

TRABAJO PRÁCTICO INTEGRADOR

Presentado por:

Pablo Garay

Israel Garcia Moscoso

Jose Darío Gimenez

Juan Esteban Gelos

Comisión: 14

Materia: Base de Datos I.

Profesor: Sergio Neira

Tutor: Martín García

Video explicativo: <https://www.youtube.com/watch?v=ZcliKgilwMY>

Introducción

Para este trabajo práctico elegimos el dominio *Paciente* → *HistoriaClinica*. El objetivo final es tratar de replicar lo más fiel posible como funcionaría en la vida real.

Etapa 1 – Modelado y Definición de Constraints

- 1) Se adjunta archivo en PDF (Diagrama ER.pdf)
- 2) Evidencia del uso de la IA en Etapa 1.

Interacción con IA

Pregunta:

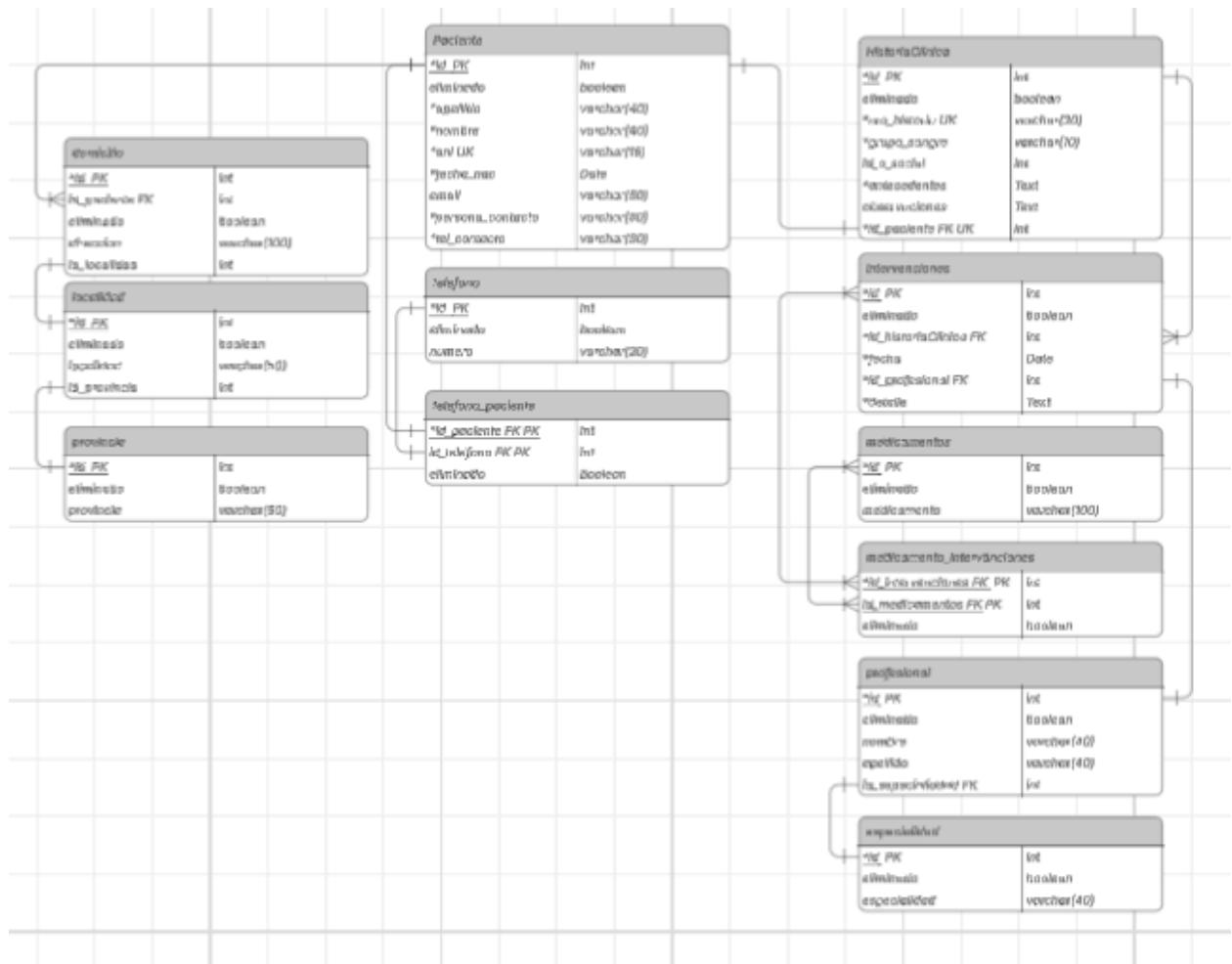
Estás interactuando con estudiantes de la materia Base de Datos I de una Tecnicatura Superior. que están resolviendo un Trabajo Práctico sobre modelos entidad–relación y modelo relacional.

Instrucciones para la IA

1. No des la solución completa de manera directa.
2. Ofrece pistas graduales que guíen al estudiante hacia la respuesta correcta.

3. Señala los errores conceptuales o de sintaxis de forma clara y respetuosa.
4. Explica los motivos de cada corrección y, cuando corresponda, da ejemplos similares que ayuden a entender mejor.
5. Si el estudiante se queda bloqueado, sugiere pasos intermedios o preguntas disparadoras.
6. Utiliza un lenguaje accesible, evitando tecnicismos innecesarios.

En base a lo anterior, te pasamos el diagrama ER para que nos hagas observaciones sobre las claves primarias y foráneas elegidas, los campos, los tipo de datos y si estaría normalizada la BD.



Respuesta:

¡Hola! Excelente trabajo al armar este Diagrama Entidad-Relación (DER). Es un modelo complejo que abarca varias áreas de un sistema de gestión de salud.

Como estamos en Base de Datos I, me enfocaré en las **bases conceptuales** de tu modelo relacional. No te daré las respuestas directas, sino **pistas y preguntas clave** para que revises y optimices tu diseño.

1. Observaciones sobre Claves Primarias (PK) y Foráneas (FK)

💡 Pista sobre Claves Primarias (PK)

Mira las claves primarias que has definido (los campos con `*Id_PK` en la mayoría de las tablas).

- **Pregunta Clave:** ¿Es necesario que todos los campos PK sean de tipo `Int`? Si bien es una práctica común, ¿has considerado si en tu modelo existen entidades que por su naturaleza puedan usar una **clave natural** (un campo ya existente en la entidad que es único e inmutable) en lugar de una clave subrogada (`Id` autoincremental)?
- **Recomendación:** Para la mayoría de tus entidades, usar un `Int` autoincremental (como `Id_PK`) es correcto y eficiente. Sin embargo, revisa la tabla `Paciente`.

🔗 Pista sobre Claves Foráneas (FK) y Relaciones

Observa la relación entre `Paciente` e `HistoriaClinica`.

- **Relación:** Has definido una relación 1:1 (uno a uno) usando el campo `*id_paciente FK UK` en `HistoriaClinica`.
- **Pregunta Clave:** Cuando tienes una relación 1:1 entre la tabla A (`Paciente`) y la tabla B (`HistoriaClinica`), ¿en qué tabla es conceptualmente **más apropiado** colocar la Clave Foránea (FK) y por qué? ¿Qué pasaría si quisieras que un paciente pudiera tener múltiples registros de historia clínica (un escenario más común)?

- **Concepto a Revisar:** La cláusula `UK` (Unique Key) que has puesto en `*id_paciente FK UK` garantiza el 1:1. Piensa si es la **única** forma de modelar esa unicidad.
-

2. Observaciones sobre Campos y Tipos de Datos

Revisemos algunos campos específicos para asegurar que la elección del tipo de dato y su rol sea óptima.

Campo eliminado

Has incluido un campo `eliminado` de tipo `boolean` en casi todas tus tablas (Paciente, Domicilio, Localidad, etc.).

- **Pregunta Clave:** ¿Cuál es el propósito de este campo? ¿Es para realizar una eliminación física (borrar el registro) o una **eliminación lógica** (marcar el registro como no visible, pero mantenerlo por si acaso)?
- **Pista:** Si el objetivo es la eliminación lógica (muy común en sistemas de salud), el uso del `boolean` es **correcto**. Simplemente ten claro que esto afecta a **cada consulta** que hagas, ya que siempre deberás añadir la condición `WHERE eliminado = false`.

Tipos de Datos y Longitudes

Mira los campos de texto (`varchar` y `Text`).

- **Tabla Paciente:** Tienes campos como `apellido` y `nombre` con `varchar(40)`.
- **Pregunta Clave:** ¿Son 40 caracteres suficientes para un nombre o apellido completo en el mundo real? ¿O te arriesgas a truncar nombres compuestos muy largos?
- **Pista (Textos):** En la tabla `HistoriaClinica`, el campo `observaciones` es de tipo `Text`. Esto es **correcto** para almacenar grandes cantidades de texto. Sin embargo, revisa el campo `antecedentes` que también has marcado como `Text`. ¿Es una cadena de texto larga o podría modelarse de otra forma para facilitar la búsqueda?

3. Observaciones sobre Normalización

Normalizar es esencial para evitar redundancia y anomalías en la base de datos. Analicemos dos tablas.

Tabla Intervenciones

Esta tabla parece registrar las intervenciones médicas.

- **Campos:** *id_profesional FK, *id_historiaClinica FK, *fecha, *detalle.
- **Pregunta Clave:** La `Intervenciones` parece depender de la `HistoriaClinica` (la llave compuesta `id_intervenciones + id_historiaClinica`). ¿Qué nivel de normalización estás aplicando aquí? ¿Es esta tabla la que tiene la responsabilidad de registrar al **Profesional** que realiza la intervención?

Tabla Medicamento_Intervenciones

Esta tabla es una tabla de relación (many-to-many) entre `Medicamentos` e `Intervenciones`.

- **Pista:** Las tablas de relación (o tablas puente) generalmente deben tener como clave primaria compuesta las **dos Claves Foráneas** que unen. Tu tabla tiene un `*Id_PK` propio. Aunque no es un error, es un diseño subóptimo en este caso. La clave primaria debería ser compuesta por:
- (`id_intervenciones FK,id_medicamentos FK`)
 - **¿Por qué?** Porque la combinación de una intervención y un medicamento específico **es única** y define la fila.

Tu **siguiente paso** es revisar estos puntos y hacer las correcciones que consideres necesarias en tu diagrama. Luego, discutiremos cómo quedó la normalización. ¡Adelante!

Pregunta:

En cuanto a la clave primaria de HC, la hemos reemplazado directamente por la clave foránea de id_paciente, haciéndola clave primaria y única para asegurar una relación 1:1.

En cuanto a eliminado como boolean es para hacer eliminación lógica. Modificamos la longitud de nombre y apellido a 50, y los campos tipo texto son para almacenar como una historia del paciente, de sus afecciones anteriores o de sus ascendientes.

Una HC puede tener muchas intervenciones, y cada intervención puede ser realizada por el mismo profesional u otro profesional distinto. Además, HC_Intervenciones es una relación muchos a muchos.

