

Trabajo Final Integrador (TFI) - Bases de Datos I

Datos Generales

❖ **Alumnos:**

Bruno Ludueña – brunoluduena@abc.gob.ar

Pablo Mariasch – pablolariasch85@gmail.com

Pablo Grassi – pjgrassi@gmail.com

❖ **Carrera:** Tecnicatura Universitaria en Programación

❖ **Materia:** Bases de Datos I

❖ **Profesor/a:** Ana Eugenia Mutti - Alejandro Lasalas

❖ **Fecha de Entrega:** 23/10/2025

Introducción

En este anexo reunimos los *prompts* y las interacciones que cada integrante del grupo mantuvo con herramientas de Inteligencia Artificial durante el desarrollo del trabajo práctico.

Cada estudiante presenta su propia experiencia, mostrando cómo utilizó la IA como apoyo para resolver dudas, revisar razonamientos o buscar distintas formas de encarar las actividades.

A continuación, se presentan las secciones correspondientes a cada integrante del grupo con sus respectivos *prompts* y comentarios.

Etapa 1:

Pregunta:

Me podés ayudar a crear la tabla Paciente con los campos básicos (id, nombre, apellido, fecha de nacimiento, dni y grupo sanguíneo) y que el grupo tenga una validación con CHECK para los valores A+, A-, B+, B-, AB+, AB-, O+, O-?

Respuesta:

Perfecto 👍

Entonces, como estás usando **MySQL 8** y querés que el **grupo sanguíneo sea obligatorio**, te guío paso a paso para construir la sentencia **CREATE TABLE**.

Vamos a hacerlo **por partes**, explicando el porqué de cada decisión antes de mostrar el código completo ⤵

Primero, definimos la tabla y la clave primaria:

`CREATE TABLE paciente (`

`id_paciente INT AUTO_INCREMENT PRIMARY KEY,`

 *Usamos un `id_paciente` numérico autoincremental para que cada paciente tenga un identificador único, sin depender del DNI.*

2 Datos personales

Agregamos las columnas de texto y la fecha:

```
nombre VARCHAR(50) NOT NULL,  
apellido VARCHAR(50) NOT NULL,  
fecha_nacimiento DATE NOT NULL,  
dni CHAR(20) NOT NULL UNIQUE,
```

3 Validación del grupo sanguíneo

Ahora la parte del **CHECK**:

```
grupo_sanguineo IS NULL OR grupo_sanguineo IN ('A+', 'A-', 'B+', 'B-', 'AB+',  
'AB-', 'O+', 'O-')
```

💡 Tabla completa

```
CREATE TABLE paciente (
    id_paciente INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL,
    apellido VARCHAR(50) NOT NULL,
    fecha_nacimiento DATE NOT NULL,
    dni VARCHAR(20) NOT NULL UNIQUE,
    grupo_sanguineo VARCHAR(5),
    CONSTRAINT chk_grupo_sanguineo
        CHECK (grupo_sanguineo IS NULL OR grupo_sanguineo IN ('A+', 'A-',
'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-'))
);
```

Pregunta:

¿Cómo creo un trigger que valide que la fecha de nacimiento no sea futura ni mayor a 120 años al insertar un paciente?

Respuesta:

Qué queremos validar

Al insertar un paciente (**INSERT**) o actualizar la fecha de nacimiento (**UPDATE**):

1. No puede ser **futura** → `fecha_nacimiento <= CURRENT_DATE()`.
2. No puede tener más de **120 años** → `fecha_nacimiento >= CURRENT_DATE() - INTERVAL 120 YEAR.`

Idea general

1. Declaramos una variable **edad INT**.
2. Calculamos la edad del paciente usando `TIMESTAMPDIFF(YEAR, fecha_nacimiento, CURRENT_DATE())`.
3. Hacemos **dos validaciones separadas**:
 - Fecha no futura (`fecha_nacimiento <= CURRENT_DATE()`)
 - Edad ≤ 120 (`edad <= 120`)

DELIMITER \$\$

```
CREATE TRIGGER trg_validar_fecha_nacimiento
BEFORE INSERT ON paciente
FOR EACH ROW
BEGIN
    DECLARE edad INT;

    -- Validación 1: fecha no futura
    IF NEW.fecha_nacimiento > CURRENT_DATE() THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Fecha de nacimiento inválida: no puede ser
futura';
    END IF;

    -- Calcular edad
    SET edad = TIMESTAMPDIFF(YEAR, NEW.fecha_nacimiento,
CURRENT_DATE());

    -- Validación 2: edad máxima 120 años
    IF edad > 120 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Fecha de nacimiento inválida: no puede ser
mayor a 120 años';
    END IF;

END$$
```

DELIMITER ;

Nota: Se usa el mismo trigger para UPDATE.

Pregunta:

¿Cómo creo una tabla HistoriaClinica con clave primaria, un campo único para número de historia y una relación 1 a 1 con Paciente con ON DELETE CASCADE?

