Universidad del Valle de Guatemala
Base de datos 1
Sección 20
Milton Polanco 23471

Reflexión Individual – Milton Polanco

1. **Criterios para decidir entidades y relaciones**

Partí de los procesos clave del sistema real: definir ubicaciones (ciudades, instalaciones), gestionar canchas con sus horarios y servicios, y permitir que usuarios reserven y valoren esas canchas. Del análisis de requerimientos derivé estas entidades:

- **ciudad & instalación**: representé primero la jerarquía geográfica con **ciudad** y **instalación**, para luego asociar canchas a una instalación específica.
- **deporte** y **cancha**: separé el tipo de cancha en su propia tabla **deporte** (e.g. fútbol, tenis) para mantener consistencia y poder agregar más deportes sin modificar la estructura de **cancha**.
- **horario**, **cancha_horario**: modelé los rangos de tiempo disponibles en **horario**, y relacioné horarios y canchas mediante la tabla intermedia **cancha_horario**, evitando listas de horas en un solo campo y asegurando un diseño 1NF.
- **usuario** y **usuario_telefono**: separé teléfonos en una tabla hija para permitir múltiples números por usuario.
- **reserva**, **reserva_servicio**, **pago** y **valoracion**: abstraje la lógica de reserva en varias tablas: **reserva** guarda el intervalo y usuario–cancha; **reserva_servicio** asocia servicios adicionales; **pago** enlaza un único pago por reserva; **valoracion** permite puntuaciones de 1 a 5 con texto.

Universidad del Valle de Guatemala
Base de datos 1
Sección 20
Milton Polanco 23471

2. Adecuación de claves primarias y foráneas

- Cada tabla usa un **PK serial** (id GENERATED ALWAYS AS IDENTITY), lo que simplifica referencias y garantiza unicidad.
- Las **FK** están declaradas con comportamientos coherentes:
    - En **instalacion**, id_ciudad → ciudad(id) ON DELETE RESTRICT para no eliminar ciudades con instalaciones.
    - En **cancha**, id_instalacion → instalacion(id) y id_deporte → deporte(id) ON DELETE RESTRICT, obligando a que cada cancha pertenezca a una instalación y deporte válidos.
    - En **reserva**, id_usuario → usuario(id) y id_cancha → cancha(id) ON DELETE CASCADE, de modo que al borrar un usuario o cancha se limpien reservas asociadas.
    - En **reserva_servicio**, id_reserva → reserva(id) ON DELETE CASCADE y id_servicio → servicio_adicional(id) ON DELETE RESTRICT, evitando servicios huérfanos.
    - En **pago**, la FK id_reserva UNIQUE REFERENCES reserva(id) garantiza un pago por reserva.

3. Normalización aplicada

- **1FN**: todos los atributos son atómicos. No hay columnas con listas (por ejemplo, horarios) y cada entidad tiene su propia tabla.
- **2FN**: no existen dependencias parciales porque cada tabla usa un PK simple (id), y cada atributo depende de toda la clave.
- **3FN**: no almaceno datos derivados: la duración de la reserva se calcula en tiempo real en un trigger (trg_calcular_duracion), y el total de pago se calcula dinámicamente con fn_calcular_pago_total.

Beneficios:

- Evito duplicación de datos (la tarifa de cancha se define una sola vez).
- Facilito el mantenimiento y la extensibilidad (añadir un nuevo servicio o deporte no requiere alterar tablas existentes).

Limitaciones:

- Calcular duración o totales con triggers en cada inserción/actualización puede penalizar el rendimiento si hay miles de operaciones por segundo. En ese caso evaluaría denormalizar parcialmente (por ejemplo, almacenar la duración real o el total en la tabla) o migrar el cálculo a lotes asíncronos.

4. Restricciones y reglas de negocio implementadas

- **NOT NULL**: en campos clave como ciudad.nombre, cancha.precio_hora, reserva.fecha y horas, para garantizar datos completos.
- **UNIQUE**: en ciudad.nombre, en la combinación (nombre, id_ciudad) de **instalacion**, en (id_usuario, id_cancha) de **valoracion** y (id_reserva, id_servicio) en **reserva_servicio**, evitando duplicados lógicos.
- **CHECK**: en cancha.capacidad >= 1, servicio_adicional.precio >= 0, reserva.hora_fin > reserva.hora_inicio, y en **valoracion** puntuacion BETWEEN 1 AND 5.
- **DEFAULT**: en valoracion.fecha_valoracion y en pago.fecha_pago con NOW(), para registrar automáticamente la fecha.
- **TRIGGERS**:
    - **trg_calcular_duracion**: antes de INSERT/UPDATE en **reserva**, ejecuta fn_calcular_duracion() para rellenar duracion.
    - **fn_evitar_solapamiento** (en **triggers.sql**, parte omitida arriba): bloquea reservas cuya franja tsrange(fecha + hora_inicio, fecha + hora_fin) se solape con otra existente para la misma cancha.
    - **trg_calcular_pago**: antes de INSERT/UPDATE en **pago**, calcula el total sumando reserva.duracion * cancha.precio_hora + SUM(servicios).

Esto traslada validaciones críticas al motor de la base, reduciendo la posibilidad de inconsistencias.

Universidad del Valle de Guatemala
Base de datos 1
Sección 20
Milton Polanco 23471

5. Ventajas y desventajas al hacer consultas complejas

- **Ventajas**
    - El diseño modular permite combinar fácilmente tablas con múltiples filtros (por ciudad, por rango de fechas, por estado de pago).
    - Las FK y CHECK aseguran resultados coherentes, y las vistas o queries parametrizadas pueden reutilizarse en la capa de aplicación (por ejemplo, en AdminApp).
- **Desventajas**
    - Los cálculos en triggers, unidos a JOINs en tablas como **reserva**, **cancha** y **servicio_adicional**, pueden ralentizar consultas de reportes masivos (e.g. “ingresos totales por mes” en crear_reporte_comparativo_mensual).
    - Consultas para detectar huecos libres (horarios disponibles) requieren funciones de ventana o subconsultas complejas sobre tsrange, afectando legibilidad y potencialmente el rendimiento.

Para análisis intensivo (big data), sería preferible usar vistas materializadas o un almacén de datos OLAP.

6. Cambios para escalar a producción

- **Particionamiento** de **reserva** (por año o por ciudad) para mejorar tiempos de consulta y mantenimiento de índices.
- **Índices compuestos** en columnas frecuentemente consultadas:
  - (id_cancha, fecha, hora_inicio) en **reserva**,
  - (id_servicio, fecha) en **reserva_servicio**,
  - (id_usuario, fecha) para reportes por usuario.
- **Caching** o **vistas materializadas** para reportes de uso intensivo (ingresos diarios, ocupación por hora).
- **Desacoplar** triggers muy costosos y trasladar su lógica a procesos asíncronos (e.g., cálculo de totales en una cola de mensajes).
- **Escalabilidad horizontal**: en entornos con múltiples sedes, shardear por id_ciudad o usar schemas separados por región.
- **Alta disponibilidad**: configurar réplicas de lectura, balanceadores y backups continuos, junto con un plan de recuperación ante desastres.

Con estas mejoras, el modelo mantendría integridad y rendimiento aún ante altos volúmenes de usuarios y datos.