Por último, la tabla medicamentos_intervenciones no tiene una PK propia, es una PK compuesta por la PK de medicamentos y la PK de intervenciones. Fijate bien

Respuesta:

¡Excelentes ajustes y aclaraciones! 🌟 Es genial que tengas claros los conceptos de eliminación lógica y la cardinalidad de tus relaciones.

Vamos a revisar tus decisiones y a plantear las siguientes pistas:

1. Revisión de Claves Primarias (PK) y Foráneas (FK)

A. HistoriaClinica (HC) y la Relación 1:1 🎯

- Tu Solución: Reemplazar Id_PK de HistoriaClinica por Id_paciente FK y hacerlo PK y UK a la vez.
- Análisis:
 - Corrección Conceptual: ¡Es correcto! Para forzar una relación 1:1 donde la existencia de B (HC) depende de A (Paciente), puedes usar la clave primaria de A (Id_paciente) como la clave primaria de B (HistoriaClinica). Esto garantiza automáticamente que: 1) Cada HC pertenece a exactamente un Paciente. 2) Un Paciente solo puede tener una HC (porque Id_paciente es la PK de HC).
 - Pista: Asegúrate de que, al implementar esto, el campo Id_paciente en HistoriaClinica NO acepte valores nulos, ya que es la PK de esa tabla.

B. Medicamento_Intervenciones (Tabla Puente) 🌉

- Tu Aclaración: Afirmas que la clave primaria es compuesta por (Id_intervenciones FK, Id_medicamentos FK).

- Análisis del Diagrama: Al revisar tu diagrama, tienes un campo *Id_PK dentro de Medicamento_Intervenciones con una llave, lo que visualmente sugiere una clave subrogada propia (un Id autoincremental).
 - Pista de Sintaxis/Notación: Si tu intención es usar la clave compuesta (Id_intervenciones FK, Id_medicamentos FK), debes remover el *Id_PK autogenerado del diagrama o, al menos, marcar las dos FK como parte de la clave primaria compuesta (normalmente con un asterisco * o subrayado). Revisa la notación en esta tabla para asegurarte de que represente tu intención de usar la clave compuesta.
-

2. Pista sobre Normalización y Dependencias

Hablemos de la tabla Intervenciones y la información sobre el profesional.

- Tu Aclaración: "Una HC puede tener muchas intervenciones, y cada intervención puede ser realizada por el mismo profesional u otro profesional distinto." (Relación 1:M entre HC e Intervenciones y 1:M entre Profesional e Intervenciones). Esto es correcto y está bien modelado.
- El Desafío (Tercera Forma Normal - 3NF): Revisa la tabla Intervenciones. ¿Qué pasa si una Intervención fuera la consulta o turno con el profesional?
 - Campos de Intervenciones: Id_PK, Id_historiaClinica FK, Id_profesional FK, fecha, detalle.
 - Pregunta Clave: ¿Depende el campo detalle (que describes como el registro de la intervención) directamente y solo de la clave primaria Id_PK de Intervenciones? Parece que sí.
 - Pista para 3NF: Revisa si tienes alguna dependencia transitiva en cualquiera de tus tablas. Una dependencia transitiva ocurre cuando un campo que no es clave determina el valor de otro campo que tampoco es clave. *Ejemplo:* Si tuvieras el campo **especialidad** en la tabla **Profesional**, y **especialidad** dependiera solo de **id_especialidad FK**, tendrías un problema de 3NF. (En tu diagrama no lo tienes, pero es el concepto a buscar).

Tu diseño parece estar en Segunda Forma Normal (2NF). ¿Puedes identificar alguna potencial repetición de datos o dependencia transitiva que impida alcanzar la Tercera Forma Normal (3NF) en alguna de tus tablas? Piénsalo bien.

3. Pista de Diseño: Entidad Telefono

Observa las tablas **telefono** y **telefono_paciente**.

- Tabla Telefono: Almacena la PK y el número.
- Tabla Telefono_paciente: Relaciona **id_paciente FK** con **id_telefono FK**.
- Pregunta Clave: ¿Es posible que dos pacientes distintos comparten exactamente el mismo número de teléfono? Y, si es así, ¿sería mejor que la tabla **telefono** guardara la información de contacto de *cualquier* persona (paciente o contacto de emergencia) o solo números de pacientes?
- Recomendación: Este modelo de separar Telefono como una entidad independiente es muy flexible y permite que un paciente tenga varios teléfonos y que un teléfono se comparta. Es una buena elección de modelado. Simplemente, asegúrate de que la clave de **Telefono_paciente** también sea compuesta (**id_paciente FK,id_telefono FK**) para que un mismo paciente no asigne el mismo teléfono dos veces.

Pregunta:

En medicamentos_intervenciones no tenemos ninguna clave id PK, esa está en medicamentos. La tabla medicamentos_intervenciones es una tabla intermedia para la relación muchos a muchos entre medicamentos e intervenciones.

No podemos identificar ninguna relación transitiva en las tablas propuestas.

Con respecto a teléfono, la tabla telefono_paciente tiene una clave compuesta.

Respuesta:

¡Perfecto! Tus aclaraciones son muy valiosas y demuestran un sólido entendimiento de las relaciones M:N (Muchos a Muchos) y las claves compuestas.

Revisemos y ajustemos los puntos con base en tus últimas respuestas.

1. Revisión de Clave Compuesta y Diagrama

A. Medicamento_Intervenciones

- Tu Afirmación: La clave primaria es la compuesta (**Id_intervenciones FK,Id_medicamentos FK**), y no hay un **Id_PK** propio.

- Análisis del Diagrama (Observación Crítica): Tienes razón conceptualmente, esa es la forma óptima de modelar una tabla puente. Sin embargo, el diagrama que proporcionaste SÍ muestra un campo *Id_PK en la tabla medicamento_intervenciones.
- Pista: En un examen de Base de Datos I, la interpretación del diagrama es clave. Debes eliminar visualmente el campo *Id_PK y asegurarte de que ambas FK (id_intervenciones FK y id_medicamentos FK) estén subrayadas o marcadas como PK para representar correctamente la clave primaria compuesta (PK1,PK2). *Tu concepto es correcto, pero tu notación en el diagrama es contradictoria.*

B. Telefono_Paciente

- Tu Afirmación: Telefono_paciente tiene una clave compuesta.
- Análisis del Diagrama: El diagrama muestra el campo *id_paciente FK PK y id_telefono FK (sin PK/UK).
- Pista: Para que sea una clave compuesta correcta, ambos campos deben formar la PK. La clave primaria compuesta debe ser:
(id_paciente FK, id_telefono FK)
Esto garantiza que un paciente no pueda tener el mismo número de teléfono registrado dos veces. Asegúrate de marcar ambos en el diagrama (o al menos incluir *id_telefono FK PK junto con *id_paciente FK PK) para reflejar tu intención.

2. Pista sobre la Normalización (Dependencia Transitoria)

Afirmas que no identificas dependencias transitivas (lo que te pondría en 3NF). Analicemos la tabla Profesional para una última revisión.

Tabla Profesional

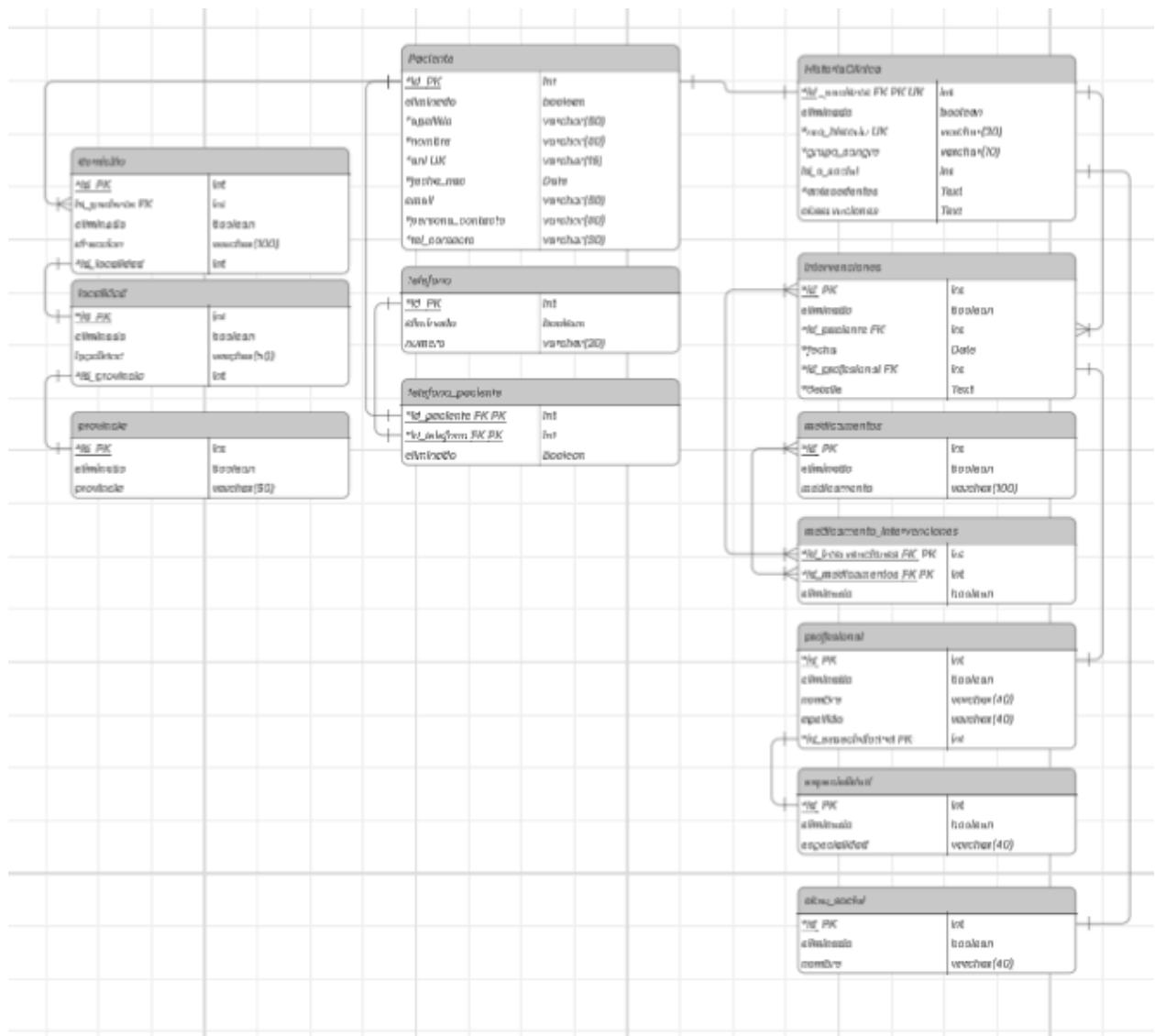
- Campos: Id_PK, eliminado, nombre, apellido, id_especialidad FK.
- Pregunta Clave: ¿Depende el nombre o apellido del profesional directamente de la clave primaria Id_PK? Sí.
- Pregunta de 3NF (La Trampa): ¿Podría existir una dependencia transitiva en esta estructura?
 - Observa la relación entre Profesional y Especialidad. Profesional "tiene" una Especialidad por medio de id_especialidad FK.
 - Si hubieras incluido el campo especialidad (el nombre textual, ej. 'Cirujano') directamente en la tabla Profesional (en lugar de la FK id_especialidad), ¿tendrías una dependencia transitiva? Sí, porque especialidad dependería de id_especialidad FK (un campo no clave), y no directamente de Id_PK.