Respuesta:

1 Conceptos clave

Queremos:

1. **Tabla HistoriaClinica.**
2. **Clave primaria** propia (`id_historia` autoincremental).
3. **Número de historia único** (`numero_historia`).
4. **Relación 1 a 1 con Paciente** usando `id_paciente`.
5. **Borrar la historia si se borra el paciente (ON DELETE CASCADE)**.

💡 Pista conceptual:

- En una relación 1:1, la FK puede estar en cualquiera de las dos tablas.
- Lo más común es poner la FK en `HistoriaClinica`, apuntando a `Paciente`.
- Para garantizar 1:1, además de la FK, hacemos que `id_paciente` sea **UNIQUE** en `HistoriaClinica`.

2 Estructura básica de la tabla

```
CREATE TABLE HistoriaClinica (
    id_historia INT AUTO_INCREMENT PRIMARY KEY,
    numero_historia VARCHAR(20) NOT NULL UNIQUE,
    id_paciente INT NOT NULL UNIQUE,
    -- otros campos de historia clínica
    CONSTRAINT fk_paciente
        FOREIGN KEY (id_paciente)
        REFERENCES paciente(id_paciente)
        ON DELETE CASCADE
);
```

Pregunta:

le podemos agregar antecedentes, medicacion actual y observaciones, que pueden ser null

Respuesta:

¡Perfecto! 👍

Si queremos agregar esos campos y permitir que queden vacíos (`NULL`), simplemente los definimos como `VARCHAR` o `TEXT` sin `NOT NULL`.

Te muestro cómo quedaría la tabla **HistoriaClinica** completa con esos campos:

```
CREATE TABLE HistoriaClinica (
    id_historia INT AUTO_INCREMENT PRIMARY KEY,
    numero_historia VARCHAR(20) NOT NULL UNIQUE,
    antecedentes TEXT,
    medicacion_actual TEXT,
    observaciones TEXT,
    id_paciente INT NOT NULL UNIQUE,
    CONSTRAINT fk_paciente
        FOREIGN KEY (id_paciente)
        REFERENCES paciente(id_paciente)
        ON DELETE CASCADE
);
```

Etapa 2:

Pregunta:

¿Podés darme un script SQL que inserte 200.000 pacientes con datos aleatorios usando combinaciones de `SELECT ... UNION ALL SELECT`, y funciones como `RAND()`, `DATE_SUB` y `ELT` para nombre, dni, fecha de nacimiento y grupo sanguíneo?

Respuesta:

¡Perfecto! 😊 Vamos a armar un **script SQL puro** que genere 200.000 pacientes aleatorios usando `SELECT ... UNION ALL SELECT, RAND(), DATE_SUB() y ELT()`.

Para no escribir manualmente 200.000 `UNION ALL SELECT`, vamos a usar **bloques de 10 filas combinados con producto cartesiano**, y luego limitamos a 200.000 con `LIMIT`.

```
INSERT INTO paciente (nombre, apellido, dni, fecha_nacimiento,
grupo_sanguineo)
SELECT
    CONCAT('Nombre', t1.i, t2.i, t3.i, t4.i),
    CONCAT('Apellido', t2.i, t3.i, t4.i, t5.i),
    LPAD((t1.i*10000 + t2.i*1000 + t3.i*100 + t4.i*10 + t5.i), 8, '0') ,
    DATE_SUB(CURDATE(), INTERVAL FLOOR(RAND()*43830) DAY),
```

```
ELT(FLOOR(RAND())*8), 'A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-')
FROM
    (SELECT 0 AS i UNION ALL SELECT 1 UNION ALL SELECT 2 UNION ALL
     SELECT 3 UNION ALL SELECT 4
      UNION ALL SELECT 5 UNION ALL SELECT 6 UNION ALL SELECT 7
      UNION ALL SELECT 8 UNION ALL SELECT 9) t1,
    (SELECT 0 AS i UNION ALL SELECT 1 UNION ALL SELECT 2 UNION ALL
     SELECT 3 UNION ALL SELECT 4
      UNION ALL SELECT 5 UNION ALL SELECT 6 UNION ALL SELECT 7
      UNION ALL SELECT 8 UNION ALL SELECT 9) t2,
    (SELECT 0 AS i UNION ALL SELECT 1 UNION ALL SELECT 2 UNION ALL
     SELECT 3 UNION ALL SELECT 4
      UNION ALL SELECT 5 UNION ALL SELECT 6 UNION ALL SELECT 7
      UNION ALL SELECT 8 UNION ALL SELECT 9) t3,
    (SELECT 0 AS i UNION ALL SELECT 1 UNION ALL SELECT 2 UNION ALL
     SELECT 3 UNION ALL SELECT 4
      UNION ALL SELECT 5 UNION ALL SELECT 6 UNION ALL SELECT 7
      UNION ALL SELECT 8 UNION ALL SELECT 9) t4,
    (SELECT 0 AS i UNION ALL SELECT 1 UNION ALL SELECT 2 UNION ALL
     SELECT 3 UNION ALL SELECT 4
      UNION ALL SELECT 5 UNION ALL SELECT 6 UNION ALL SELECT 7
      UNION ALL SELECT 8 UNION ALL SELECT 9) t5
LIMIT 200000;
```

Pregunta:

Io intente pero me da error por el volumen, se podra hacer en dos lotes de 100.000, teniendo en cuenta que los dni son unicos.

