_date, blood_type, medical_insurance, height, and weight. The results are displayed in a table with 4 rows of data.

	patient_id int(11)	full_name varchar(50)	gender enum('MALE','FEMALE','C')	birth_date date	blood_type varchar(3)	medical_insurance varchar(30)	height double	weight double
>1	1	Luis Colmenares	MALE	1980-05-15	O+	Seguro ABC	1.75	70
>2	2	Lucia Gómez	FEMALE	1992-10-25	A-	Seguro XYZ	1.6	55
>3	3	Carlos Martínez	MALE	1985-07-20	B+	Seguro DEF	1.8	85
>4	4	Ana Serrano	FEMALE	1995-04-22	AB+	Seguro OPQ	1.65	60

pueden ser tan sencillas como la anterior, en otros casos podemos modificarlo para poder obtener resultados y combinarlos en mas de una vista.

```
Create View hospital_doctors2
AS
SELECT d.full_name AS doctor, h.address_hospital AS direccion_hospital,
f.name_speciality AS nombre_especialidad
FROM doctor AS d
INNER JOIN hospitals AS h ON d.hospital_id = h.id_hospital
INNER JOIN focus AS f ON d.id_speciality = f.id_speciality;
```

aquí realizamos un join para poder unificar varias columnas de nuestra preferencia.



con esto ya podemos ver el poder de las vistas, almacenar alguna consulta de nuestra preferencia, solo debemos añadir "CREATE VIEW" y el nombre que queramos para poder guardar alguna consulta de nuestra preferencia.

- *Modificación de vistas*

en este caso para poder modificar una vista podemos hacerlo usando la sentencia **CREATE OR REPLACE**

Ejemplo

```
CREATE OR REPLACE VIEW vista_paciente AS
SELECT pa.* , ap.appointment_date, ap.reason, ap.status
FROM patient AS pa
INNER JOIN appointments AS ap ON pa.patient_id= ap.id_patient
```

Resultado

The screenshot shows the same database management interface, but now the query is `SELECT * FROM vista_paciente LIMIT 100`. The results table has more columns: `patient_id`, `full_name`, `gender`, `birth_date`, `blood_type`, `medical_insurance`, `height`, `weight`, and `appointment_date`. It shows four rows of patient data.

	patient_id	full_name	gender	birth_date	blood_type	medical_insurance	height	weight	appointment_date
> 1	2	Lucia Gómez	FEMALE	1992-10-25	A-	Seguro XYZ	1.6	55	2024-09-15 11:30:00
> 2	3	Carlos Martínez	MALE	1985-07-20	B+	Seguro DEF	1.8	85	2024-09-20 09:00:00
> 3	4	Ana Serrano	FEMALE	1995-04-22	AB+	Seguro OPQ	1.65	60	2024-10-01 15:00:00
> 4	4	Ana Serrano	FEMALE	1995-04-22	AB+	Seguro OPQ	1.65	60	2024-10-01 15:00:00

- *Actualización de vistas utilizando tablas*

ahora para actualizar haciendo uso de tablas necesitamos que debe estar ligada a solo una tabla y no contener cosas complejas como joins, entonces modificaremos una vista basica llamada: **PACIENTE BASICO**

Ejemplos

```
CREATE VIEW paciente_basico AS
SELECT * FROM patient
```

Antes

	* patient_id int(11)	* full_name varchar(50)	* gender enum('MALE','FEMALE','C')	birth_date date	blood_type varchar(3)	medical_insurance varchar(30)	* height double	* weight double
>1	1	Luis Colmenares	MALE	1980-05-15	O+	Seguro ABC	1.75	70
>2	2	Lucia Gómez	FEMALE	1992-10-25	A-	Seguro XYZ	1.6	55
>3	3	Carlos Martínez	MALE	1985-07-20	B+	Seguro DEF	1.8	85
>4	4	Ana Serrano	FEMALE	1995-04-22	AB+	Seguro OPQ	1.65	60

Despues

```
UPDATE paciente_basico AS pa
SET pa.weight = 75
WHERE pa.patient_id = 1;
```

