Documentación de la Base de Datos

Proyecto: Hermandad – Gestión de pasos, tramos y participantes

---

1. Objetivo general del sistema

Esta base de datos tiene como finalidad **modelar y gestionar la estructura organizativa de una hermandad**, permitiendo:

- La gestión de hermanos (usuarios) y su sistema de seguridad mediante roles y permisos.
- La organización de pasos procesionales y su división en tramos.
- La asignación de hermanos a tramos con **roles funcionales excluyentes** (costalero, músico o nazareno).
- Un diseño optimizado para su **consumo mediante una API REST**, reduciendo la lógica necesaria en el backend.

El sistema ha sido diseñado siguiendo principios de:

- Normalización de datos
- Separación clara de responsabilidades
- Protección del dominio desde la base de datos
- Escalabilidad y mantenibilidad a largo plazo

---

2. Principios de diseño aplicados

2.1 Separación entre identidad y función

Un principio clave del diseño es distinguir entre **la identidad del hermano** y **la función que desempeña**.

- Un hermano **no es permanentemente** costalero, músico o nazareno.
- Un hermano **participa en un tramo concreto** desempeñando un rol funcional determinado.

Este enfoque evita incoherencias conceptuales y permite que un mismo hermano pueda desempeñar funciones distintas en tramos diferentes.

2.2 Exclusividad de rol por tramo

Se establece la siguiente regla fundamental del dominio:

> Un hermano **solo puede desempeñar un único rol funcional en un mismo tramo**.

Esta restricción se garantiza mediante:

- Restricciones de unicidad (`UNIQUE (id_hermano, id_tramo)`)
- Triggers de validación en tablas especializadas

Gracias a esto, la integridad del modelo no depende únicamente de la API.

2.3 Base de datos defensiva, API orquestadora

El reparto de responsabilidades es el siguiente:

- **Base de datos**
  - Garantiza la integridad estructural del dominio
  - Impide estados inconsistentes
- **API**
  - Controla los flujos de negocio
  - Decide cuándo y cómo se crean participaciones
  - Gestiona reglas funcionales y temporales

Este equilibrio evita sobrecargar la base de datos con lógica de negocio y mantiene la API limpia.

---

3. Estructura general del modelo de datos

3.1 Entidades principales

Entidad	Responsabilidad
`hermano`	Identidad del miembro
`rol`	Rol de seguridad
`permiso`	Permisos asociados a roles
`paso`	Paso procesional
`tramo`	División organizativa de un paso
`participacion`	Relación hermano–tramo–rol
`rol_funcional`	Rol operativo en un tramo
`banda`	Información musical

---

4. Modelo de participación (núcleo del sistema)

4.1 Tabla `participacion`

La tabla `participacion` es el **núcleo funcional del sistema**.

Representa el concepto:

> "Un hermano participa en un tramo con un rol funcional concreto".

Campos clave: - `id_hermano` - `id_tramo` - `id_rol_funcional`

Restricción crítica:

```
UNIQUE (id_hermano, id_tramo)
```

Esto garantiza que:
• Un hermano no puede duplicar su participación en un tramo. • No puede existir más de un rol funcional por tramo y hermano. "' ### 4.2 Especialización por rol funcional

Cada rol funcional posee información específica que se almacena en **tablas especializadas**, separadas de la tabla `participacion`:

- `participacion_costalero`
- `participacion_musico`
- `participacion_nazareno`

Este diseño responde a los siguientes criterios:

- Evitar columnas nulas en una única tabla genérica.
- Mantener una **normalización correcta** del modelo.
- Facilitar la **extensión futura** del sistema con nuevos roles funcionales.
- Garantizar coherencia entre el rol declarado y los datos almacenados.

Cada una de estas tablas contiene únicamente la información relevante para su rol, manteniendo una relación uno a uno con `participacion`.

---

5. Triggers: protección de la integridad del dominio

Los triggers se utilizan exclusivamente para **proteger reglas estructurales del dominio**, aquellas que:

- No pueden expresarse únicamente mediante claves foráneas.
- No deben depender de la lógica de la API.

Su objetivo no es implementar lógica de negocio, sino **evitar estados inválidos** en la base de datos.