- Conclusión: Como has modelado Especialidad en una tabla separada y usas solo el id_especialidad FK en Profesional, tu diseño SÍ cumple con la Tercera Forma Normal (3NF).

Respuesta Final: Con la estructura actual, asumiendo que las claves compuestas se implementan como indicas (PKs en tablas puente y Telefono_Paciente), y dada la separación de Especialidad, el modelo está normalizado en 3NF. Solo tienes que corregir la notación de las claves compuestas en tu diagrama.

Pregunta:

Te pasamos el diagrama como quedó para que hagas una última revisión general.



Respuesta:

¡Perfecto! El diagrama final luce muy bien. Las correcciones que hiciste mejoraron significativamente la **integridad** y la **normalización** del modelo.

Aquí tienes un resumen de los puntos fuertes y una última recomendación de diseño para la tabla Profesional.

1. Puntos Fuertes del Modelo Final

1. **Normalización (3NF)**: Al separar Especialidad y usar la clave foránea (id_especialidad FK) en Profesional, y al usar las tablas puente para las relaciones M:N, has logrado un diseño que cumple con la **Tercera Forma Normal (3NF)**.
 2. **Relación 1:1**: El uso de id_paciente FK PK en HistoriaClinica es la forma más limpia de implementar una relación 1:1 (donde la HC no existe sin el Paciente).
 3. **Claves Compuestas**: La notación de la clave compuesta en **medicamento_intervenciones** y **telefono_paciente** ahora es correcta, ya que ambas FKs están marcadas como PKs. Esto es óptimo para las tablas de relación.
 4. **Unicidad**: El uso de dni UK en Paciente garantiza la unicidad del documento.
 5. **Eliminación Lógica**: La inclusión del campo eliminado (**boolean**) en todas las tablas principales es una buena política de negocio para conservar la integridad histórica de los datos.
-

2. Última Sugerencia de Diseño (Tabla Profesional)

Tu tabla Profesional está bien, pero hay un pequeño detalle en la FK:

- **Tabla Profesional**: Tienes id_especialidad FK.
- **Pista**: Revisa la tabla Intervenciones (que has renombrado a informes en el diagrama final). La tabla Intervenciones apunta al Profesional que realizó la acción con id_profesional FK.

El Problema a Considerar (Si el Profesional puede tener Múltiples Especialidades):

En tu diseño actual, un Profesional solo puede tener **una** Especialidad (por la relación 1:M con la FK en Profesional).

- **Si la regla de negocio es:** Un profesional **solo** puede tener una especialidad primaria → **Tu diseño es correcto.**
 - **Si la regla de negocio es:** Un profesional **puede** tener **varias** especialidades (ej. Cardiólogo y Clínico) → **Deberías crear una tabla puente (M:N).**

Entidad	Campos Sugeridos
Profesional_Especialidad	*id_profesional FK PK *id_especialidad FK PK

Exportar a Hojas de cálculo

Si tu modelo solo permite una especialidad, el diseño actual es **definitivo y correcto**. Si necesitas múltiples especialidades, considera añadir esa tabla M:N.

¡Excelente trabajo final! El modelo está listo para ser traspasado al Modelo Relacional e implementado en SQL.

4) Validación práctica: ejecutar 2 inserciones correctas y 2 erróneas mostrando mensajes de error distintos (ej. violación de UNIQUE y CHECK o FK).

```
INSERT INTO paciente (apellido, nombre, dni, fecha_nac, persona_contacto, telefono_contacto)
VALUES ('Ramírez', 'Lucía', '41222333', '1995-07-15', 'Jose Marin', '35134445454');
```

1 • `SELECT * FROM pacientehistoriaclinica.paciente;`

```
INSERT INTO historiaClinica (id_paciente, nro_historia, grupo_sangre, id_o_social, antecedentes, observaciones)
VALUES (1, 'HC200', 'O-', 1, 'Sin antecedentes relevantes', 'Paciente estable');
```

1 • `SELECT * FROM pacientehistoriaclinica.historiaclinica;`

```
INSERT INTO paciente (apellido, nombre, dni, fecha_nac, persona_contacto,  
telefono_contacto)  
VALUES ('Perez', 'Marcelo', '41222333', '1955-09-17', 'Pablo Picapiedra',  
'32334445454');
```

```
5 •  INSERT INTO paciente (apellido, nombre, dni, fecha_nac, persona_contacto, telefono_contacto)
6   VALUES ('Perez', 'Marcelo', '41222333', '1955-09-17', 'Pablo Picapiedra', '32334445454');
```

Output					
#	Time	Action	Message		
58	17:44:53	INSERT INTO historiaClinica (id_paciente, nro_historia, grupo_sangre, id_o_social, antecedentes, obse...	1 row(s) affected		
59	17:45:02	SELECT * FROM pacientehistoriaclinica.historiaclinica	1 row(s) returned		
60	17:47:07	INSERT INTO paciente (apellido, nombre, dni, fecha_nac, persona_contacto, telefono_contacto) VAL...	Error Code: 1062. Duplicate entry '41222333' for key 'paciente.dni'		

INSERT INTO paciente (apellido, nombre, dni, fecha_nac, persona_contacto,
telefono_contacto)
VALUES ('Perez', 'Marcelo', '412333', '2027-09-17', 'Pablo Picapiedra', '32334445454');

```
8 •  INSERT INTO paciente (apellido, nombre, dni, fecha_nac, persona_contacto, telefono_contacto)
9   VALUES ('Perez', 'Marcelo', '412333', '2027-09-17', 'Pablo Picapiedra', '32334445454');
```

Output					
#	Time	Action	Message		
59	17:45:02	SELECT * FROM pacientehistoriaclinica.historiaclinica	1 row(s) returned		
60	17:47:07	INSERT INTO paciente (apellido, nombre, dni, fecha_nac, persona_contacto, telefono_contacto) VAL...	Error Code: 1062. Duplicate entry '41222333' for key 'paciente.dni'		
61	17:50:13	INSERT INTO paciente (apellido, nombre, dni, fecha_nac, persona_contacto, telefono_contacto) VAL...	Error Code: 1644. La fecha de nacimiento no puede ser una fecha futura.		

Etapa 2. Generación de datos masivos con SQL puro

- 1) Se adjunta Script SQL para la generación masiva 03_esquema.sql.
- 2) Resultados de las mediciones con/sin índices.
- 3) Evidencia del proceso (chat + producto final).

Interacción con IA

Pregunta:

Vamos a seguir interactuando como empezamos, considerando que somos un grupo de estudiantes de Bases de Datos I de la Tecnicatura Universitaria en Programación, y queremos que nos ayudes a completar nuestra base de datos con datos ficticios a través de procedimientos sql. , Te paso nuevamente el diagrama ER para que veas cuales son las tablas que debemos completar con 10000 registros ficticios.

No nos des el código, danos pistas de como hacerlo y tampoco queremos que sea Medicamento_001 sino mas bien con nombres específicos

Respuesta:

¡Absolutamente! Es mucho mejor que ustedes mismos construyan el código. Les daré una hoja de ruta con pistas y funciones clave para generar datos realistas sin usar nombres genéricos como "Medicamento_001".

El desafío de generar 10,000 registros ficticios con nombres específicos se resuelve con la pre-carga de datos de catálogo y el uso de funciones de selección aleatoria.

Pistas y Estrategia para Generar 10,000 Registros Ficticios

1. Pre-Carga de Catálogos (Nombres Específicos)

Antes de empezar cualquier procedimiento con bucles, deben tener listas de nombres reales para usar. Esto es clave para reemplazar el `CONCAT('Medicamento_', i)`.

Acciones Recomendadas:

1. Crear Tablas Temporales o Cargar Catálogos Iniciales: Creen un *script* de inserción inicial para las tablas de catálogo con nombres reales.
 - Para **medicamentos**: Inserten nombres de fármacos comunes (ej., "Ibuprofeno", "Amoxicilina", "Paracetamol", etc.).
 - Para **provincias, obra_social, especialidad**: Inserten entre 5 y 20 registros reales (ej., "Cardiología", "OSDE", "Buenos Aires").
 - Para **persona** (Nombres y Apellidos): Creen una tabla temporal llamada **Nombres_Ficticios** y otra **Apellidos_Ficticios** con al menos 100 nombres y 100 apellidos diferentes.
2. Generación de *Strings* Realistas: Para campos como **calle** o **localidad**, combinen valores aleatorios de listas reales.
 - Pista para Domicilios: Podrían tener una lista de tipos de calle ("Av.", "Calle", "Pje.") y luego concatenar uno de esos tipos con un nombre aleatorio (ej., "Av. San Martín").

2. Generación de Valores Aleatorios (SQL Funciones Clave)

Para insertar 10,000 registros dentro de un procedimiento almacenado (**STORED PROCEDURE**), necesitarán usar funciones de aleatoriedad para:

Propósito	Función (MySQL/SQL Server)	Pista de Uso
Ejemplo)		

Bucle Principal **WHILE i <= 10000 DO ...** Usen un contador simple (*i*) para controlar (10,000 veces) **SET i = i + 1;** la cantidad de inserciones.

Claves Foráneas (FK) **SELECT id_pk FROM tabla ORDER BY RAND() LIMIT 1** Esta es la pista más importante. Para **id_domicilio_FK** en la tabla **persona**, no elijan un número fijo, sino selecciona una ID existente al azar de la tabla **domicilios**.

Fechas (ej. **fecha_nac**) **DATE_ADD(CURDATE(), INTERVAL - FLOOR(1 + RAND() * 30000) DAY)** Resta una cantidad aleatoria de días a la fecha actual para obtener fechas históricas realistas. (30,000 días son unos 82 años).

Booleanos `FLOOR(RAND() * 2)` Genera 0 o 1 de forma aleatoria.
(habilitado)

Números de 8 dígitos `LPAD(FLOOR(RAND() * 99999999), 8, '0')` * Combina `RAND()` con `LPAD` (llenar a la izquierda) para generar DNI o números de teléfono con un formato consistente.

Seleccionar Nombres `(SELECT nombre FROM Nombres_Ficticios ORDER BY RAND() LIMIT 1)` Clave para nombres reales. Utilicen una subconsulta para obtener un nombre aleatorio de su lista pre-cargada.