	* patient_id int(11)	* full_name varchar(50)	* gender enum('MALE','FEMALE','C')	birth_date date	blood_type varchar(3)	medical_insurance varchar(30)	* height double	* weight double
>1	1	Luis Colmenares	MALE	1980-05-15	O+	Seguro ABC	1.75	75
>2	2	Lucia Gómez	FEMALE	1992-10-25	A-	Seguro XYZ	1.6	55
>3	3	Carlos Martínez	MALE	1985-07-20	B+	Seguro DEF	1.8	85
>4	4	Ana Serrano	FEMALE	1995-04-22	AB+	Seguro OPQ	1.65	60

- *Eliminación de vistas*

finalmente para la eliminacion de una vista de nuestra preferencia puede ser tan sencillo como simplemente realizar un: **DROP VIEW** y el nombre de la vista requerida

Ejemplo

```
DROP VIEW paciente_basico;
```

```
--eliminacion
Run | New Tab
DROP VIEW paciente_basico; 10ms
```

- *Vistas vs Tablas Temporales (Investigacion)*

RTA: ambas segun investigue son muy importantes dependiendo de los requerimientos

1. tablas temporales

- Las tablas temporales son tablas que se crean durante la sesión de base de datos y se eliminan automáticamente al final de la sesión o cuando se cierra la conexión. Se crean usando el comando `CREATE TEMPORARY TABLE` casi siempre segun lei se crean para poder capturar algun requerimiento de suma importancia para luego realizar alguna operacion para poder llevarlo a una tabla real y poder cumplir con la persistencia que a la final es el objetivo de todas las apps.

```
CREATE TEMPORARY TABLE tablatemp (  
    atributos  
    atributos  
);
```

2. Vistas

- pero por otro lado tenemos las vistas que solo ayudan a almacenar una consulta o una accion o tambien se pueden llamar stored queries y pueden ayudarnos cuando no queremos volver a ejecutar el querie completo y de una manera mas sencilla tengamos el estado actual de una tabla o querie de nuestra preferencia.

```
CREATE VIEW vistaaa AS  
SELECT * FROM tablaaa
```

la verdad dar una opinion de cual seria mejor o mas adecuado dependeria mucho la situacion o lo que queramos lograr, pero honestamente veo las vistas mas del lado de analisis de datos, creo que en el desarrollo real tal vez sea mas probable hacer uso de las tablas temporales antes que de las vistas. Segun mi opinion personal.

Trabajando con procedimientos almacenados

- *Creación de procedimientos almacenados para consultar datos de una tabla*

empezando con la creacion de procedimientos almacenados, como vimos en clase este contiene varias instrucciones dentro de una sola tarea, dandonos la capacidad de llamarlos cuando necesitemos.

Ejemplo

```
CREATE PROCEDURE Pacienteby_id( IN id INT )  
BEGIN  
    SELECT * FROM patient WHERE patient_id = id;  
END;
```

luego de lo llamamos

```
CALL `Pacienteby_id`(1);
```

Resultado

Q	patient_id int	full_name varchar	gender string	birth_date date	blood_type varchar	medical_insurance varchar	height double	weight double
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
>1	1	Luis Colmenares	MALE	1980-05-15	O+	Seguro ABC	1.75	75

- Creación de procedimientos almacenados para actualizar datos en una tabla

los stored procedures son bastante utiles y poco a poco nos damos cuenta porque siguen siendo usados en apps de antaño que niegan actualizarse, pero es innegable su gran utilidad, no se restringen solo a selects, podemos realizar acciones crud con total libertad.

Ejemplo

```
CREATE PROCEDURE actualizar_peso_paciente (IN paciente_id INT, IN nuevo_peso  
DECIMAL(5,2))  
BEGIN  
    UPDATE patient  
    SET weight = nuevo_peso  
    WHERE patient_id = paciente_id;  
END;
```

aqui estariamos actualizando el peso de un paciente especifico basado en su id con el procedimiento almacenado nos ahorramos realizar updates continuos y solo tenemos que hacer el call y pasar el id que queramos y su nuevo peso.

```
CALL actualizar_peso_paciente(1, 90);
```

Resultado

	Q	patient_id int(11)	full_name varchar(50)	gender enum('MALE','FEMALE','C')	birth_date date	blood_type varchar(3)	medical_insurance varchar(30)	height double	weight double
		Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
	>1	1	Luis Colmenares	MALE	1980-05-15	O+	Seguro ABC	1.75	90
	>2	2	Lucía Gómez	FEMALE	1992-10-25	A-	Seguro XYZ	1.6	55
	>3	3	Carlos Martínez	MALE	1985-07-20	B+	Seguro DEF	1.8	85
	>4	4	Ana Serrano	FEMALE	1995-04-22	AB+	Seguro OPQ	1.65	60