Respuesta:

Estrategia

1. **Primer lote:** `LIMIT 100000`
2. **Segundo lote:** `LIMIT 100000 OFFSET 100000` o usando otra combinación de los bloques generadores para que no se repitan los valores.

💡 En nuestro caso, como usamos los bloques `t1..t5` y combinaciones determinísticas para el DNI, podemos hacer **dos inserts iguales con un OFFSET diferente en los cálculos** para asegurar DNIs distintos, o simplemente cambiar ligeramente el cálculo del DNI en el segundo lote.

Script SQL – Lote 1 (primeros 100.000 pacientes)

```
INSERT INTO paciente (nombre, apellido, dni, fecha_nacimiento,  
grupo_sanguineo)  
SELECT  
    CONCAT('Nombre', t1.i, t2.i, t3.i, t4.i),  
    CONCAT('Apellido', t2.i, t3.i, t4.i, t5.i),  
    LPAD((t1.i*10000 + t2.i*1000 + t3.i*100 + t4.i*10 + t5.i), 8, '0'),  
    DATE_SUB(CURDATE(), INTERVAL FLOOR(RAND()*43830) DAY),  
    ELT(FLOOR(RAND()*8), 'A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-')  
FROM  
    (SELECT 0 AS i UNION ALL SELECT 1 UNION ALL SELECT 2 UNION ALL  
    SELECT 3 UNION ALL SELECT 4  
        UNION ALL SELECT 5 UNION ALL SELECT 6 UNION ALL SELECT 7  
        UNION ALL SELECT 8 UNION ALL SELECT 9) t1,  
    (SELECT 0 AS i UNION ALL SELECT 1 UNION ALL SELECT 2 UNION ALL  
    SELECT 3 UNION ALL SELECT 4  
        UNION ALL SELECT 5 UNION ALL SELECT 6 UNION ALL SELECT 7  
        UNION ALL SELECT 8 UNION ALL SELECT 9) t2,  
    (SELECT 0 AS i UNION ALL SELECT 1 UNION ALL SELECT 2 UNION ALL  
    SELECT 3 UNION ALL SELECT 4  
        UNION ALL SELECT 5 UNION ALL SELECT 6 UNION ALL SELECT 7  
        UNION ALL SELECT 8 UNION ALL SELECT 9) t3,  
    (SELECT 0 AS i UNION ALL SELECT 1 UNION ALL SELECT 2 UNION ALL  
    SELECT 3 UNION ALL SELECT 4  
        UNION ALL SELECT 5 UNION ALL SELECT 6 UNION ALL SELECT 7  
        UNION ALL SELECT 8 UNION ALL SELECT 9) t4,  
    (SELECT 0 AS i UNION ALL SELECT 1 UNION ALL SELECT 2 UNION ALL  
    SELECT 3 UNION ALL SELECT 4  
        UNION ALL SELECT 5 UNION ALL SELECT 6 UNION ALL SELECT 7  
        UNION ALL SELECT 8 UNION ALL SELECT 9) t5  
LIMIT 100000;
```

Script SQL – Lote 2 (siguientes 100.000 pacientes)

```
INSERT INTO paciente (nombre, apellido, dni, fecha_nacimiento,  
grupo_sanguineo)  
SELECT  
    CONCAT('Nombre', t1.i, t2.i, t3.i, t4.i, 'B'),  
    CONCAT('Apellido', t2.i, t3.i, t4.i, t5.i, 'B'),  
    LPAD((t1.i*10000 + t2.i*1000 + t3.i*100 + t4.i*10 + t5.i + 100000), 8, '0'),  
    DATE_SUB(CURDATE(), INTERVAL FLOOR(RAND()*43830) DAY) AS  
    fecha_nacimiento,
```

```
ELT(FLOOR(RAND())*8), 'A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-') AS
grupo_sanguineo
FROM
    (SELECT 0 AS i UNION ALL SELECT 1 UNION ALL SELECT 2 UNION ALL
     SELECT 3 UNION ALL SELECT 4
      UNION ALL SELECT 5 UNION ALL SELECT 6 UNION ALL SELECT 7
      UNION ALL SELECT 8 UNION ALL SELECT 9) t1,
    (SELECT 0 AS i UNION ALL SELECT 1 UNION ALL SELECT 2 UNION ALL
     SELECT 3 UNION ALL SELECT 4
      UNION ALL SELECT 5 UNION ALL SELECT 6 UNION ALL SELECT 7
      UNION ALL SELECT 8 UNION ALL SELECT 9) t2,
    (SELECT 0 AS i UNION ALL SELECT 1 UNION ALL SELECT 2 UNION ALL
     SELECT 3 UNION ALL SELECT 4
      UNION ALL SELECT 5 UNION ALL SELECT 6 UNION ALL SELECT 7
      UNION ALL SELECT 8 UNION ALL SELECT 9) t3,
    (SELECT 0 AS i UNION ALL SELECT 1 UNION ALL SELECT 2 UNION ALL
     SELECT 3 UNION ALL SELECT 4
      UNION ALL SELECT 5 UNION ALL SELECT 6 UNION ALL SELECT 7
      UNION ALL SELECT 8 UNION ALL SELECT 9) t4,
    (SELECT 0 AS i UNION ALL SELECT 1 UNION ALL SELECT 2 UNION ALL
     SELECT 3 UNION ALL SELECT 4
      UNION ALL SELECT 5 UNION ALL SELECT 6 UNION ALL SELECT 7
      UNION ALL SELECT 8 UNION ALL SELECT 9) t5
LIMIT 100000;
```

