DISEÑO DE

ARQUITECTURAS

DE BASE DE DATOS

TIPOS DE ARQUITECTURA DE BASE DE DATOS

Ejercicio Propuesto:

Comparar ventajas y desventajas de cliente-servidor frente a nube para un e-commerce.

Conclusión práctica para e-commerce

Para una tienda con demanda variable y necesidad de crecer rápido, la nube (p. ej., PostgreSQL/MySQL administrado) suele ganar: escalabilidad elástica, alta disponibilidad y menor carga operativa.

Excepciones donde on-prem puede ser útil: requisitos regulatorios estrictos de datos en sitio, latencias ultra-bajas en un campus, o costos prepagados ya hundidos con equipo y personal disponible.

Híbrido viable: app en nube + réplica local para analytics, o al revés con DR en nube.

Endulzar con: caché (Redis), CDN, réplicas de lectura, multi-AZ, backups automáticos y métricas (APM).

Tabla comparativa

Criterio	Cliente-Servidor (on-premises)	Nube (DBaaS / managed)
Costo inicial	Alto (hardware, licencias, data center).	Bajo-medio (pago por uso).
Costo operativo	Fijo + personal especializado.	Variable/optimizable; menos carga operativa.
Escalabilidad	Lenta; requiere comprar/instalar equipos.	Elástica (auto-scaling, read replicas). Ideal para picos (Black Friday).
Disponibilidad	Depende de redundancia propia; costosa.	Multi-AZ/Región disponible; SLA elevados.
RTO/RPO	Mayores si no se invierte en DR.	Bajos con backups automáticos y restauraciones point-in-time.
Latencia	Baja para usuarios locales si el DC está cerca.	Baja si eliges región cercana + CDN; globalmente superior.
Seguridad & Cumplimiento	Control total; mayor responsabilidad (físico, redes, parches).	Controles integrados (cifrado, IAM, auditoría); debes configurar bien y cumplir normas locales.
Time-to-market	Lento (aprovisionamiento).	Rápido (minutos).
Mantenimiento	Tú te encargas de todo.	Proveedor gestiona parches, upgrades, failover.
Riesgos	Obsolescencia, capacidad ociosa.	Dependencia del proveedor (lock-in), costos si no se monitorea.

CRITERIOS PARA SELECCIONAR UNA ARQUITECTURA SEGUN EL CONTEXTO DE LA APLICACION

Ejercicio Propuesto:

Proponer criterios de selección para una startup de software educativo.

Solución

1.Modelo de datos y transacciones

- ¿Necesitas ACID (matrículas, pagos, progreso, calificaciones)? → Relacional (PostgreSQL/MySQL) administrado.
- Contenido/archivos (videos, PDFs) → Almacenamiento de objetos.
- Eventos/telemetría a gran escala → NoSQL/streaming (Kafka/PubSub) + warehouse.

2. Patrones de acceso y picos

- Uso por horarios (mañana/tarde) y temporadas (exámenes).
- Requiere auto-escalado y réplicas de lectura; latencia < 150 ms global.

3. Escalabilidad y crecimiento

- Meta de usuarios/DAU a 12–24 meses.
- ¿Sharding/multi-tenant? Decide entre multi-tenant por esquema (más eficiente) vs por BD (más aislamiento).

4. Disponibilidad y continuidad

- SLA objetivo (ej.: 99.9%+).
- RTO/RPO (p. ej., RTO ≤ 15 min, RPO ≤ 5 min).
- Multi-AZ/región y restauraciones point-in-time.

5. Seguridad y cumplimiento

- Datos de menores y docentes: cifrado en reposo/en tránsito, control de acceso por rol (RBAC), logs de auditoría, residencia de datos (elige región cercana a tus usuarios y a la normativa local).

6. Costos
- Presupuesto mensual; preferir pago por uso y capacidad elástica.
- Evitar sobre-provisionamiento; monitoreo continuo del cost-per-student.

7. Equipo y operación
- ¿Tienen DBA/SRE? Si no, servicios administrados y/o serverless para reducir operación.

8.Analítica y reporting
- Necesidad de dashboards (retención, engagement, aprendizaje).
- Pipeline ELT a data warehouse (BigQuery/Redshift/Snowflake) desacoplado del OLTP.

9.Integraciones
- LMS/LTI, pagos, identidad (SSO/OAuth), mensajería.
- Preferir arquitecturas que expongan APIs y soporten CDC (Change Data Capture) si se requiere sincronizar.

10. Portabilidad / lock-in
- Usar SQL estándar (PostgreSQL) y herramientas portables; evitar extensiones propietarias críticas al inicio.

Baseline recomendado para una EdTech early-stage:

- OLTP principal: PostgreSQL administrado (multi-AZ, backups PITR).
- Cache: Redis administrado para sesiones y lecturas calientes.
- Contenido: almacenamiento de objetos (con CDN).
- Eventos: cola/pub-sub para tracking en tiempo real.
- Analítica: ETL/ELT nocturno hacia warehouse.
- Observabilidad: métricas, trazas y alertas de costo/latencia.

MODELADO CONCEPTUAL Y LÓGICO DE DATOS

Ejercicio Propuesto:

Transformar un modelo conceptual de un hospital a modelo lógico

1. **Modelo conceptual (E-R del hospital, simplificado):**

Entidades principales:

- Paciente (ID_Paciente, Nombre, Apellido, Fecha_Nacimiento, Dirección).
- Doctor (ID_Doctor, Nombre, Especialidad).
- Cita (ID_Cita, Fecha, Hora).
- Historia_Clínica (ID_Historia, Diagnóstico, Tratamiento).