- Manejo de excepciones en procedimientos almacenados (Investigacion)

podemos denotar que los stored procedures tienen muchas ventajas y nos facilitan multiples acciones que podrian verse complejas y juntarlo todo en una unica peticion o sentencia. Podriamos modificar la sentencia anterior que actualizaba el precio para añadirles condicionales o exepciones

Ejemplo

por lo general cuando manejamos excepciones en procedimientos almacenados siempre hacemos uso de una herramienta llamada excepciones y de nuestro rollback por si en dado caso algo no salga como se espera, podamos devolver un mensaje adecuado y no aplicar ningun cambio que se alcance a realizar durante la ejecucion de dicho procedimiento, en este caso me apoye del mismo ejemplo del peso, pero paso a paso vamos añadiendole mas y mas dificultad haciendolo mucho mas robusto.

```
CREATE PROCEDURE actualizar_peso_paciente (IN paciente_id INT, IN nuevo_peso
DECIMAL(5,2))
BEGIN
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'paso un error, no se guardaron los cambios';
    END;

    START TRANSACTION;

    IF nuevo_peso > 100 THEN
        SELECT 'el peso no puede ser mayor a 100, necesitaras un hospital
mas grande';
    END IF;

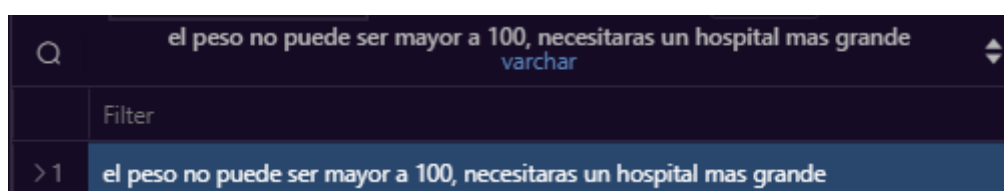
    IF nuevo_peso < 40 THEN
        SELECT 'el peso no puede ser menor a 40, necesitaras un tratamiento
para subir el peso';
    END IF;

    UPDATE patient
    SET weight = nuevo_peso
    WHERE patient_id = paciente_id;

    COMMIT;
END;
```

ahora si hacemos un call ignorando la excepcion y pasamos un peso mayor a 100 veremos lo siguiente:

```
CALL actualizar_peso_paciente(1, 190);
```



o en otros casos donde pasemos menor a 40 nos devolvera la otra excepcion:

Q	el peso no puede ser menor a 40, necesitaras un tratamiento para subir el pes varchar
	Filter
> 1	el peso no puede ser menor a 40, necesitaras un tratamiento para subir el peso

- Consulta a una vista utilizando procedimientos almacenados

segun lo visto, no solamente podemos usar los store procedures para manejar tablas sino a su vez nos ayudan a poder crearlos en base a una vista de nuestra preferencia.

Ejemplo

creamos la vista

```
CREATE VIEW citas_pacientes AS
SELECT pa.patient_id, pa.full_name, pa.blood_type, app.appointment_date,
app.reason FROM patient AS pa
INNER JOIN appointments AS app ON pa.patient_id = app.id_patient
```

luego podemos hacer uso del stored procedure en base a esta vista

```
CREATE PROCEDURE citas_pacienteby_id(IN id INT)
BEGIN
    SELECT * FROM citas_pacientes AS pa
    WHERE pa.patient_id = id;
END;
```

finalmente solo aplicamos un call como siempre y tendremos la respuesta esperada.

```
CALL citas_pacienteby_id(3);
```

Resultado

Q	patient_id int	full_name varchar	blood_type varchar	appointment_date datetime	reason varchar
	Filter	Filter	Filter	Filter	Filter
> 1	3	Carlos Martínez	B+	2024-09-20 09:00:00	Examen de la vista

- Procedimientos Almacenados vs Funciones (investigacion)

para dar respuesta a esto debemos conocer muy bien los stored procedures, estos son muy similares, pero las funciones estan mas orientadas a realizar una unica tarea, a diferencia de los procedimientos almacenados que realizan multiples procesos, cuando sumamos o restamos valores entre ellos. Ademas Siempre tendra un parametro return

