Reflexión individual Roberto Nájera

1. Criterios para decidir entidades y relaciones
   - Identificación de "núcleos funcionales"

Se detectaron primero los dominios básicos del sistema: *Eventos*, *Usuarios*, *Sedes* y *Categorías*. A partir de ahí, se añadieron las entidades que participaban o aportaban a esos núcleos: *Actividades* dentro de cada evento; *Artistas* y su vinculación a actividades; *Patrocinadores* y su relación con eventos; *Recursos* y su asignación a eventos; y la inscripción y asistencia de usuarios a eventos y actividades.

   - Balance entre detalle y complejidad

Para no sobredimensionar el modelo, algunos aspectos del "mundo real" se abstrajeron o simplificaron:

      - El contacto de artista o sede es un simple VARCHAR en lugar de un sub-objeto con teléfono, email y direcciones separadas.
      - Se usan tipos ENUM para estados y clasificaciones (por ejemplo, estado_evento, clasificacion_patrocinador) que limitan el dominio de valores sin necesidad de tablas auxiliares.
   - Decisiones de modelado
      - Surrogate keys (SERIAL) para facilitar joins y evolución del esquema.
      - Unique constraints en campos de negocio (p. ej. uk_patrocinio_patrocinador_evento) para evitar duplicados lógicos.
      - Relaciones con ON DELETE CASCADE en actividades y asignaciones de recursos para simplificar limpieza de datos huérfanos.

2. Adecuación de claves primarias y foráneas
   - Claves primarias (PK)
      - Se usaron PKs enteras con SERIAL en todas las tablas: homogéneo y eficiente para índices y joins.
      - Facilitan la creación de relaciones y permiten cambiar el valor "natural" de un registro sin romper dependencias.
   - Claves foráneas (FK)
      - Aseguran integridad referencial (p. ej. fk_actividad_evento con ON DELETE CASCADE).
      - Permiten consultas sencillas usando JOINs: por ejemplo, obtener todas las actividades de un evento o todos los patrocinios de un patrocinador.
      - En combinaciones únicas (como inscripción + actividad o patrocinador + evento) evitan duplicar relaciones lógicas.
   - Balance
      - Gracias a estas PK/FK, las consultas complejas mantienen coherencia y la base impide que queden huérfanos o inconsistentes.
      - A nivel de rendimiento, los índices automáticos en PK y FK suelen ser suficientes; si surgieran cuellos de botella podría añadirse índices compuestos.

3. Normalización y sus efectos

- Aplicación de 1FN, 2FN y 3FN
  - 1FN: todos los atributos son atómicos.
  - 2FN y 3FN: no hay atributos dependientes de sólo parte de la clave ni dependencias transitivas: cada tabla describe un solo concepto.
- Beneficios
  - Reducción drástica de redundancia (por ejemplo, datos de usuario o sede sólo aparecen en su tabla).
  - Consistencia y facilidad de mantenimiento (cambios de nombre, estado, etc., en un solo lugar).
- Limitaciones y trade-offs
  - JOINs múltiples: consultas muy normalizadas pueden requerir combinar varias tablas, lo cual resta rendimiento si no hay buenos índices.

4. Restricciones y reglas de negocio en la base de datos

Mecanismo	Ejemplo en el esquema	Justificación
CHECK	CHK (fecha_fin > fecha_inicio) en eventos	Evitar eventos con fechas inválidas
	CHECK (cupo_limite > 0) en actividades	Asegurar que cupos sean positivos
DEFAULT	estado evento DEFAULT 'Planificado'	Simplificar inserciones iniciales
NOT NULL	nombre VARCHAR NOT NULL en casi todas tablas	Campo fundamental para cada entidad
UNIQUE	uk_categoria_nombre, uk_artista_email, otros uk en nombres de tablas principales	Evitar duplicados lógicos
FOREIGN KEY	fk_evento_categoria, fk_inscripcion_usuario	Mantener integridad y dependencias
TRIGGERS	trg_actualizar_estado_evento	Actualizar estados según fechas
	trg_actualizar_disponibilidad_recursos	Control automático de stock de recursos
	trg_registrar_asistencia_evento	Marcar asistencia en evento
	trg_verificar_cupo_evento	Verificar la disponibilidad de un evento
	trg_restaurar_disponibilidad_recursos	Volver a disponer de recursos liberados

Todas estas reglas trasladan parte de la lógica de negocio a nivel de base, garantizando que las aplicaciones cliente no puedan violar integridad ni dejar datos en estado inconsistente.

5. Ventajas y desventajas al hacer consultas complejas

- Ventajas
  - Flexibilidad: el modelo claramente separa entidades, lo que facilita filtrar por categoría, sede, estado, patrocinador, etc.
  - Escalabilidad conceptal: nuevas entidades (p. ej. “comentarios” o “valoraciones globales”) se integran sin romper el diseño.
  - Mantenimiento: cambios de reglas (CHECK, triggers) no requieren alterar la lógica de la capa de aplicación.
- Desventajas
  - Coste de JOINs: consultas que crucen 6–8 tablas (evento→actividad→asistencia→usuario…) pueden volverse lentas si no hay índices adecuados.
  - Complejidad: el SQL necesario para agregaciones o subconsultas muy profundas es más difícil de escribir y optimizar.
  - Sobrecarga de triggers: en escenarios de muchos registros simultáneos, la ejecución de lógica en triggers puede convertirse en cuello de botella.

6. Cambios para escalar a producción

1. Índices adicionales
   - Crear índices compuestos en columnas frecuentemente filtradas (p. ej. (id_evento, estado), (hora_inicio, tipo_actividad)).
2. Particionamiento
   - Particionar tablas grandes (inscripciones, asistencia, recursos_evento) por rango de fechas o “id_evento” para mejorar I/O y limpieza de datos antiguos.
3. UUIDs y replicación
   - Pasar de SERIAL a UUID para entornos distribuidos; configurar réplicas de sólo lectura para consultas analíticas.
4. Caching y materialized views
   - Materializar agregaciones de uso intensivo (p. ej. total de participantes por evento) para reducir cálculos en caliente.
5. Monitorización y ajuste de triggers
   - Evaluar trasladar parte de la lógica de triggers muy costosos al nivel de aplicación o a jobs asíncronos.
6. Seguridad y auditoría
   - Añadir campos de auditoría (created_by, updated_by) y activar RLS en tablas sensibles.
7. Escalabilidad horizontal

- Considerar separar microservicios: uno para gestión de eventos, otro para facturación/pagos, otro para recursos, etc., cada uno con su propia base de datos optimizada.