Exportar a Hojas de cálculo

3. Orden de Inserción y Bucles (Estructura Lógica)

Deben escribir un procedimiento almacenado para cada grupo de tablas siguiendo este orden:

1. Catálogos Simples: Insertar 10 registros en `obra_social`, `especialidad`, `provincias`, y 100 en `medicamentos`.
2. Bucles para Base de Población (10,000):
 - o Procedimiento 1: `localidades`
 - Dentro del bucle, seleccionar un `id_provincia_FK` aleatorio.
 - o Procedimiento 2: `domicilios`
 - Dentro del bucle, seleccionar un `id_localidad_FK` aleatorio.
 - o Procedimiento 3: `persona` (El más complejo)
 - Dentro del bucle, seleccionar un `id_domicilio_FK` aleatorio y un `id_obra_social_FK` aleatorio.
 - Usar subconsultas para seleccionar nombres y apellidos aleatorios de sus tablas de apoyo.
3. Bucles para Eventos (2,000):
 - o Procedimiento 4: `historiaclinicainterconsulta`
 - Dentro del bucle, seleccionar un `id_paciente_FK` y un `id_profesional_FK` aleatorios.

¡Consejo Final! Si están usando MySQL, recuerden usar el delimitador (`DELIMITER //`) al inicio de cada procedimiento y el `CALL` para ejecutarlo. ¡Manos a la obra!

Pregunta

¿Por cual tabla debemos comenzar?

Respuesta

Deben comenzar por las **tablas de catálogo (independientes)**. 

Estas tablas no tienen claves foráneas (FK) que apunten a ninguna de las otras tablas principales del diagrama, lo que significa que pueden insertar datos en ellas sin violar ninguna restricción de integridad referencial. Sus claves primarias (PK) serán necesarias para empezar a poblar las tablas más grandes.

1. Tablas por donde Empezar (Catálogos)

Comiencen insertando un número razonable de registros (generalmente menos de 100) con nombres específicos y realistas en las siguientes tablas:

1. **provincias**
2. **obra_social**
3. **especialidad**
4. **medicamentos**

Pista para estas tablas

La clave aquí no es generar 10,000 registros, sino generar un **catálogo de valores reales** para que las siguientes tablas puedan hacer referencia a ellos de forma creíble. Por ejemplo, inserten 10-15 provincias o 100 medicamentos con nombres específicos.

2. El Siguiente Paso Lógico (Jerarquía de Dependencia)

Una vez que tengan las PK de los catálogos, el orden lógico de inserción debe seguir la jerarquía de dependencias:

Paso	Tablas a Insertar	Dependencias (FK) Requeridas
A	localidades	Necesita PK de provincias .
B	domicilios	Necesita PK de localidades .
C	profesional	Necesita PK de obra_social y especialidad .
D	persona (El paciente)	Necesita PK de domicilios y obra_social .

persona es la tabla que agrupa la mayor cantidad de información y que luego será referenciada por las tablas de eventos ([historiaclinicainterconsulta](#), [intervenciones](#)). Por lo tanto, después de los catálogos, la prioridad es poblar las tablas A, B, y C para poder generar los 10,000 pacientes en la tabla D (**persona**) de forma correcta.

Pregunta:

¿Cómo se hace un procedimiento en sql?

Respuesta:

Un procedimiento almacenado en SQL (a menudo llamado *Stored Procedure* o SP) es un **conjunto de instrucciones SQL precompiladas** que se guardan en el servidor de la base de datos para ser ejecutadas posteriormente.

Aquí te muestro la estructura general y las partes clave de cómo se hace un procedimiento, usando como ejemplo la sintaxis más común (MySQL o SQL Server/Transact-SQL).

1. Estructura General del Procedimiento

El código de un procedimiento almacenado se compone de tres elementos fundamentales: la declaración, los parámetros y el cuerpo.

Sintaxis (Ejemplo en MySQL)

En MySQL, debes usar la instrucción `DELIMITER` para redefinir el terminador de sentencias (normalmente `;`) a algo diferente (como `$$` o `//`). Esto permite que el cuerpo del procedimiento contenga múltiples sentencias SQL terminadas en `;`.

```
SQL
DELIMITER //
CREATE PROCEDURE nombre_del_procedimiento (
    -- Declaración de Parámetros (opcional)
    IN parametro1 TIPO_DATO,
    OUT parametro2 TIPO_DATO
)
BEGIN
    -- Cuerpo del Procedimiento (Instrucciones SQL)
    -- 1. Declaración de Variables Locales (si es necesario)
    DECLARE variable_local INT DEFAULT 0;

    -- 2. Lógica del Procedimiento
    SELECT * FROM tabla_ejemplo WHERE columna = parametro1;

    -- 3. Asignación a Parámetros de Salida (OUT)
    SET parametro2 = variable_local + 1;

END //
DELIMITER ; -- Volvemos a establecer el terminador a ;
```

2. Componentes Clave

A. El Delimitador (`DELIMITER`)

- **Propósito:** Es específico de MySQL. Dado que un procedimiento puede contener muchas sentencias SQL (cada una terminada en `;`), MySQL necesita saber dónde termina *todo el bloque* de la instrucción `CREATE PROCEDURE`.

- **Acción:** Se cambia temporalmente el terminador de sentencias de ; a // (o \$\$). Al finalizar, se vuelve a ;.

B. La Declaración (CREATE PROCEDURE)

- **Sintaxis:** CREATE PROCEDURE nombre_del_procedimiento.
- **Pista de Estudio:** Es recomendable usar DROP PROCEDURE IF EXISTS nombre_del_procedimiento antes de CREATE para evitar errores si intentas ejecutar el script varias veces.

C. Los Parámetros (Argumentos)

Los procedimientos pueden aceptar valores de entrada y devolver valores de salida.

Tipo	Propósito	Ejemplo
IN	El parámetro es para entrada . Se pasa un valor al procedimiento, y este no puede ser modificado dentro del SP.	IN num_registros INT
OUT	El parámetro es para salida . El SP puede modificar su valor, y el resultado es devuelto al programa o usuario que lo llamó.	OUT mensaje_estado VARCHAR(50)
INOUT	El parámetro es tanto de entrada como de salida.	

D. El Cuerpo (BEGIN y END)

El cuerpo contiene la lógica principal del procedimiento.

1. **Declaración de Variables (DECLARE):** Se usa para definir variables internas que solo existen durante la ejecución del procedimiento.
 - *Ejemplo:* DECLARE i INT DEFAULT 1;
2. **Asignación de Valores (SET):** Se usa para asignar un valor a una variable local o a un parámetro OUT.
 - *Ejemplo:* SET i = i + 1; o SET variable_local = (SELECT MAX(id) FROM tabla);

3. **Instrucciones de Control:** Es donde realmente se programan las acciones, incluyendo:
 - Consultas (SELECT, INSERT, UPDATE, DELETE).
 - Estructuras de control de flujo (IF/ELSE, WHILE, LOOP).

3. Ejecución del Procedimiento

Una vez creado y guardado en la base de datos, lo ejecutas con la instrucción CALL.

SQL

-- Para ejecutar un procedimiento sin parámetros:

`CALL nombre_del_procedimiento();`

-- Para ejecutar el procedimiento que usamos en el ejemplo:

`CALL nombre_del_procedimiento(10, @resultado);`

-- Y luego, para ver el resultado del parámetro OUT:

`SELECT @resultado;`

Etapa 3 – Consultas Avanzadas y Reportes

Características de las consultas:

Técnicas SQL utilizadas:

- JOIN múltiples (INNER, LEFT)
- GROUP BY con múltiples columnas
- HAVING para filtrar grupos
- Subconsultas (correlacionadas y no correlacionadas)
- Funciones de agregación (COUNT, AVG, MAX, MIN)
- CASE WHEN para lógica condicional
- DATE_FORMAT y funciones de fecha
- GROUP_CONCAT para concatenar resultados
- Subconsultas en SELECT

Reportes incluidos:

1. Pacientes por obra social con intervenciones
2. Top 10 profesionales más activos
3. Pacientes frecuentes (>5 intervenciones/año)
4. Medicamentos más recetados por especialidad
5. Pacientes inactivos (sin visitas en 6 meses)
6. Distribución por rango etario
7. Pacientes con antecedentes específicos
8. Resumen mensual de actividad
9. Pacientes por provincia/localidad
10. Análisis medicamentos vs antecedentes

3 Vistas útiles:

- [`v_pacientes_completo`](#): Info completa de pacientes
- [`v_estadisticas_profesional`](#): Rendimiento de profesionales

Interacción con la IA

Resumen de las preguntas y respuestas realizadas a la IA a modo de apoyo.

1. ¿Cómo puedo ver qué columnas tiene mi vista?

Respuesta: Ejecuta DESCRIBE `v_intervenciones_detalle`; o SHOW COLUMNS FROM `v_intervenciones_detalle`;.. Esto te mostrará todas las columnas disponibles y sus tipos de datos. Así podrías ver que la columna se llama `fecha_ordenamiento` y no `fecha`.

2. ¿Por qué mi vista solo devuelve una fila si tengo múltiples intervenciones?

Respuesta: Revisa si tu vista tiene un GROUP BY. Cuando agrupas datos con GROUP_CONCAT, todas las filas que comparten la misma clave se colapsan en una sola. Pregúntate: ¿realmente necesitas agrupar o quieres ver cada intervención por separado?

3. ¿Cómo obtener múltiples valores (como medicamentos) para una fila sin usar GROUP BY?

Respuesta: Usa una subconsulta correlacionada en el SELECT. Por ejemplo:

```
sql(SELECT GROUP_CONCAT(m.medicamento)
```

```
FROM medicamento_intervenciones mi
```

```
WHERE mi.id_intervencion = i.id) AS medicamentos
```

Esto te da los valores concatenados sin necesidad de agrupar toda la consulta.

4. ¿Cómo puedo debuggear qué está mal en mi consulta SQL cuando obtengo un error de columna desconocida?

Respuesta:

Lee el mensaje de error cuidadosamente (te dice qué columna falta)

Verifica los nombres exactos de las columnas con DESCRIBE tabla;

Revisa si la columna existe con un alias diferente

Comprueba si estás referenciando una vista en lugar de una tabla

ETAPA 4 – Seguridad e Integridad

Objetivo: Proteger PII/PHI, aplicar principio de mínimo privilegio y demostrar integridad y resistencia a inyección.

1) Vistas sanitizadas (ocultan PII/PHI)

A. v_pacientes_publico

Expone:

id_paciente (ID interno), nombre y apellido combinados, edad (derivada), obra social, cantidad y última fecha de intervención.

Oculta:

DNI, email, teléfonos, fecha_nac cruda, nro_historia, grupo_sangre, antecedentes, observaciones, domicilio.

Justificación: la vista cubre necesidades de listados/consultas sin entregar PII/PHI; la edad se deriva para no exponer fecha_nac.

B. v_intervenciones_publico

Expone:

id_intervencion, id_paciente (solo ID), fecha (DATE), profesional y especialidad, conteo de medicamentos.