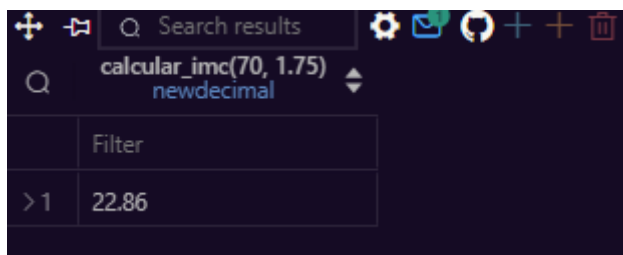
Ejemplos funciones

```
CREATE FUNCTION calcular_imc(peso DOUBLE, altura DOUBLE) RETURNS DOUBLE
DETERMINISTIC
BEGIN
    DECLARE imc DOUBLE;
    SET imc = peso / (altura * altura);
    RETURN imc;
END;
```

y a diferencia de los procedimientos almacenados o como diríamos en programación podemos invocarla usando select

```
SELECT calcular_imc(70, 1.75);
```

Resultado



The screenshot shows a SQL query result in a dark-themed interface. The query is `calcular_imc(70, 1.75)` and the result is `newdecimal`. The result is displayed in a table with one row and one column, showing the value `22.86`.

Filter
>1 22.86

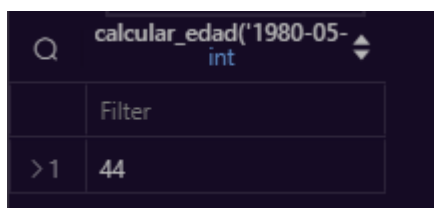
otra idea que me propuse hacer fue calcular la edad en base a la fecha de nacimiento

```
CREATE FUNCTION calcular_edad(fecha_nacimiento DATE)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE edad INT;
    SET edad = TIMESTAMPDIFF(YEAR, fecha_nacimiento, CURDATE());
    RETURN edad;
END;
```

y cuando hagamos el select pasamos un formato de fecha esperado

```
SELECT calcular_edad('1980-05-15');
```

Resultado



The screenshot shows a SQL query result in a dark-themed interface. The query is `calcular_edad('1980-05-15')` and the result is `int`. The result is displayed in a table with one row and one column, showing the value `44`.

Filter
>1 44

Trabajando con triggers

- Creación de triggers

los disparadores sin duda alguna considero que son lo mas usado cuando estamos administrando un servidor de base de datos, pero estan basadas en eventos y automatizados para accionarse cuando dichos eventos ocurran, ejecutando la accion que tengan asignada, estos se crean asociados a una tabla especifica, para guardar registro a modo historial lo cual es una utilidad muy grande, estos se puede accionar solo con eventos asociados a tablas, como inserts, updates o deletes

Ejemplos

```
CREATE TRIGGER validar_altura_peso
BEFORE INSERT ON patient
FOR EACH ROW
BEGIN
    IF NEW.height < 0.5 OR NEW.height > 2.5 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: La altura debe estar entre 0.5 y 2.5 metros.';
    END IF;

    IF NEW.weight < 3 OR NEW.weight > 300 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: El peso debe estar entre 3 y 300 kg.';
    END IF;
END;
```

por ejemplo en este trigger estamos haciendo uso del insert, este disparador apenas se realice un insert a la base de datos nos ayudara realizando la validacion de la altura o peso donde tenemos un rango predefinido entre altura y peso, pero en este caso solo cuando se realiza un insert, si realizamos un update el trigger no se ejecutara, por lo tanto debemos crear uno nuevo asociado a esta accion.

```
CREATE TRIGGER validar_altura_peso_update
BEFORE UPDATE ON patient
FOR EACH ROW
BEGIN
    IF NEW.height < 0.5 OR NEW.height > 2.5 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: La altura debe estar entre 0.5 y 2.5 metros.';
    END IF;

    IF NEW.weight < 3 OR NEW.weight > 300 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: El peso debe estar entre 3 y 300 kg.';
    END IF;
END;
```

```
END IF;  
END;
```

ahora lo que queda seria probarlo:

Resultado

```
-- valores no validos  
Run | New Tab  
INSERT INTO patient (full_name, gender, birth_date, blood_type, medical_insurance, height, weight) Error: El peso debe estar entre 3 y 300 kg.  
VALUES ('Juliana', 'FEMALE', '1990-04-15', 'O+', 'Sanitas', 1.8, 350); 8ms
```

y tambien podemos verificar que tambien se cumpla cuando realizamos modificaciones:

```
Run | New Tab  
UPDATE patient Error: La altura debe estar entre 0.5 y 2.5 metros.  
SET height = 3.0  
WHERE patient_id = 1; 7ms
```

si bien el rango que decidi usar no es muy coherente creo que se entiende el punto xD.