5.1 Asignación automática de rol base

Motivación
Todo hermano debe poseer al menos el rol de seguridad `HERMANO_BASE`.

Implementación
Se define un trigger `AFTER INSERT ON hermano` que:

- Obtiene el identificador del rol `HERMANO_BASE`.
- Inserta automáticamente dicho rol en la tabla `hermano_rol`.

Beneficios

- El sistema garantiza que todo hermano tenga un rol mínimo.
- La API no necesita preocuparse de esta asignación inicial.
- Se evita que existan hermanos sin rol de seguridad.

5.2 Validación de coherencia entre rol funcional y tabla de detalle

Problema
Sin validaciones adicionales, sería posible insertar información específica de un rol en una participación cuyo rol funcional no coincide, produciendo datos inconsistentes.

Solución
Se implementan triggers `BEFORE INSERT` y `BEFORE UPDATE` en las siguientes tablas:

- `participacion_costalero`
- `participacion_musico`
- `participacion_nazareno`

Estos triggers:

- Verifican el rol funcional asociado a la participación.
- Impiden inserciones o modificaciones incoherentes.
- Lanzan errores claros mediante `SIGNAL SQLSTATE '45000'`.

Resultado
La base de datos queda protegida frente a inconsistencias incluso ante errores de la API.

---

6. Vistas: capa de lectura para la API

Las vistas se utilizan como una **capa de abstracción de lectura**, permitiendo que la API:

- Consuma datos ya estructurados.

- Evite realizar joins complejos.
- Mantenga independencia del esquema interno.

6.1 Vista base `vw_participaciones`

Centraliza toda la información relevante de una participación:

- Datos del hermano.
- Información del tramo.
- Paso asociado.
- Rol funcional desempeñado.

Esta vista sirve como base para la mayoría de endpoints de lectura y es la pieza central del consumo de datos desde la API.

6.2 Vistas especializadas por rol funcional

Se definen vistas específicas para cada rol funcional:

- `vw_costaleros`
- `vw_musicos`
- `vw_nazarenos`

Estas vistas:

- Filtran automáticamente por rol funcional.
- Añaden únicamente los campos específicos del rol.
- Permiten endpoints claros y semánticos.

Ejemplos de uso:

- `/pasos/{id}/costaleros`
- `/tramos/{id}/musicos`

6.3 Vista `vw_tramos`

Esta vista facilita la obtención de los tramos junto con la información del paso al que pertenecen.

Su objetivo es permitir listados simples de tramos sin necesidad de lógica adicional en la API.

6.4 Vista `vw_pasos_resumen`

Proporciona información agregada por cada paso:

- Número total de participantes.
- Desglose por rol funcional.

Es especialmente útil para:

- Dashboards.
- Resúmenes generales.
- Estadísticas rápidas.

6.5 Vistas de seguridad

6.5.1 `vw_hermanos_roles` Muestra cada hermano junto con sus roles de seguridad asociados.

Uso principal:

- Paneles de administración.
- Gestión de usuarios desde la API.

6.5.2 `vw_hermano_permisos` Devuelve el conjunto de permisos efectivos de cada hermano, teniendo en cuenta todos sus roles de seguridad.

Es clave para:

- Autenticación.
- Autorización.
- Control de acceso en la API.

---

7. Responsabilidades excluidas de la base de datos

Por decisión de diseño, la base de datos **no asume** las siguientes responsabilidades:

- Orquestación de flujos de negocio.
- Inserción automática de participaciones o detalles.
- Modificación de roles funcionales.
- Gestión de fechas, eventos o reglas temporales.

Estas tareas corresponden exclusivamente a la capa de API.

---

8. Beneficios del diseño final

- Dominio protegido desde la base de datos.
- API limpia y desacoplada.
- Modelo flexible y extensible.
- Consultas optimizadas mediante vistas.
- Documentación viva integrada en el propio esquema.

---

9. Conclusión

Este diseño de base de datos va más allá del almacenamiento de información y **modela correctamente el dominio real de una hermandad**, proporcionando una base sólida para un desarrollo moderno basado en APIs REST.

El equilibrio entre:

- restricciones estructurales,
- triggers de validación,
- vistas de lectura,
- y lógica de negocio en la API,

garantiza un sistema robusto, coherente y mantenible a largo plazo.