Oculta:

Detalle clínico de la intervención y cualquier dato identificable del paciente más allá del ID interno.

Justificación: permite auditoría/estadística de actividad clínica sin revelar PHI del contenido del acto médico.

C. v_estadisticas_profesional_publico

Expone:

Profesional, especialidad, pacientes atendidos (conteo), total de intervenciones, recetas emitidas, primera/última atención.

Oculta:

Identidades individuales de pacientes.

Justificación: métricas agregadas sin PII/PHI, adecuadas para tableros/indicadores.

2) Usuario de mínimos privilegios (principle of least privilege)

Usuario: usuario_reporte

Permisos: SOLO SELECT sobre las vistas anteriores. Sin acceso a tablas base.

Pruebas de permisos (evidencia):

ÉXITO (debe funcionar):

```
SELECT * FROM pacienteHistoriaClinica.v_pacientes_publico LIMIT 5;
```

```
SELECT * FROM pacienteHistoriaClinica.v_intervenciones_publico LIMIT 5;
```

DENEGADO (debe fallar):

```
SELECT * FROM pacienteHistoriaClinica.paciente;
```

```
SELECT dni FROM pacienteHistoriaClinica.paciente;
```

Mensaje esperado (puede variar según cliente): ERROR 1142 (42000): SELECT command denied to user 'usuario_reporte' '@'...' for table 'paciente'

3) Pruebas de integridad (mínimo 2, se documentan 4)

A. UNIQUE (paciente.dni)

Acciones: insertar dos pacientes con el mismo DNI (véase script 04_pruebas_integridad.sql).

Evidencia esperada: ERROR 1062 (23000): Duplicate entry '...' for key 'paciente.uq_dni' (nombre de índice puede variar).

B. FOREIGN KEY (domicilio.id_localidad / historiaClinica.id_o_social)

Acciones: insertar domicilio con id_paciente e id_localidad inexistentes.

Evidencia esperada: ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (...).

C. CHECK (eliminado IN (0,1))

Acciones: insertar en especialidad/el resto de tablas con eliminado = 2.

Evidencia esperada: ERROR 3819 (HY000): Check constraint 'chk_eliminado' is violated (el nombre puede diferir).

D. Trigger de integridad temporal (fechas futuras)

Acciones: insertar intervención con fecha > CURDATE().

Evidencia esperada: ERROR 1644 (45000): (mensaje definido en SIGNAL del trigger).

4) Consulta segura (sin SQL dinámico) + prueba anti-inyección

Implementación elegida: Procedimiento almacenado
sp_buscar_intervenciones_por_paciente(p_id_paciente INT, p_desde DATE, p_hasta DATE, p_id_especialidad INT|NULL).

Seguridad por diseño:

Parámetros tipados (INT/DATE): el motor castea la entrada; una cadena como "1 OR 1=1 --" no se evalúa como SQL.

No hay SQL dinámico ni concatenación de strings.

Filtro por rango de fechas y especialidad opcional, con ORDER BY determinístico.

Prueba anti-inyección (evidencia):

Llamada “maliciosa”:

```
CALL sp_buscar_intervenciones_por_paciente('1 OR 1=1 --', '2024-01-01', '2025-12-31', NULL);
```

Resultado esperado:

El motor castea "1 OR 1=1 --" a entero (1) o rechaza el valor; en ningún caso se altera la cláusula WHERE.

La salida coincide con pasar 1 como ID válido, o se recibe un error de casteo. En ningún caso devuelve “todas” las filas.

Alternativa (Java/JDBC): PreparedStatement con setLong setDate produce el mismo efecto (datos separados de la consulta).

Checklist de entregables (para el TFI)

[] 04_vistas_seguras.sql (DROP/CREATE de v_pacientes_publico, v_intervenciones_publico, v_estadisticas_profesional_publico).

[] 04_usuario_permisos.sql (CREATE USER, GRANT solo sobre vistas, pruebas OK/DENEGADO).

[] 04_pruebas_integridad.sql (bloques A–D con ejecución individual y capturas de error del motor).

[] 04_consulta_segura.sql (SP y llamadas de prueba, incluyendo anti-inyección).

[] Capturas/logs: resultados de SELECT sobre vistas con usuario_reporte y errores por acceso a tablas base.

[] Breve explicación: por qué las columnas ocultas son PII/PHI y cómo se respeta mínimo privilegio.

[] Anexo: esta interacción IA y versión de MySQL usada.

Orden sugerido de ejecución

1) 04_vistas_seguras.sql

2) 04_usuario_permisos.sql

3) 04_pruebas_integridad.sql (bloques A, B, C, D por separado)

4) 04_consulta_segura.sql (probar llamadas normales y anti-inyección)

Conclusión

Las vistas sanitizadas limitan la exposición de PII/PHI, el usuario usuario_reporte aplica mínimo privilegio, las pruebas evidencian integridad (UNIQUE, FK, CHECK, triggers) y el SP parametrizado demuestra resistencia frente a inyección. Con las capturas correspondientes, el Punto 4 queda completo conforme a la rúbrica.

Interacción con IA

Objetivo:

Aplicar principio de mínimo privilegio; ocultar PII/PHI vía vistas; demostrar integridad y consulta segura resistente a inyección.

Preguntas guía del estudiante:

¿Qué columnas son PII/PHI y deben ocultarse?

¿Cómo crear un usuario con acceso sólo a vistas?

¿Qué pruebas de integridad son suficientes y fáciles de demostrar?

¿Cómo implementar una consulta segura sin SQL dinámico?

Extracto de interacción (IA como tutor):

Estudiante: Quiero exponer listados sin DNI/email/fecha de nacimiento.

IA Tutor: Creá `v_pacientes_publico` con edad derivada y sin PII; evitá exponer detalle clínico en vistas públicas.

Estudiante: ¿Usuario de lectura de mínimo privilegio?

IA Tutor: Definí `app_report` con GRANT SELECT sólo sobre las vistas; sin permisos sobre tablas base.

Estudiante: Para integridad ¿qué pruebo?

IA Tutor: Mínimo UNIQUE (dni), FK inválida, CHECK (eliminado), y trigger de fecha futura.

Estudiante: Consulta segura contra inyección.

IA Tutor: Procedimiento almacenado con parámetros tipados (INT/DATE), sin SQL dinámico: `sp_buscar_intervenciones_por_paciente`.

Acciones realizadas por el estudiante (a partir de la tutoría):

Se crearon vistas sanitizadas: `v_pacientes_publico`, `v_intervenciones_publico`, `v_estadisticas_profesional_publico`.

Se creó el usuario `app_report` con SELECT exclusivo sobre vistas. Se verificó acceso denegado a tablas base.

Se ejecutaron pruebas de integridad: UNIQUE duplicado, FK inexistente, CHECK inválido, trigger temporal (fecha futura).

Se implementó SP seguro y se probó llamada “maliciosa” con cadena que no altera el WHERE.

Evidencia generada:

04_vistas_seguras.sql

04_usuario_permisos.sql

04_pruebas_integridad.sql

04_consulta_segura.sql (sp_buscar_intervenciones_por_paciente)

Punto_4_Informe.docx (resumen técnico y evidencias esperadas).

Reflexión y Buenas Prácticas

Rol de la IA: apoyo como tutor técnico para acelerar decisiones de diseño y buenas prácticas de seguridad e integridad.

Limitaciones: la IA no reemplaza validación empírica; todas las propuestas se verificaron con scripts y mensajes del motor.

Responsabilidad académica: el estudiante integra, adapta y valida; el código entregado es resultado propio guiado por tutoría.

Etapa 5. Concurrencia y transacciones

Parte 1: Simular bloqueos y deadlocks

Ejemplo simple con dos sesiones (puedes probar en dos ventanas de MySQL Workbench o terminal):

Sesión 1

START TRANSACTION;

UPDATE paciente SET nombre = 'Juan (bloqueado)' WHERE id = 1;

Sesión 2

START TRANSACTION;

UPDATE paciente SET nombre = 'Pedro (conflicto)' WHERE id = 2;

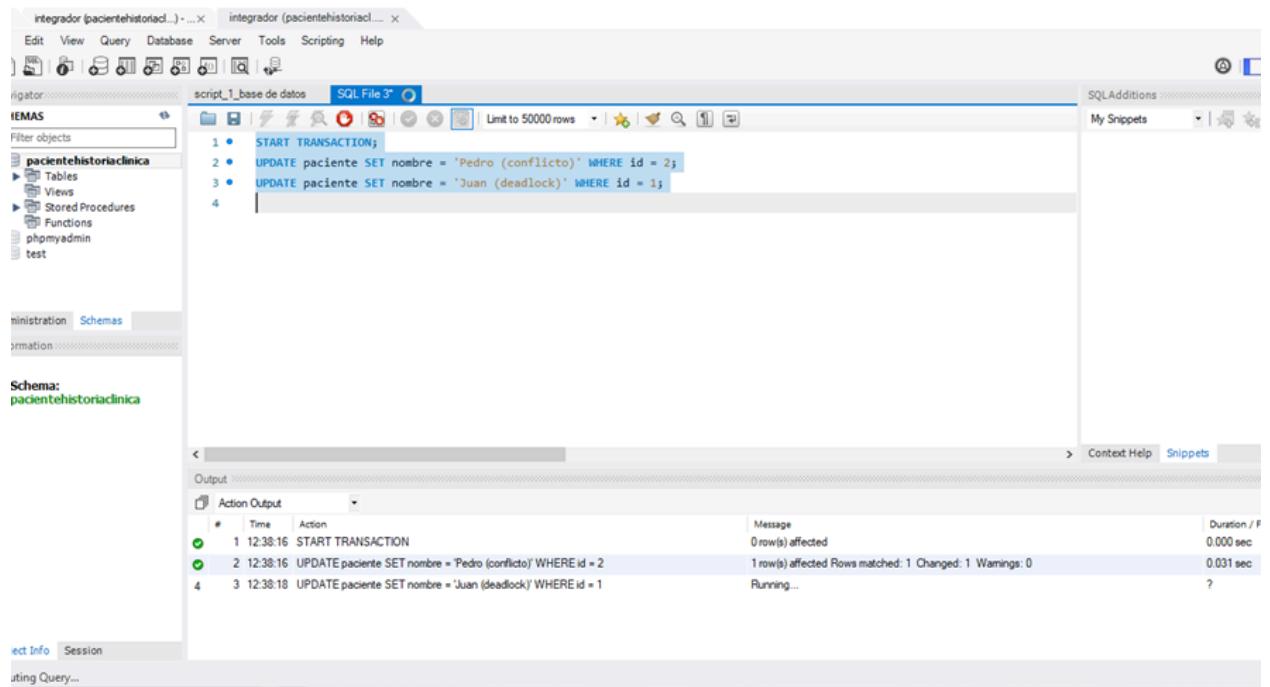
UPDATE paciente SET nombre = 'Juan (deadlock)' WHERE id = 1;

Sesión 1 (continúa)

UPDATE paciente SET nombre = 'Pedro (deadlock)' WHERE id = 2;

COMMIT;

Esto genera un deadlock (error 1213 o SQLSTATE 40001).