- Validación de datos antes de permitir operaciones al hacer (INSERT, UPDATE, DELETE)

si bien los disparadores son muy utiles como vimos en el anterior paso podemos usarlos para validar que los pasos sean correctos y incluso evitar que los datos se guarden en caso de no cumplir las validaciones esperadas. Por ejemplo en nuestro caso con las citas, si quiero que no se puedan agendar citas con fechas anteriores a la actual o tampoco se puedan agendar citas con demasiado tiempo despues o incluso no se puedan acumular muchas citas en un mismo dia, mi tabla citas me olvide añadir el campo hora por esto tendre un maximo de 4 citas por dia

Ejemplos

```
CREATE TRIGGER validar_cita  
BEFORE INSERT ON appointments  
FOR EACH ROW  
BEGIN  
    DECLARE citas_existentes INT;  
  
    IF NEW.appointment_date < CURDATE() THEN  
        SIGNAL SQLSTATE '45000'  
        SET MESSAGE_TEXT = 'Error: La cita no puede ser anterior a la fecha  
actual.';  
    END IF;  
  
    IF NEW.appointment_date > CURDATE() + INTERVAL 14 DAY THEN  
        SIGNAL SQLSTATE '45000'  
        SET MESSAGE_TEXT = 'Error: La cita no puede ser posterior a 14
```

```
días.';
    END IF;

    SELECT COUNT(*) INTO citas_existentes
    FROM appointments
    WHERE DATE(appointment_date) = DATE(NEW.appointment_date);

    IF citas_existentes >= 4 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: No se pueden registrar más de 4 citas en
la misma fecha.';
    END IF;
END;
```

siguiendo la misma logica para hacer un update usariamos la misma logica pero usariamos el before update

```
CREATE TRIGGER validar_cita_update
BEFORE UPDATE ON appointments
FOR EACH ROW
BEGIN
    DECLARE citas_existentes INT;

    IF NEW.appointment_date < CURDATE() THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: La cita no puede ser anterior a la fecha
actual.';
    END IF;

    IF NEW.appointment_date > CURDATE() + INTERVAL 14 DAY THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: La cita no puede ser posterior a 14
días.';
    END IF;

    SELECT COUNT(*) INTO citas_existentes
    FROM appointments
    WHERE DATE(appointment_date) = DATE(NEW.appointment_date);

    IF citas_existentes >= 4 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: No se pueden registrar más de 4 citas en
la misma fecha.';
    END IF;
END;
```

```
END IF;  
END;
```

y finalmente para hacer el delete tendria que tomar en cuenta otros aspectos, el estado de la cita no puede estar ya confirmada y a su vez tampoco se pueden cancelar citas con un tiempo menor a 24hr de la fecha actual

```
CREATE TRIGGER validar_cita_delete  
BEFORE DELETE ON appointments  
FOR EACH ROW  
BEGIN  
    DECLARE horas_faltantes INT;  
  
    IF OLD.status = 'CONFIRMED' THEN  
        SIGNAL SQLSTATE '45000'  
        SET MESSAGE_TEXT = 'Error: No se pueden eliminar citas  
confirmadas.';  
    END IF;  
  
    SET horas_faltantes = TIMESTAMPDIFF(HOUR, NOW(), OLD.appointment_date);  
  
    IF horas_faltantes < 24 THEN  
        SIGNAL SQLSTATE '45000'  
        SET MESSAGE_TEXT = 'Error: No se pueden eliminar citas que ocurren  
en menos de 24 horas.';  
    END IF;  
END;
```

Resultado

```
Run | New Tab  
INSERT INTO appointments (id_doctor, id_patient, appointment_date, status) Error: La cita no puede ser anterior a la fecha actual.  
VALUES (1, 1, '2023-09-01 10:00:00', 'PENDING'); 11ms
```

```
Run | New Tab  
INSERT INTO appointments (id_doctor, id_patient, appointment_date, status) Error: La cita no puede ser posterior a 14 días.  
VALUES (1, 1, CURDATE() + INTERVAL 15 DAY, 'PENDING'); 9ms
```

```
Run | New Tab  
DELETE FROM appointments WHERE id_appointment = 5; 6ms Error: No se pueden eliminar citas confirmadas.
```

podemos ver que todas las validaciones fueron pasadas correctamente.