Pregunta:

¿Cómo inserto 200000 historias clínicas basadas en los pacientes existentes, generando datos ficticios con CONCAT y LIMIT?

Respuesta:

¡Perfecto! 😊 Como ya tenés 200.000 pacientes en la tabla `paciente`, podemos generar **una historia clínica por paciente** usando SQL puro y funciones como `CONCAT()` y `RAND()`.

```
INSERT INTO HistoriaClinica (nro_historia, antecedentes, medicacion_actual,
observaciones, id_paciente)
SELECT
    CONCAT('HC', LPAD(id_paciente, 6, '0')),
    CONCAT('Antecedentes del paciente ', id_paciente),
    CONCAT('Medicación del paciente ', id_paciente),
    CONCAT('Observaciones del paciente ', id_paciente),
    id_paciente
FROM Paciente
```

```
ORDER BY id_paciente  
LIMIT 200000;
```

Etapa 3:

Pregunta:

Quiero realizar una consulta SQL que combine ambas tablas con un join tomando como referencia el id paciente, donde pacientes sea p e historias clinicas h, que concatene nombre y apellido como nombre completo, no incluya observaciones ni antecedentes, y que salga ordenado por id paciente, todo con un limite de 20 registros.

Respuesta:

Excelente 🍀

Vamos a armar la consulta paso a paso según lo que pedís:

- Combinar tablas **paciente (p)** e **historia_clinica (h)**
- Usar **JOIN** (una relación 1 a 1)
- Crear un campo **nombre_completo** uniendo **nombre** y **apellido**
- No mostrar **observaciones**, **antecedentes** ni ningún **id**
- Ordenar por **id** del paciente
- Limitar el resultado a 20 registros

Consulta SQL

SELECT

```
p.id AS id_paciente,  
CONCAT(p.nombre, ' ', p.apellido) AS nombre_completo,  
p.dni,  
p.fecha_nacimiento,  
p.grupo_sanguineo,  
h.numero_historia,
```

```
h.medicacion_actual  
  
FROM paciente AS p  
  
INNER JOIN historia_clinica AS h  
  
ON p.id = h.paciente_id  
  
ORDER BY p.id  
  
LIMIT 20;
```

Pregunta:

¿Podés escribirme un script SQL que muestre los pacientes menores de 18 años, mostrando su ID, nombre completo, DNI, edad, grupo sanguíneo, número de historia clínica, antecedentes y medicación actual, haciendo un JOIN entre las tablas Paciente y HistoriaClinica, y ordenando el resultado por edad de menor a mayor?

Respuesta:

¡Claro! Para armar este script necesitamos:

- Calcular la edad del paciente a partir de su `fechaNacimiento`.
- Hacer un `JOIN` entre `Paciente` y `HistoriaClinica` usando el `idPaciente`.
- Seleccionar los campos que mencionaste y ordenar por edad ascendente.

Suponiendo que las columnas de tus tablas son algo así:

Tabla Paciente: `idPaciente`, `nombre`, `apellido`, `dni`, `fechaNacimiento`, `grupoSanguineo`

Tabla HistoriaClinica: `idHistoriaClinica`, `idPaciente`, `antecedentes`, `medicacionActual`

El script SQL podría ser:

```
SELECT  
p.idPaciente,  
CONCAT(p.nombre, ' ', p.apellido) AS nombreCompleto,  
p.dni,
```

```
TIMESTAMPDIFF(YEAR, p.fechaNacimiento, CURDATE()) AS edad,  
p.grupoSanguineo,  
h.idHistoriaClinica,  
h.antecedentes,  
h.medicacionActual  
FROM  
    Paciente p  
JOIN  
    HistoriaClinica h ON p.idPaciente = h.idPaciente  
WHERE  
    TIMESTAMPDIFF(YEAR, p.fechaNacimiento, CURDATE()) < 18  
ORDER BY  
    edad ASC;
```

Pregunta:

Me gustaría realizar un conteo de pacientes por grupo sanguíneo, que solo muestre los grupos sanguíneos que sean mayores a 25.000.