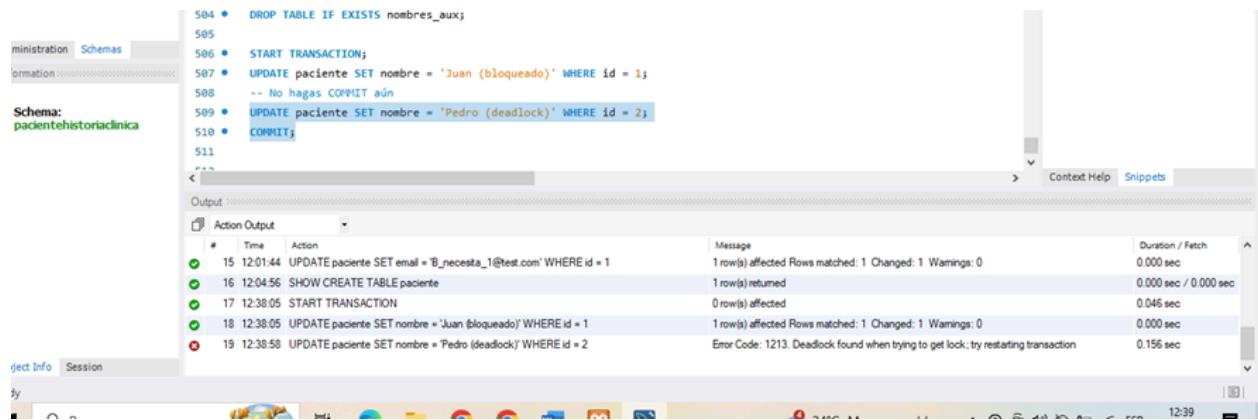
The screenshot shows the MySQL Workbench interface with two tabs open: 'integrador (pacientehistoriclinica)' and 'integrador (pacientehistoriclinica)'. The left pane displays the database schema with a single table named 'pacientehistoriclinica' containing 'Tables', 'Views', 'Stored Procedures', 'Functions', 'phpmyadmin', and 'test'. The right pane shows a script editor titled 'script_1_base de datos' with the following SQL code:

```
1 • START TRANSACTION;
2 • UPDATE paciente SET nombre = 'Pedro (conflicto)' WHERE id = 2;
3 • UPDATE paciente SET nombre = 'Juan (deadlock)' WHERE id = 1;
4 |
```

Below the script editor is the 'Output' pane, which displays the execution log:

Action	Time	Message	Duration / F
1	12:38:16	START TRANSACTION	0.000 sec
2	12:38:16	UPDATE paciente SET nombre = 'Pedro (conflicto)' WHERE id = 2	0.031 sec
3	12:38:18	UPDATE paciente SET nombre = 'Juan (deadlock)' WHERE id = 1	?

The log shows that the first update completed successfully, but the second update is still running, indicating a deadlock.



A screenshot of MySQL Workbench showing a transaction log. The code pane contains:

```
504 • DROP TABLE IF EXISTS nombres_aux;
505
506 • START TRANSACTION;
507 • UPDATE paciente SET nombre = 'Juan (bloqueado)' WHERE id = 1;
508 -- No hagas COMMIT aún
509 • UPDATE paciente SET nombre = 'Pedro (deadlock)' WHERE id = 2;
510 • COMMIT;
```

The output pane shows the execution history:

#	Time	Action	Message	Duration / Fetch
15	12:01:44	UPDATE paciente SET email = 'B_necesita_1@test.com' WHERE id = 1	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.000 sec
16	12:04:56	SHOW CREATE TABLE paciente	1 row(s) returned	0.000 sec / 0.000 sec
17	12:38:05	START TRANSACTION	0 row(s) affected	0.046 sec
18	12:38:05	UPDATE paciente SET nombre = 'Juan (bloqueado)' WHERE id = 1	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.000 sec
19	12:38:58	UPDATE paciente SET nombre = 'Pedro (deadlock)' WHERE id = 2	Error Code: 1213. Deadlock found when trying to get lock; try restarting transaction	0.156 sec

Error Code: 1213. Deadlock found when trying to get lock; try restarting transaction

Parte 2: Implementar retry ante deadlock (en SQL o Java)

Este procedimiento detecta el error 1213 y reintenta automáticamente hasta 2 veces.

DELIMITER \$\$

```
CREATE PROCEDURE ActualizarPacienteRetry(IN p_id INT, IN p_nombre VARCHAR(50))
```

BEGIN

DECLARE retries INT DEFAULT 0;

DECLARE exit_loop BOOL DEFAULT FALSE;

WHILE retries < 2 AND NOT exit_loop DO

BEGIN

DECLARE CONTINUE HANDLER FOR 1213 -- Deadlock

```
BEGIN  
  
    SET retries = retries + 1;  
  
    DO SLEEP(1); -- Espera breve antes del reintento  
  
END;  
  
START TRANSACTION;  
  
    UPDATE paciente SET nombre = p_nombre WHERE id = p_id;  
  
    COMMIT;  
  
    SET exit_loop = TRUE;  
  
END;  
  
END WHILE;  
  
END$$
```

DELIMITER ;

CALL ActualizarPacienteRetry(1, 'Juan Retried');

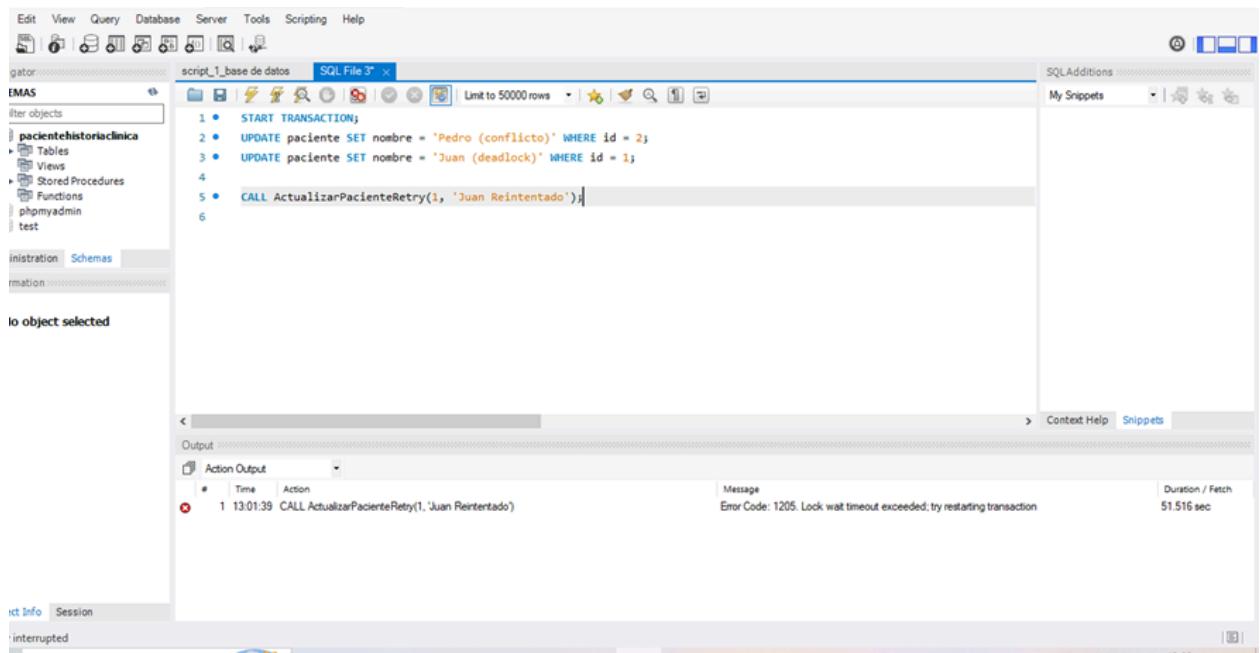
Ejemplo de prueba:

Sesión 1

```
START TRANSACTION;  
  
UPDATE paciente SET nombre = 'Juan bloqueado' WHERE id = 1;
```

Sesión 2

```
CALL ActualizarPacienteRetry(1, 'Juan Reintentado');
```



The screenshot shows the MySQL Workbench interface. In the top panel, a script named 'script_1_base de datos' is open, containing the following SQL code:

```
1 • START TRANSACTION;
2 • UPDATE paciente SET nombre = 'Pedro (conflicto)' WHERE id = 2;
3 • UPDATE paciente SET nombre = 'Juan (deadlock)' WHERE id = 1;
4
5 • CALL ActualizarpacienteRetry(1, 'Juan Reintentado');
6
```

In the bottom panel, the 'Output' tab displays the execution results:

#	Time	Action	Message	Duration / Fetch
1	13:01:39	CALL ActualizarpacienteRetry(1, 'Juan Reintentado')	Error Code: 1205. Lock wait timeout exceeded; try restarting transaction	51.516 sec

The status bar at the bottom indicates 'Session interrupted'.

Parte 3: Comparar niveles de aislamiento

Sesión 1 – READ COMMITTED

```
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