- Utilizar los triggers para el registro de historial al momento de hacer un registro (INSERT, UPDATE, DELETE)

creo que lo mas cercano donde podemos aplicarlo es a los pacientes cuando se registren, cuando actualizen sus datos o cuando eliminen sus cuentas, seria la parte donde lo veo mas asociado. Pero lo primero seria crear la nueva tabla que almacenara todos estos registros.

```
CREATE TABLE patients_history (  
    history_id INT AUTO_INCREMENT PRIMARY KEY,  
    patient_id INT,  
    action_type VARCHAR(10),  
    old_full_name VARCHAR(50),  
    old_gender ENUM('MALE', 'FEMALE', 'OTHER'),  
    old_birth_date DATE,  
    old_blood_type VARCHAR(3),  
    old_medical_insurance VARCHAR(30),  
    old_height DOUBLE,  
    old_weight DOUBLE,  
    new_full_name VARCHAR(50),  
    new_gender ENUM('MALE', 'FEMALE', 'OTHER'),  
    new_birth_date DATE,  
    new_blood_type VARCHAR(3),  
    new_medical_insurance VARCHAR(30),  
    new_height DOUBLE,  
    new_weight DOUBLE,  
    action_timestamp DATETIME DEFAULT CURRENT_TIMESTAMP  
);
```

luego seguidamente comenzamos a crear los triggers que dara la logica a todo lo que ya hemos realizado.

CREATE

```
CREATE TRIGGER patients_history_insert  
AFTER INSERT ON patient  
FOR EACH ROW  
BEGIN  
    INSERT INTO patients_history (  
        patient_id, action_type, new_full_name, new_gender, new_birth_date,  
        new_blood_type, new_medical_insurance, new_height, new_weight  
    )  
    VALUES (  
        NEW.patient_id, 'INSERT', NEW.full_name, NEW.gender, NEW.birth_date,  
        NEW.blood_type, NEW.medical_insurance, NEW.height, NEW.weight  
    );  
END
```

```
);  
END;
```

UPDATE

```
CREATE TRIGGER patients_history_update  
AFTER UPDATE ON patient  
FOR EACH ROW  
BEGIN  
    INSERT INTO patients_history (  
        patient_id, action_type, old_full_name, old_gender, old_birth_date,  
old_blood_type,  
        old_medical_insurance, old_height, old_weight, new_full_name,  
new_gender, new_birth_date,  
        new_blood_type, new_medical_insurance, new_height, new_weight  
    )  
    VALUES (  
        OLD.patient_id, 'UPDATE', OLD.full_name, OLD.gender, OLD.birth_date,  
OLD.blood_type,  
        OLD.medical_insurance, OLD.height, OLD.weight, NEW.full_name,  
NEW.gender,  
        NEW.birth_date, NEW.blood_type, NEW.medical_insurance, NEW.height,  
NEW.weight  
    );  
END;
```

DELETE

```
CREATE TRIGGER patients_history_delete  
AFTER DELETE ON patient  
FOR EACH ROW  
BEGIN  
    INSERT INTO patients_history (  
        patient_id, action_type, old_full_name, old_gender, old_birth_date,  
old_blood_type,  
        old_medical_insurance, old_height, old_weight  
    )  
    VALUES (  
        OLD.patient_id, 'DELETE', OLD.full_name, OLD.gender, OLD.birth_date,  
OLD.blood_type,  
        OLD.medical_insurance, OLD.height, OLD.weight  
    );  
END;
```

```
);  
END;
```

Resultados

una vez ya los creamos donde simplemente lo que hacemos es redireccionar las acciones a nuestra tabla history algo que considero que si es demasiado util a partir de ahora en nuestra carrera, ahora comprobaremos que nuestro disparador que simula registro de historial de registros, actualizaciones y eliminaciones:

cuando realizamos un insert:

```
INSERT INTO patient (full_name, gender, birth_date, blood_type,  
medical_insurance, height, weight)  
VALUES ('Angel UwU', 'MALE', '2005-05-10', 'O+', 'sura', 1.69, 56);
```