Respuesta:

Muy buena idea 🎉 — estás buscando **agrupar por grupo sanguíneo** y **filtrar** solo aquellos con más de 25.000 pacientes.

En SQL eso se hace con **GROUP BY** junto con **HAVING** (no con **WHERE**, porque el filtro se aplica *después* de agrupar).

Consulta SQL

```
SELECT  
    grupo_sanguineo,  
    COUNT(*) AS cantidad_pacientes  
FROM paciente  
GROUP BY grupo_sanguineo  
HAVING COUNT(*) > 25000  
ORDER BY cantidad_pacientes DESC;
```

Pregunta:

Generá un script SQL que obtenga id_paciente, nombre, apellido, dni y edad (calculada con **TIMESTAMPDIFF**) de pacientes mayores o iguales a 80 años, usando una subconsulta en el WHERE para filtrar por edad. El query debe ordenar los resultados por edad en orden ascendente.

Respuesta:

Perfecto, podemos usar una subconsulta en el `WHERE` para filtrar por edad calculada. Una forma común es calcular la edad dentro de un `SELECT` en línea. Por ejemplo:

```
SELECT
    idPaciente,
    nombre,
    apellido,
    dni,
    TIMESTAMPDIFF(YEAR, fechaNacimiento, CURDATE()) AS edad
FROM
    Paciente
WHERE
    idPaciente IN (
        SELECT idPaciente
        FROM Paciente
        WHERE TIMESTAMPDIFF(YEAR, fechaNacimiento, CURDATE()) >= 80
    )
ORDER BY
    edad ASC;
```

Explicación:

- La subconsulta (`SELECT idPaciente FROM Paciente WHERE TIMESTAMPDIFF(YEAR, fechaNacimiento, CURDATE()) >= 80`) devuelve los IDs de los pacientes con 80 años o más.
- El `WHERE idPaciente IN (...)` filtra los registros de la tabla principal usando esos IDs.
- `TIMESTAMPDIFF(YEAR, fechaNacimiento, CURDATE()) AS edad` calcula la edad para mostrarla.
- `ORDER BY edad ASC` ordena de menor a mayor.

Pregunta:

¿Podés crear una vista que se llame `Vista_Donantes_Universales` y que muestre los pacientes que tienen grupo sanguíneo 'O-' y tienen entre 18 y 50 años?

Quiero que incluya el id, el nombre completo (nombre y apellido juntos), el DNI, el grupo sanguíneo y la edad calculada.

Además, que la vista haga un JOIN entre las tablas Paciente y HistoriaClinica usando el id del paciente.

Respuesta:

¡Claro! Podemos crear la vista considerando los requisitos que mencionaste. Suponiendo que las tablas tienen estas columnas:

- **Paciente:** `idPaciente, nombre, apellido, dni, fechaNacimiento, grupoSanguineo`
- **HistoriaClinica:** `idHistoriaClinica, idPaciente`

El script SQL sería:

```
CREATE VIEW Vista_Donantes_Universales AS
SELECT
    p.idPaciente,
    CONCAT(p.nombre, ' ', p.apellido) AS nombreCompleto,
    p.dni,
    p.grupoSanguineo,
    TIMESTAMPDIFF(YEAR, p.fechaNacimiento, CURDATE()) AS edad
FROM
    Paciente p
JOIN
    HistoriaClinica h ON p.idPaciente = h.idPaciente
WHERE
    p.grupoSanguineo = 'O-'
    AND TIMESTAMPDIFF(YEAR, p.fechaNacimiento, CURDATE()) BETWEEN
    18 AND 50;
```

Pregunta:

¿Podés escribir un script SQL para mostrar todos los registros de la vista llamada `vista_donantes_universales`?

Respuesta:

¡Claro! Para mostrar todos los registros de una vista en SQL, simplemente se utiliza un `SELECT *` sobre el nombre de la vista. El script sería:

```
SELECT *
FROM vista_donantes_universales;
```

Pregunta:

¿Me podés armar un query que saque el id, nombre, apellido, dni y grupo sanguíneo de la tabla Paciente, pero que ignore el índice dni y filtre por dni igual a '00123456'?