```
START TRANSACTION;
```

```
SELECT nombre FROM paciente WHERE id = 1;
```

Sesión 2

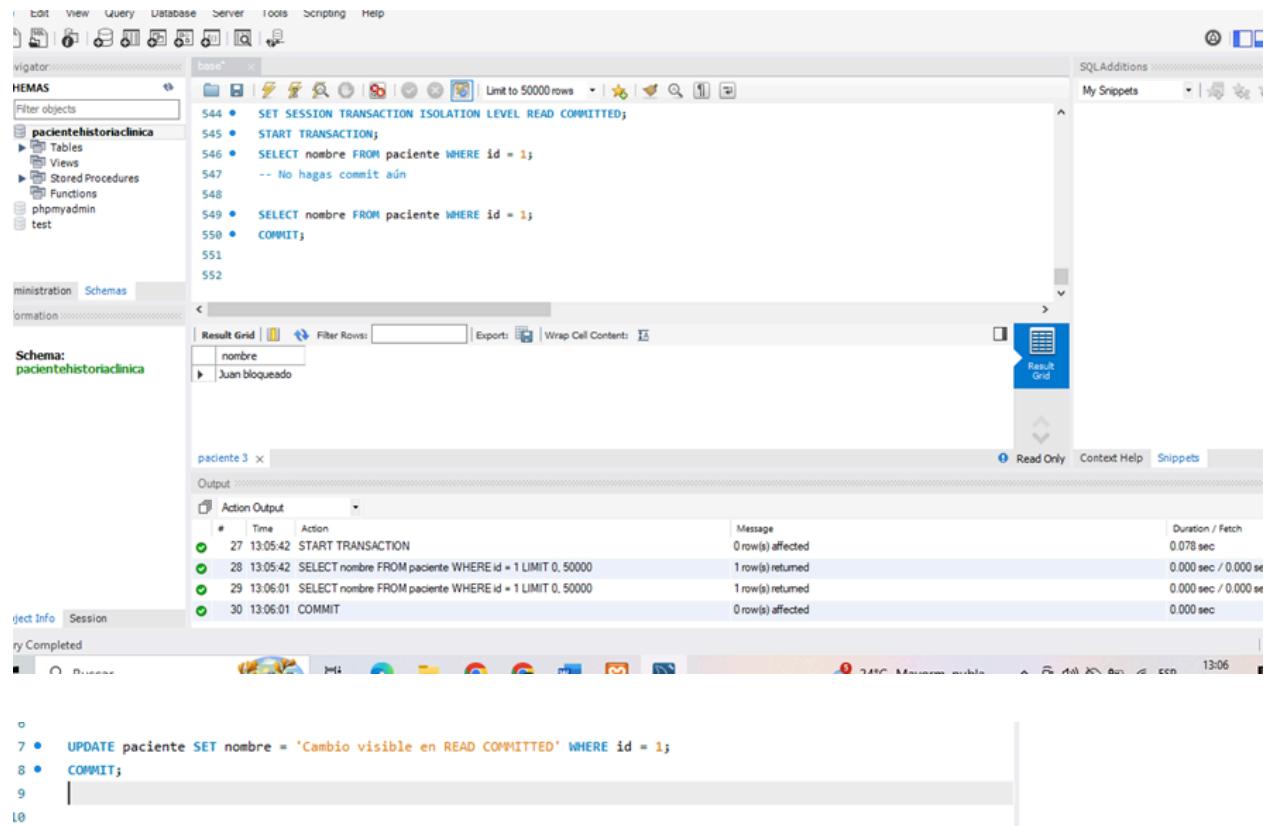
```
UPDATE paciente SET nombre = 'Cambio visible en READ COMMITTED' WHERE id = 1;
```

```
COMMIT;
```

Sesión 1 – vuelve a ejecutar:

```
SELECT nombre FROM paciente WHERE id = 1;
```

COMMIT;



The screenshot shows a MySQL Workbench interface. In the top-left pane, the database 'pacientehistori clinica' is selected. The 'Tables' node contains a single table named 'paciente'. In the main query editor window, the following SQL code is visible:

```
544 • SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
545 • START TRANSACTION;
546 • SELECT nombre FROM paciente WHERE id = 1;
547 -- No hagas commit aún
548
549 • SELECT nombre FROM paciente WHERE id = 1;
550 • COMMIT;
551
552
```

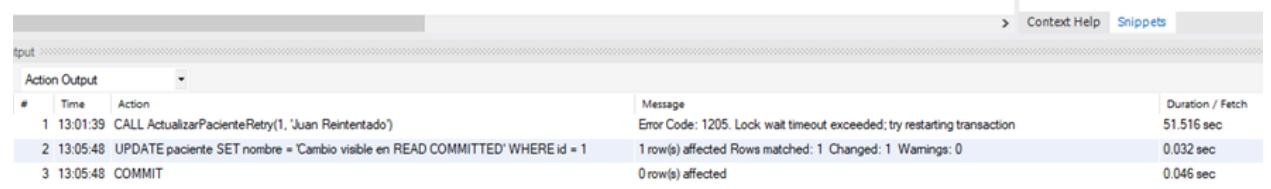
Below the code, the 'Result Grid' shows a single row with the name 'Juan bloqueado'. The 'Output' tab displays the transaction log:

#	Time	Action	Message	Duration / Fetch
27	13:05:42	START TRANSACTION	0 row(s) affected	0.078 sec
28	13:05:42	SELECT nombre FROM paciente WHERE id = 1 LIMIT 0, 50000	1 row(s) returned	0.000 sec / 0.000 sec
29	13:06:01	SELECT nombre FROM paciente WHERE id = 1 LIMIT 0, 50000	1 row(s) returned	0.000 sec / 0.000 sec
30	13:06:01	COMMIT	0 row(s) affected	0.000 sec

In the bottom-left pane, the session history shows:

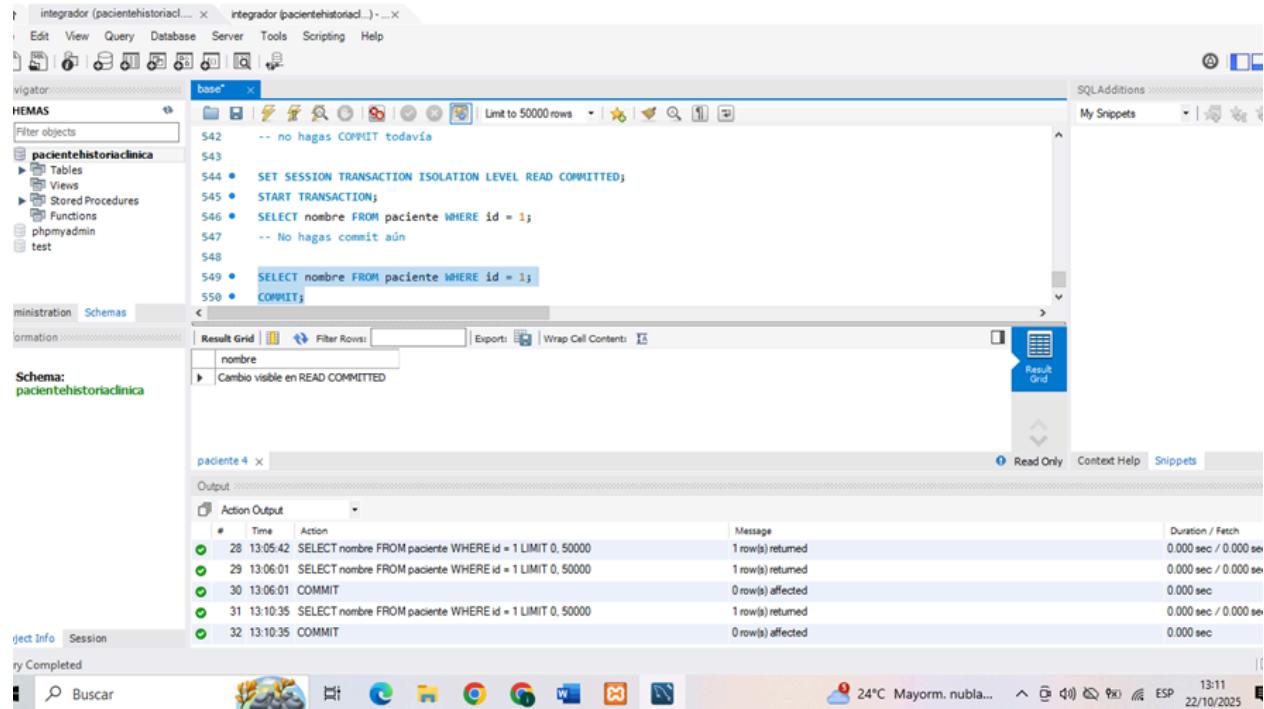
```
7 • UPDATE paciente SET nombre = 'Cambio visible en READ COMMITTED' WHERE id = 1;
8 • COMMIT;
9 |
```

The number '10' is also present at the bottom left.



The screenshot shows a continuation of the MySQL Workbench session. The 'Output' tab displays the following transaction log:

#	Time	Action	Message	Duration / Fetch
1	13:01:39	CALL ActualizarPacienteRetry(1, 'Juan Reintentado')	Error Code: 1205. Lock wait timeout exceeded; try restarting transaction	51.516 sec
2	13:05:48	UPDATE paciente SET nombre = 'Cambio visible en READ COMMITTED' WHERE id = 1	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.032 sec
3	13:05:48	COMMIT	0 row(s) affected	0.046 sec



The screenshot shows a MySQL Workbench interface. In the top query editor, a transaction is being run:

