EduTech Plus — Base de Datos Relacional

Actividad Integral – Sustentación Escrita

1. Introducción

En esta actividad se desarrolló una base de datos relacional para la empresa ficticia EduTech Plus, cuyo objetivo es gestionar información académica y administrativa como estudiantes, docentes, cursos, matrículas, calificaciones, pagos y certificaciones.

El desarrollo del proyecto se realizó paso a paso, iniciando desde el modelo entidad–relación (MER) hasta la implementación completa en MySQL, incluyendo procedimientos almacenados, triggers, vistas y consultas SQL avanzadas.

2. Modelo Entidad–Relación (MER)

El primer paso fue diseñar el MER, el cual fue pensado desde el inicio aplicando normalización hasta la Tercera Forma Normal (3FN), con el objetivo de evitar redundancia, mejorar la integridad de los datos y facilitar las consultas.

Relaciones Muchos a Muchos (N:M)

Durante el modelado identifiqué relaciones muchos a muchos, las cuales no pueden implementarse directamente en una base de datos relacional, por lo que fue necesario crear tablas intermedias:

Estudiantes ↔ Cursos

Un estudiante puede matricularse en muchos cursos.

Un curso puede tener muchos estudiantes.

Se normalizó mediante la tabla intermedia registrations (matriculas).

Cursos ↔ Evaluaciones (de forma indirecta a través de calificaciones)

Un curso puede tener varias evaluaciones.

Cada evaluación puede generar múltiples calificaciones.

Se manejó mediante las tablas evaluations y grades.

Consideré que usar tablas intermedias era lo más apropiado porque:

Permite mantener la base de datos normalizada.

Facilita el control de información adicional (fechas, estados, periodos).

Optimiza consultas complejas como el historial académico.

3. Creación de Tablas y Tipos de Datos

A partir del MER, se crearon las tablas usando MySQL Workbench, seleccionando los tipos de datos y restricciones que consideré más adecuados según el contexto del sistema.

Algunas decisiones tomadas:

Uso de INT con AUTO_INCREMENT para claves primarias.

Uso de VARCHAR para nombres, correos y descripciones.

Uso de DATE para fechas académicas.

Restricciones como:

NOT NULL para campos obligatorios.

UNIQUE para correos electrónicos.

Estados controlados mediante valores definidos (Paid, Pending, etc.).

Todo esto se hizo buscando coherencia con el enunciado y una base sólida para el sistema académico.

4. Integridad Referencial

Además de definir las claves foráneas dentro de las tablas, se aplicó integridad referencial explícita mediante FOREIGN KEY y ALTER TABLE.

El propósito de esto fue:

Evitar registros huérfanos.

Garantizar que no se inserten datos inválidos.

Mantener la consistencia entre entidades relacionadas.

Ejemplo:

No se puede registrar una matrícula si el estudiante o el curso no existen.

No se puede generar un pago para un periodo académico inexistente.

Esto hace que la base de datos no dependa únicamente del backend para validaciones críticas.

5. Procedimientos Almacenados

Se implementaron procedimientos almacenados para encapsular la lógica de negocio principal, evitando que esta lógica quede solo en el backend.

Enfoque general

Todos los procedimientos reciben parámetros (IN).

Se usan validaciones con condicionales (IF, EXISTS).

Se manejan errores con SIGNAL.

Procedimiento más relevante: Certificación Académica

En el procedimiento sp_generate_academic_certification:

Usé GROUP BY para agrupar las notas por curso.

Usé AVG() para calcular el promedio.

Usé HAVING para filtrar solo los cursos aprobados (promedio ≥ 3.0).

Esto fue necesario porque:

HAVING permite filtrar resultados después de una agregación o agrupaciones.

Se requería validar que el estudiante tuviera cursos aprobados antes de generar la certificación.

Los demás procedimientos se enfocan más en:

Validar existencia de datos.

Evitar duplicados.

Cumplir reglas indicadas en el enunciado.

6. Triggers

Se implementaron triggers como una capa adicional de seguridad, asegurando que ciertas reglas se cumplan siempre, incluso si no se usan los procedimientos almacenados.

Triggers implementados según el enunciado:

Validación del rango de calificaciones

Trigger BEFORE INSERT que evita insertar notas fuera del rango 0–5.

Esto asegura integridad incluso si alguien hace un INSERT directo.

Auditoría de pagos

Trigger AFTER UPDATE que registra automáticamente cuando un pago cambia a estado Paid.

Permite trazabilidad y control de acciones críticas.

7. Vistas (Views)

Las vistas se utilizaron para consultas repetitivas y reportes administrativos, facilitando el acceso a información compleja sin repetir consultas largas. (Segun el enunciado de la actividad).

Vista más compleja: Historial Académico del Estudiante

Esta vista fue la más compleja porque:

Requiere múltiples JOIN entre estudiantes, matrículas, cursos, evaluaciones, calificaciones y periodos.

Representa información que, según la lógica del sistema, siempre será consultada.

Es apropiado usar una vista porque:

Simplifica consultas futuras.

Mejora la legibilidad.

Facilita la generación de reportes.

8. Consultas SQL Avanzadas

Se desarrollaron las 10 consultas solicitadas en el enunciado, aplicando:

JOIN múltiples

Subconsultas

Funciones de agregación (AVG, SUM, COUNT)

HAVING

CASE WHEN

Uso de CTE en la consulta más adecuada

Uso de CASE WHEN

Utilicé CASE WHEN para clasificar estudiantes por rendimiento académico (Alto, Medio, Bajo), ya que permite:

Transformar valores numéricos en categorías.

Generar reportes más claros.

Uso de CTE

El CTE se utilizó en la consulta de ingresos por periodo porque:

El mismo cálculo se necesitaba más de una vez.

Permitía mayor claridad y orden en la consulta.

No era necesario usar CTE en todas las consultas, solo donde aportaba valor.

9. Dificultades

Una de las partes que más se me dificultó durante el desarrollo del proyecto fue la correcta aplicación de los JOINs, especialmente cuando las consultas involucraban muchas tablas relacionadas entre sí.

Aunque intente escribir todas las queries por mi cuenta, no todas funcionaron a la primera. En varios casos las consultas devolvían resultados incorrectos o simplemente no mostraban datos, lo que me obligó a revisar con más cuidado las relaciones entre tablas, las claves foráneas y el orden de los JOINs.

Esta dificultad se presentó principalmente en:

El procedimiento almacenado de generación de certificación académica, donde fue necesario combinar varias tablas y usar GROUP BY y HAVING.

Las consultas SQL avanzadas, donde algunas requerían múltiples JOINs y lógica adicional para cumplir con el enunciado.

En estos casos tuve la necesidad de apoyarme en la inteligencia artificial para identificar cuál era el error, entender por qué la consulta no funcionaba como esperaba y corregirla.

También debo aclarar que no todo fue perfecto en el resto del proyecto. Hubo partes donde tuve que leer bastante el enunciado, buscar cual funcion de agregacion era la mas adecuada para la consulta y volver a intentar varias veces, pero sin duda donde más regular me fue fue:

El uso correcto de los JOINs

La aplicación adecuada de CTE, ya que entender cuándo realmente valía la pena usarlos fue un reto adicional

10. Conclusión

Este proyecto me permitió entender que una base de datos profesional no se limita solo a crear tablas, sino que incluye:

Lógica de negocio en la base de datos.

Seguridad y trazabilidad.

Consultas claras y reutilizables.

Más allá de cumplir el enunciado, el objetivo fue entender el porqué de cada decisión y cómo se aplica en un contexto real.