Respuesta:

¡Claro! En MySQL podés **forzar que no se use un índice específico** usando la cláusula **IGNORE INDEX**. Por ejemplo, si el índice sobre la columna **dni** se llama **idx_dni**, el query sería:

```
SELECT
    idPaciente,
    nombre,
    apellido,
    dni,
    grupoSanguineo
FROM
    Paciente
IGNORE INDEX (idx_dni)
WHERE
    dni = '00123456';
```

Pregunta:

ahora necesito lo mismo pero con indice

Respuesta:

```
SELECT
    idPaciente,
    nombre,
    apellido,
    dni,
    grupoSanguineo
FROM
    Paciente
WHERE
    dni = '00123456';
```

Pregunta:

¿Me podés armar un script SQL para crear un índice llamado **idx_paciente_fecha_nac** en la tabla **Paciente**, usando la columna **fecha_nacimiento**?

Respuesta:

¡Claro! Para crear un índice en MySQL sobre la columna `fecha_nacimiento` de la tabla `Paciente`, el script sería:

```
CREATE INDEX idx_paciente_fecha_nac
ON Paciente (fecha_nacimiento);
```

Pregunta:

¿Me podés escribir un query que muestre id, nombre, apellido, dni y edad calculada de los pacientes que tengan 80 años o más?

Quiero que uses una condición en el WHERE con `fecha_nacimiento` comparándola con la fecha actual menos 80 años, y que ordene los resultados por edad de menor a mayor.

Respuesta:

¡Claro! En lugar de calcular la edad con `TIMESTAMPDIFF` en el `WHERE`, podemos comparar directamente `fechaNacimiento` con la fecha actual menos 80 años usando `DATE_SUB`. Suponiendo que la columna se llama `fechaNacimiento`, el query sería:

```
SELECT
    idPaciente,
    nombre,
    apellido,
    dni,
    TIMESTAMPDIFF(YEAR, fechaNacimiento, CURDATE()) AS edad
FROM
    Paciente
WHERE
    fechaNacimiento <= DATE_SUB(CURDATE(), INTERVAL 80 YEAR)
ORDER BY
    edad ASC;
```

Pregunta:

¿Me podés armar un script para crear un índice llamado `idx_historia_fk` en la tabla `HistoriaClinica` usando la columna `id_paciente`?

Respuesta:

¡Claro! Para crear un índice en la tabla `HistoriaClinica` sobre la columna `id_paciente`, el script SQL sería:

```
CREATE INDEX idx_historia_fk
ON HistoriaClinica (id_paciente);
```

Pregunta:

¿Me podés armar un query que muestre el id, nombre completo (nombre y apellido separados por coma), dni, grupo sanguíneo, número de historia clínica y medicación actual?

Que haga un INNER JOIN entre las tablas Paciente y HistoriaClinica, pero que ignore el índice idx_historia_fk en la tabla HistoriaClinica.

Quiero que ordene por id de paciente y muestre solo los primeros 20 resultados.

Respuesta:

¡Claro! Basado en tus indicaciones, el query en MySQL quedaría así:

```
SELECT
    p.id_paciente,
    CONCAT(p.nombre, ', ', p.apellido) AS nombre_completo,
    p.dni,
    p.grupo_sanguineo,
    h.nro_historia,
    h.medicacion_actual
FROM
    Paciente p
INNER JOIN
    HistoriaClinica h IGNORE INDEX (idx_historia_fk)
    ON p.id_paciente = h.id_paciente
ORDER BY
    p.id_paciente
LIMIT 20;
```

Pregunta:

Ahora necesito lo mismo pero con indice, teniendo en cuenta que ya lo tiene naturalmente y no es necesario forzarlo.

Respuesta:

```
SELECT
    p.id_paciente,
    CONCAT(p.nombre, ', ', p.apellido) AS nombre_completo,
    p.dni,
    p.grupo_sanguineo,
    h.nro_historia,
    h.medicacion_actual
FROM
    Paciente p
INNER JOIN
    HistoriaClinica h
```

```
ON p.id_paciente = h.id_paciente
ORDER BY
    p.id_paciente
LIMIT 20;
```

Etapa 4:

Pregunta:

Necesito crear un usuario app_user con contraseña 123456.

Respuesta:

¡Perfecto! En MySQL, podés crear un usuario con la instrucción **CREATE USER** y asignarle una contraseña. Por ejemplo:

```
CREATE USER 'app_user'@'localhost' IDENTIFIED BY '123456';
```

Pregunta:

Ahora a ese usuario autorizar Select, Insert y Update, para las dos tablas, paciente e historia clinica, para que sea entendible que se haga de a una linea de script por cada tabla y metodo.

Respuesta:

¡Perfecto! Para que sea **clarito y entendible**, podemos otorgar los permisos **una línea por tabla y tipo de operación**. Suponiendo que la base de datos se llama **clinica53**, el script sería:

-- Permisos para la tabla Paciente

```
GRANT SELECT ON clinica53.Paciente TO 'app_user'@'localhost';
GRANT INSERT ON clinica53.Paciente TO 'app_user'@'localhost';
GRANT UPDATE ON clinica53.Paciente TO 'app_user'@'localhost';
```

-- Permisos para la tabla HistoriaClinica

```
GRANT SELECT ON clinica53.HistoriaClinica TO 'app_user'@'localhost';
GRANT INSERT ON clinica53.HistoriaClinica TO 'app_user'@'localhost';
GRANT UPDATE ON clinica53.HistoriaClinica TO 'app_user'@'localhost';
```

Pregunta:

Ver los permisos del usuario app_user.