```
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
START TRANSACTION;
SELECT nombre FROM paciente WHERE id = 1;
-- No hagas commit aun
SELECT nombre FROM paciente WHERE id = 1;
COMMIT;
```

The result grid shows the output of the second SELECT statement:

nombre
Cambio visible en READ COMMITTED

In the bottom session details window, the transaction log is displayed:

#	Time	Action	Message	Duration / Fetch
28	13:05:42	SELECT nombre FROM paciente WHERE id = 1 LIMIT 0, 50000	1 row(s) returned	0.000 sec / 0.000 sec
29	13:06:01	SELECT nombre FROM paciente WHERE id = 1 LIMIT 0, 50000	1 row(s) returned	0.000 sec / 0.000 sec
30	13:06:01	COMMIT	0 row(s) affected	0.000 sec
31	13:10:35	SELECT nombre FROM paciente WHERE id = 1 LIMIT 0, 50000	1 row(s) returned	0.000 sec / 0.000 sec
32	13:10:35	COMMIT	0 row(s) affected	0.000 sec

probar REPEATABLE READ

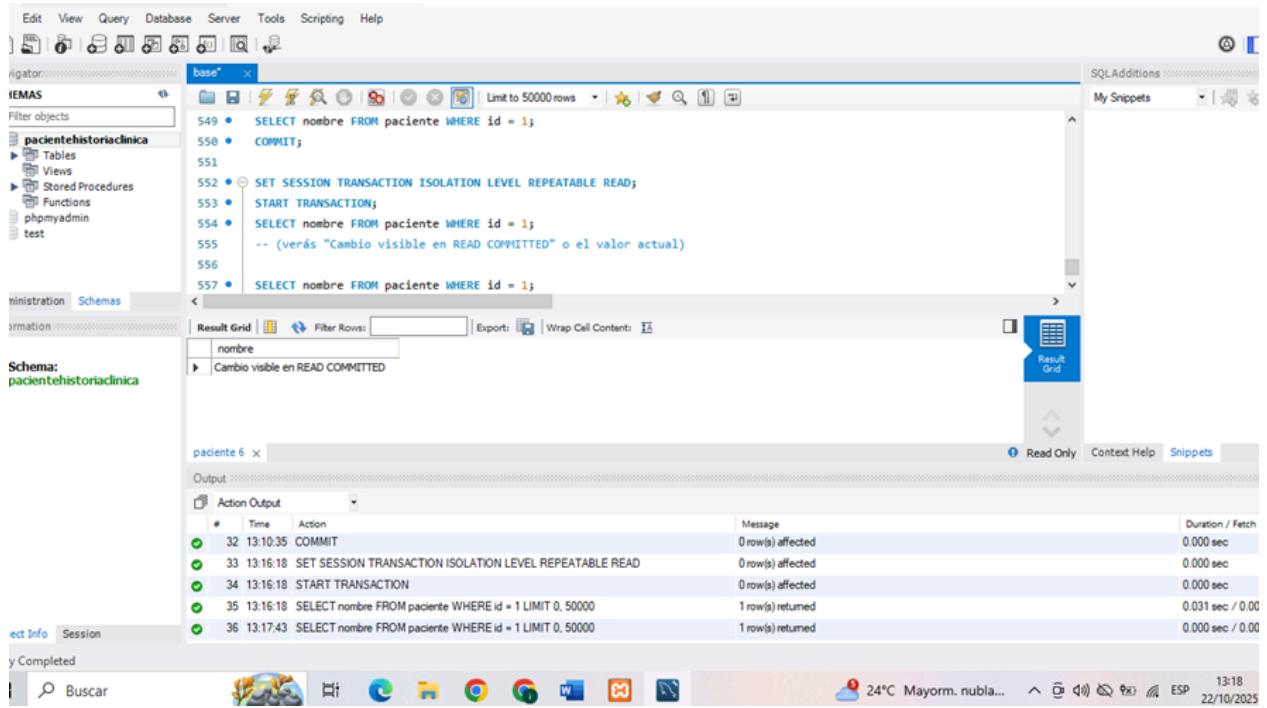
Sesión 1

SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;

START TRANSACTION;

SELECT nombre FROM paciente WHERE id = 1;

-- (se ve "Cambio visible en READ COMMITTED" o el valor actual)



The screenshot shows a MySQL Workbench interface. The top menu includes Edit, View, Query, Database, Server, Tools, Scripting, and Help. A toolbar with various icons is visible above the main area. The left sidebar shows the 'SCHEMAS' tree, with 'pacientehistori clinica' selected. The 'Tables' node under it is expanded, showing 'paciente' and 'historiaclinica'. The 'Result Grid' tab in the center displays a single row with the column 'nombre' containing 'Cambio visible en READ COMMITTED'. Below the result grid is an 'Action Output' log table:

#	Time	Action	Message	Duration / Fetch
32	13:10:35	COMMIT	0 row(s) affected	0.000 sec
33	13:16:18	SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ	0 row(s) affected	0.000 sec
34	13:16:18	START TRANSACTION	0 row(s) affected	0.000 sec
35	13:16:18	SELECT nombre FROM paciente WHERE id = 1 LIMIT 0, 50000	1 row(s) returned	0.031 sec / 0.00
36	13:17:43	SELECT nombre FROM paciente WHERE id = 1 LIMIT 0, 50000	1 row(s) returned	0.000 sec / 0.00

Sesión 2

UPDATE paciente SET nombre = 'Cambio invisible en REPEATABLE READ' WHERE id = 1;

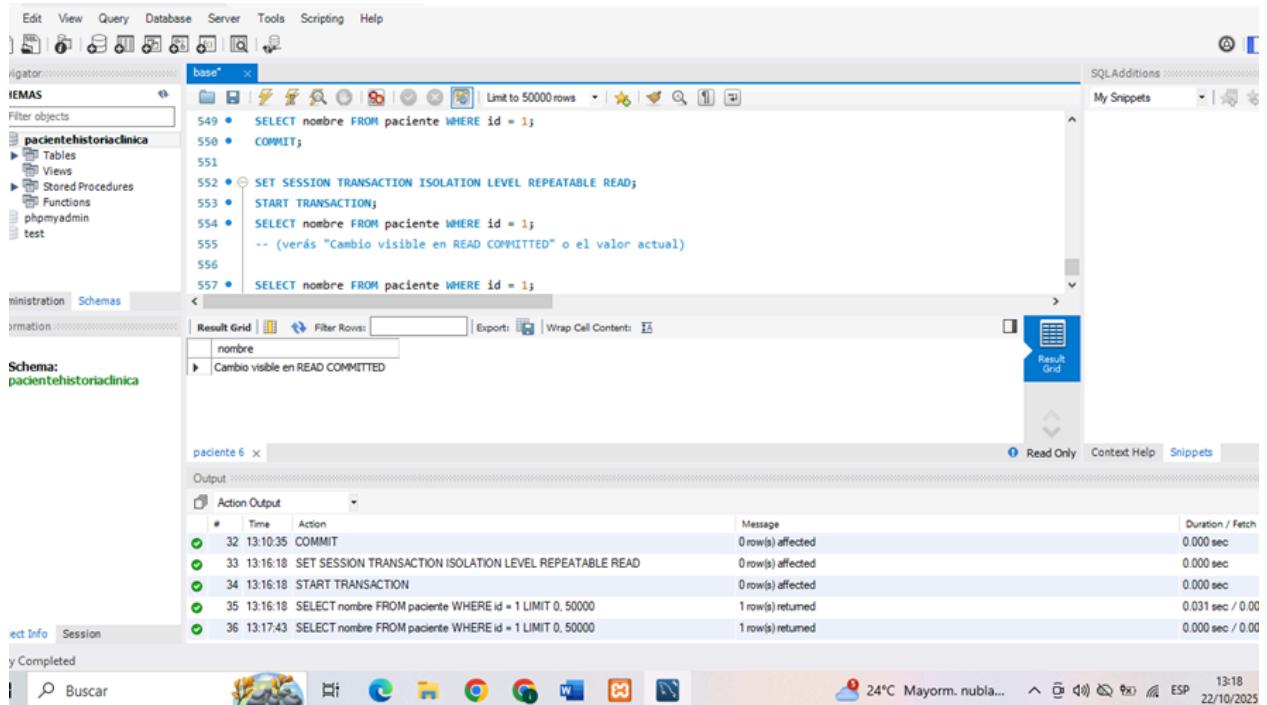
COMMIT;

Sesión 1 (sin cerrar todavía)

SELECT nombre FROM paciente WHERE id = 1;

-- verás que el valor NO cambió

COMMIT;



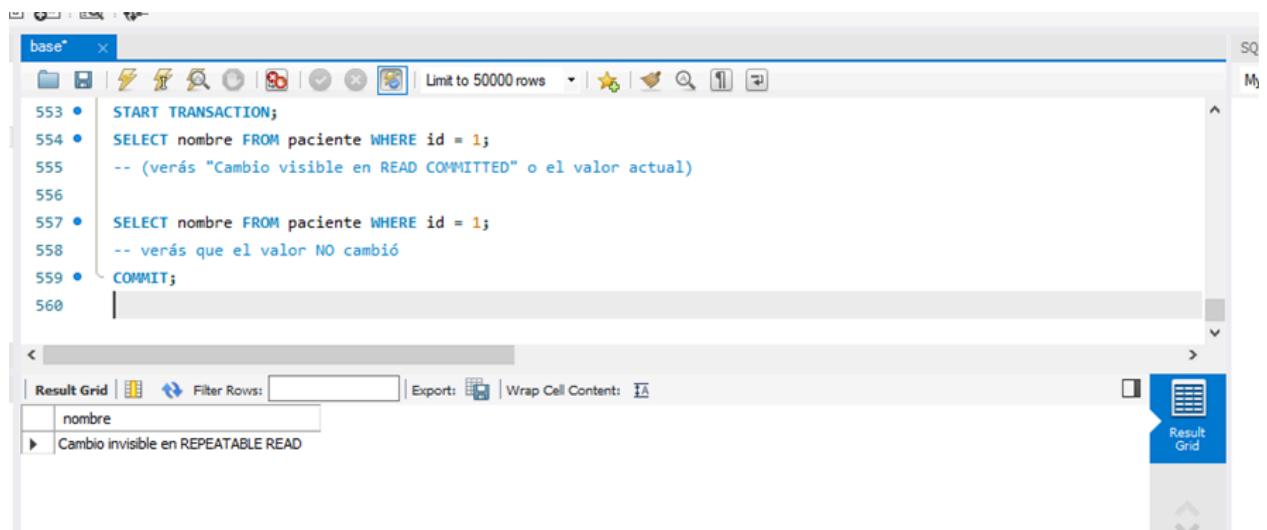
Session 1 (Top):

```
base* | Edit View Query Database Server Tools Scripting Help
File | Favorites | Tables | Views | Stored Procedures | Functions | phpmyadmin | test
Schemas | Administration Schemas
Schema: pacientehistoriclinica
Query Editor: base* | Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Limit to 50000 rows | ...
549 • SELECT nombre FROM paciente WHERE id = 1;
550 • COMMIT;
551
552 • SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
553 • START TRANSACTION;
554 • SELECT nombre FROM paciente WHERE id = 1;
555 -- (verás "Cambio visible en READ COMMITTED" o el valor actual)
556
557 • SELECT nombre FROM paciente WHERE id = 1;
```

Session 2 (Bottom):

```
paciente 6 | Edit Info | Session
Output:
Action Output | # Time Action Message Duration / Fetch
32 13:10:35 COMMIT 0 row(s) affected 0.000 sec
33 13:16:18 SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ 0 row(s) affected 0.000 sec
34 13:16:18 START TRANSACTION 0 row(s) affected 0.000 sec
35 13:16:18 SELECT nombre FROM paciente WHERE id = 1 LIMIT 0, 50000 1 row(s) returned 0.031 sec / 0.00
36 13:17:43 SELECT nombre FROM paciente WHERE id = 1 LIMIT 0, 50000 1 row(s) returned 0.000 sec / 0.00
```

En este caso, la sesión 1 no verá el cambio hecho por la sesión 2 hasta que haga su propio COMMIT.



Session 1 (Top):

```
base* | Edit View Query Database Server Tools Scripting Help
File | Favorites | Tables | Views | Stored Procedures | Functions | phpmyadmin | test
Schemas | Administration Schemas
Schema: pacientehistoriclinica
Query Editor: base* | Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Limit to 50000 rows | ...
553 • START TRANSACTION;
554 • SELECT nombre FROM paciente WHERE id = 1;
555 -- (verás "Cambio visible en READ COMMITTED" o el valor actual)
556
557 • SELECT nombre FROM paciente WHERE id = 1;
558 -- verás que el valor NO cambió
559 • COMMIT;
```

Session 2 (Bottom):

```
paciente 6 | Edit Info | Session
Output:
Action Output | # Time Action Message Duration / Fetch
553 13:16:18 START TRANSACTION 0 row(s) affected 0.000 sec
554 13:16:18 SELECT nombre FROM paciente WHERE id = 1;
555 -- (verás "Cambio visible en READ COMMITTED" o el valor actual)
556
557 13:17:43 SELECT nombre FROM paciente WHERE id = 1;
558 -- verás que el valor NO cambió
559 13:17:43 COMMIT; 0 row(s) affected 0.000 sec
```

Esto demuestra el comportamiento repeatable read (lectura consistente).

INFORME FINAL

Durante las pruebas de concurrencia en la base de datos *pacienteHistoriaClinica* se reprodujo un deadlock entre dos sesiones que intentaban actualizar registros cruzados, generando el error 1213. Posteriormente, se implementó un procedimiento almacenado con manejo de reintentos automáticos ante deadlocks, el cual permitió completar la transacción tras uno o dos intentos. En la comparación de niveles de aislamiento, el modo READ COMMITTED mostró cambios visibles entre transacciones concurrentes, mientras que REPEATABLE READ mantuvo una vista consistente hasta el commit. Estos resultados confirman la importancia del aislamiento y del manejo de errores en entornos con múltiples accesos simultáneos.

Anexo a interacción con IA: [W Interaccion con IA consigna 5 .docx](#)