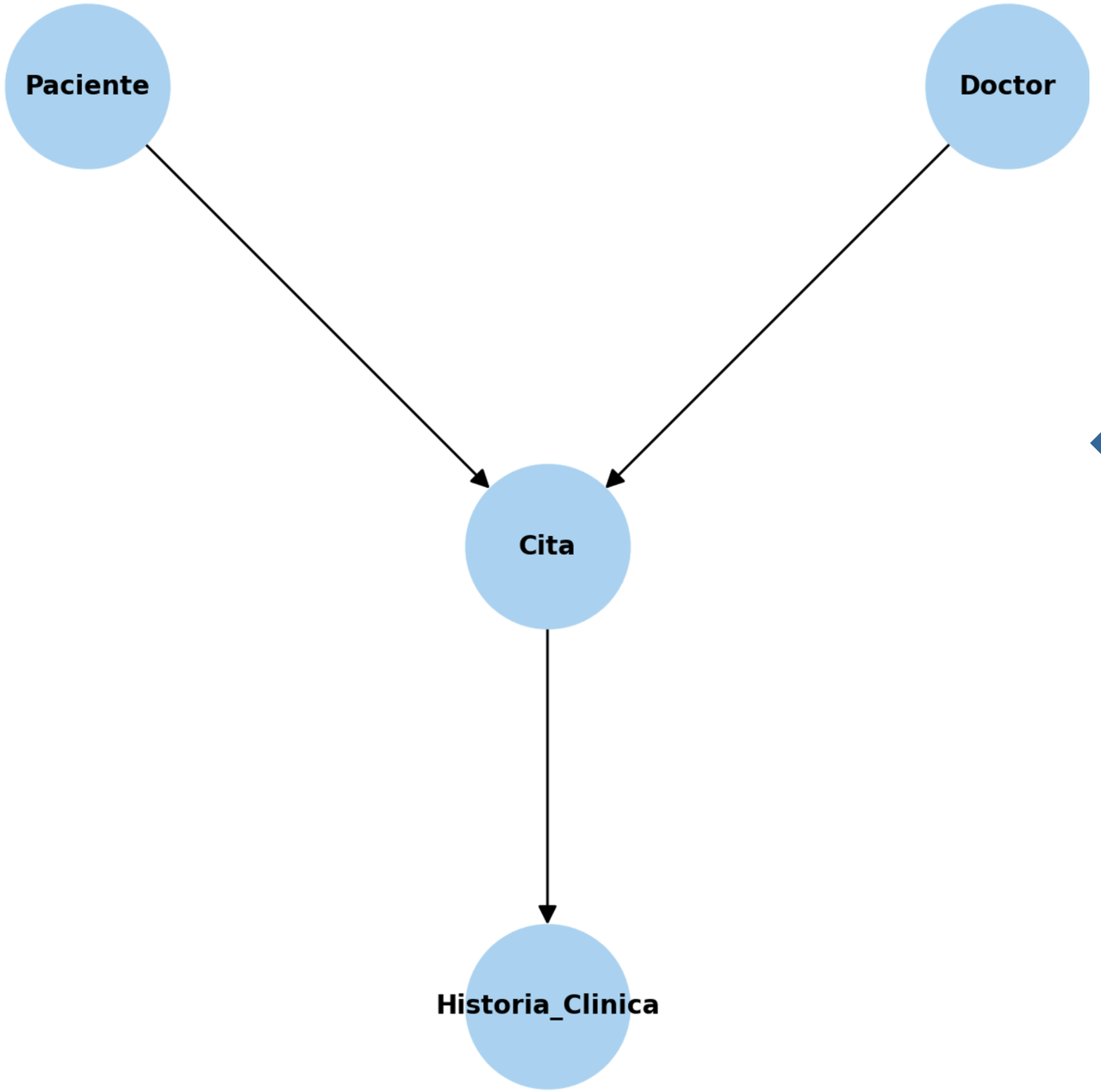
Relaciones:

- Un Paciente puede tener muchas Citas.
- Un Doctor atiende muchas Citas.
- Una Cita genera una Historia_Clínica.

2. Transformación a modelo lógico (Tablas en BD relacional):

- Paciente(ID_Paciente PK, Nombre, Apellido, Fecha_Nacimiento, Dirección).
- Doctor(ID_Doctor PK, Nombre, Especialidad).
- Cita(ID_Cita PK, Fecha, Hora, ID_Paciente FK, ID_Doctor FK).
- Historia_Clínica(ID_Historia PK, Diagnóstico, Tratamiento, ID_Cita FK).

Modelo E-R Hospital (simplificado)



NORMALIZACIÓN Y OPTIMIZACIÓN DEL DISEÑO

Ejercicio Propuesto:

Crear índices para mejorar consultas en una base de datos ventas

1

Ventas (ID_Venta, Fecha, Cliente, DirecciónCliente, Producto, Precio, Cantidad, Vendedor)

2

Problema: hay redundancia porque cada venta repite datos de Cliente y Producto.

3

Normalización y mejor diseño:

- Cliente(ID_Cliente PK, Nombre, Dirección).
- Producto(ID_Producto PK, Nombre, Precio).
- Venta(ID_Venta PK, Fecha, ID_Cliente FK, ID_Vendedor FK).
- Detalle_Venta(ID_Venta FK, ID_Producto FK, Cantidad).
- Vendedor(ID_Vendedor PK, Nombre).

Mejora en consultas:

- Consultar ventas de un cliente es directo (JOIN entre Cliente y Venta).
- Calcular ingresos por producto se hace en Detalle_Venta.
- Evita redundancia de direcciones y precios en cada fila.

Modelo Normalizado de Ventas