Respuesta:

1 Usando SHOW GRANTS

Es la manera más directa:

```
SHOW GRANTS FOR 'app_user'@'localhost';
```

Pregunta:

Como probar borrar el registro con id = 1 de historia clinica con el user_app.

Respuesta:

Intentar borrar el registro 1 de HistoriaClinica

```
DELETE FROM HistoriaClinica  
WHERE id = 1;
```

Pregunta:

Como crear una vista llamada vista_pacientes_publica, que no muestre datos sensibles, solo se quiere mostrar id_paciente, nombre y apellido concatenado como nombre_completo y grupo_sanguineo.

Respuesta:

Perfecto 👍

Podés crear esa vista muy fácilmente con el siguiente script SQL:

```
CREATE VIEW vista_pacientes_publica AS  
SELECT  
    id_paciente,  
    CONCAT(nombre, ' ', apellido) AS nombre_completo,  
    grupo_sanguineo  
FROM  
    Paciente;
```

Explicación:

- `CREATE VIEW vista_pacientes_publica AS` crea la vista con ese nombre.
- `CONCAT(nombre, ' ', apellido)` une el nombre y el apellido en una sola columna llamada `nombre_completo`.
- Solo se muestran los campos **no sensibles**: `id_paciente`, `nombre_completo` y `grupo_sanguineo`.

- No se incluyen datos privados como DNI, dirección, teléfono, etc.

Así, cualquier usuario con permisos de lectura sobre la vista podrá consultar:

```
SELECT * FROM vista_pacientes_publica;
```

Pregunta:

Ahora quiero crear una nueva vista llamada vista_historial_resumido, donde se vean datos de las dos tablas como id_historia, id_paciente, nombre_completo, nro_historia y antecedentes.

Respuesta:

```
CREATE VIEW vista_historial_resumido AS
SELECT
    h.id_historia,
    p.id_paciente,
    CONCAT(p.nombre, ' ', p.apellido) AS nombre_completo,
    h.nro_historia,
    h.antecedentes
FROM
    Paciente p
INNER JOIN
    HistoriaClinica h
ON p.id_paciente = h.id_paciente;
```

Explicación:

- Se crea la vista con `CREATE VIEW vista_historial_resumido AS.`
- Se unen ambas tablas usando `INNER JOIN` por el campo `id_paciente`.
- `CONCAT(p.nombre, ' ', p.apellido)` genera el nombre completo del paciente.
- Solo se incluyen columnas **no sensibles**, manteniendo un resumen de información médica básica.

Luego podés consultar la vista con:

```
SELECT * FROM vista_historial_resumido;
```

Pregunta:

En las ultimas dos vistas creadas otorgar permisos de lectura al usuario app_user.

Respuesta:

Perfecto 

Podés otorgar permisos de lectura (**SELECT**) al usuario **app_user** sobre ambas vistas con los siguientes comandos SQL:

```
GRANT SELECT ON vista_pacientes_publica TO 'app_user'@'localhost';
GRANT SELECT ON vista_historial_resumido TO 'app_user'@'localhost';
```

Pregunta:

Como se podria realizar una prueba de integridad referencial tratando de realizar un insert de datos donde se ingrese un id_paciente inexistente a la tabla historia clinica.

Respuesta:

Perfecto 

Esa es una **prueba clásica de integridad referencial**, y sirve para verificar que la **foreign key (clave foránea)** entre **HistoriaClinica** y **Paciente** está funcionando correctamente.

Te muestro cómo hacerlo paso a paso 

1 Suposición

Tu base de datos **clinica53** tiene esta relación:

```
ALTER TABLE HistoriaClinica
ADD CONSTRAINT fk_historia_paciente
FOREIGN KEY (id_paciente)
REFERENCES Paciente(id_paciente);
```

Esto asegura que **solo se puedan registrar historias clínicas de pacientes existentes**.

2 Prueba práctica de integridad referencial

Intentá ejecutar este **INSERT** (usando un **id_paciente** que no exista, por ejemplo **999999**):

```
INSERT INTO HistoriaClinica (id_paciente, nro_historia, antecedentes,  
medicacion_actual)  
  
VALUES (999999, 'HC-999999', 'Sin antecedentes', 'Ninguna medicación');
```

Pregunta:

Ahora quisiera realizar otra validacion de integridad para ver si funciona el check de la tabla paciente realizando un insert donde el grupo sanguineo sea XZ.

Respuesta:

Excelente 🍏

Esa es otra prueba muy útil — esta vez para verificar la **integridad de dominio**, es decir, que los valores ingresados cumplan con una restricción definida mediante un **CHECK**.