> 0	history_id	1
> 1	patient_id	7
> 2	action_type	INSERT
> 3	old_full_name	(NULL)
> 4	old_gender	(NULL)
> 5	old_birth_date	(NULL)
> 6	old_blood_type	(NULL)
> 7	old_medical_in	(NULL)
> 8	old_height	(NULL)
> 9	old_weight	(NULL)
> 10	new_full_name	Angel UwU
> 11	new_gender	MALE
> 12	new_birth_date	2005-05-10
> 13	new_blood_type	O+
> 14	new_medical_i	sura
> 15	new_height	1.69
> 16	new_weight	56
> 17	action_timesta	2024-09-06 15

vemos que como no actualizamos nada y fue una accionde tipo insert los campos old estan en null, pasaremos a la accion update:

<input type="checkbox"/>		Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
<input type="checkbox"/>	> 1	1	Luis Colmenares	MALE	1980-05-15	O+	Seguro ABC	1.75	36
<input type="checkbox"/>	> 2	2	Lucia Gómez	FEMALE	1992-10-25	A-	Seguro XYZ	1.6	55
<input type="checkbox"/>	> 3	3	Carlos Martínez	MALE	1985-07-20	B+	Seguro DEF	1.8	85
<input type="checkbox"/>	> 4	4	Ana Serrano	FEMALE	1995-04-22	AB+	Seguro OPQ	1.65	60
<input type="checkbox"/>	> 5	7	Angel UwU	MALE	2005-05-10	O+	sura	1.69	56

modificaremos a angel uwu con el id 7

```
UPDATE patient
SET full_name = 'Angel Battler', weight = 70
WHERE patient_id = 7;
```

Q	col1	col2	col3
> 0	history_id	1	2
> 1	patient_id	7	7
> 2	action_type	INSERT	UPDATE
> 3	old_full_name	(NULL)	Angel UwU
> 4	old_gender	(NULL)	MALE
> 5	old_birth_date	(NULL)	2005-05-10
> 6	old_blood_type	(NULL)	O+
> 7	old_medical_ins	(NULL)	sura
> 8	old_height	(NULL)	1.69
> 9	old_weight	(NULL)	56
> 10	new_full_name	Angel UwU	Angel Battler
> 11	new_gender	MALE	MALE
> 12	new_birth_date	2005-05-10	2005-05-10
> 13	new_blood_type	O+	O+
> 14	new_medical_ins	sura	sura
> 15	new_height	1.69	1.69
> 16	new_weight	56	70
> 17	action_timestamp	2024-09-06 15	2024-09-06 16

podemos ver claramente que esta ocacion se modifio correctamente el usuario completando los campos antiguos con los datos correspondientes, pasaremos finalmente a la eliminacion que es basicamente muy similar al insert pero a la inversa donde tenemos los campos NEW en null

```
DELETE FROM patient WHERE patient_id = 7;
```


Q	col1	col2	col3	col4
> 0	history_id	1	2	3
> 1	patient_id	7	7	7
> 2	action_type	INSERT	UPDATE	DELETE
> 3	old_full_name	(NULL)	Angel UwU	Angel Battler
> 4	old_gender	(NULL)	MALE	MALE
> 5	old_birth_date	(NULL)	2005-05-10	2005-05-10
> 6	old_blood_type	(NULL)	O+	O+
> 7	old_medical_in	(NULL)	sura	sura
> 8	old_height	(NULL)	1.69	1.69
> 9	old_weight	(NULL)	56	70
> 10	new_full_name	Angel UwU	Angel Battler	(NULL)
> 11	new_gender	MALE	MALE	(NULL)
> 12	new_birth_date	2005-05-10	2005-05-10	(NULL)
> 13	new_blood_type	O+	O+	(NULL)
> 14	new_medical_i	sura	sura	(NULL)
> 15	new_height	1.69	1.69	(NULL)
> 16	new_weight	56	70	(NULL)
> 17	action_timesta	2024-09-06 15	2024-09-06 16	2024-09-06 16

Redactado y escrito por:

angelgabrielortega