Veamos cómo hacerlo paso a paso ⤵

✳️ 1 Suposición

Tu tabla **Paciente** tiene un **CHECK** sobre el campo **grupo_sanguineo**, por ejemplo algo así:

```
CREATE TABLE Paciente (  
    id_paciente INT PRIMARY KEY,  
    nombre VARCHAR(50),  
    apellido VARCHAR(50),  
    dni VARCHAR(15),  
    fecha_nacimiento DATE,  
    grupo_sanguineo VARCHAR(3),  
    CHECK (grupo_sanguineo IN ('A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-'))  
);
```

🧪 2 Prueba del CHECK (intentando un valor inválido)

Podés probar si el **CHECK** realmente funciona con este **INSERT**:

```
INSERT INTO Paciente (nombre, apellido, dni, fecha_nacimiento,  
grupo_sanguineo)  
VALUES ('Juan', 'Pérez', '12345678', '1980-05-15', 'XZ');
```

Pregunta:

Como realizar la implementación de Seguridad: Procedimiento Almacenado Anti-Inyección SQL, donde el procedimiento se llame sp_buscar_paciente_por_dni, con el valor de un dni_busqueda que es el dni del paciente que se trata de localizar. Usar nombre_completo, y no usar dni en el select.

Respuesta:

Excelente 🎉

Esta es una **muy buena práctica de seguridad**: crear un **procedimiento almacenado que proteja contra inyección SQL**, especialmente cuando se reciben valores de entrada como un DNI.

Te dejo el ejemplo completo, explicado paso a paso ↴

1 Objetivo

Queremos crear un procedimiento llamado `sp_buscar_paciente_por_dni` que reciba un parámetro `dni_busqueda` y devuelva los datos del paciente correspondiente, evitando **inyección SQL**.

2 Script seguro del procedimiento

DELIMITER //

```
CREATE PROCEDURE sp_buscar_paciente_por_dni (
    IN dni_busqueda VARCHAR(20)
)
BEGIN
    -- Selecciona los datos del paciente filtrando por DNI
    SELECT
        id_paciente,
        CONCAT(nombre, ' ', apellido) AS nombre_completo,
        fecha_nacimiento,
        grupo_sanguineo
    FROM
        Paciente
    WHERE
        dni = dni_busqueda;
END //
```

DELIMITER ;

🔒 3 Por qué este procedimiento es anti-inyección SQL

- ✓ No usa **concatenaciones de texto dinámico** (que es lo que suele permitir inyecciones).
- ✓ El parámetro `dni_busqueda` se trata como un **valor**, no como parte del código SQL.
- ✓ El `WHERE dni = dni_busqueda` utiliza el parámetro directamente, garantizando que MySQL escape el valor correctamente.

💡 4 Ejemplo de uso

```
CALL sp_buscar_paciente_por_dni('00123456');
```

🚫 5 Intento de inyección SQL (para probar)

Si alguien intenta algo malicioso, por ejemplo:

```
CALL sp_buscar_paciente_por_dni(""); DROP TABLE Paciente; --";
```

👉 El procedimiento **no ejecutará un SQL dinámico**, así que el texto `'12345678 OR 1=1'` se interpretará como **un valor literal**, no como código SQL.

Resultado: **no se filtrarán más registros de los esperados**.

Etapa 5:

Pregunta:

Hemos realizado una prueba de acceso concurrente de dos sesiones del mismo usuario que desea acceder por ejemplo a la misma tabla e id de paciente por ejemplo, generando un error deadlock, cual sería un mini resumen de la explicación de lo sucedido.

Respuesta:

Aquí tenés un **mini resumen claro y conciso** de lo que pasó en tu prueba de acceso concurrente:

◆ Resumen de la prueba y resultado

1. Contexto:

Dos sesiones del mismo usuario (`app_user`) intentaron **modificar o bloquear simultáneamente la misma fila** de la tabla `Paciente` (mismo `id_paciente`) dentro de una transacción.

2. Qué sucedió internamente en InnoDB:

- Cada sesión adquirió **bloqueos sobre filas**.
- Una de las sesiones necesitaba un lock que ya estaba tomado por la otra.
- Como ambas transacciones esperaban recursos que la otra tenía, se generó un **deadlock**.

3. Cómo lo resolvió MySQL:

- InnoDB detecta automáticamente el deadlock.
- Una de las transacciones es **abortada automáticamente (ROLLBACK)** para romper el ciclo.
- La otra transacción puede continuar normalmente.

4. Errores que se observan en la aplicación:

- `ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction`
- Indica que la transacción abortada debe ser **reintentada** si se quiere completar la operación.

5. Lección / buenas prácticas:

- Mantener transacciones **cortas y rápidas**.
- Acceder a filas/tablas en **orden consistente** si varias transacciones interactúan.
- Implementar **reintentos en la aplicación** para manejar deadlocks.
- Usar índices adecuados y `SELECT ... FOR UPDATE` solo cuando sea necesario.

