Modelo de datos

Este documento describe únicamente el modelo de datos del sistema: entidades, relaciones, restricciones, reglas de integridad y cómo soporta el flujo funcional de negocio.

---

1) Objetivo del modelo

El esquema de base de datos está diseñado para cubrir estos escenarios:

- Gestión de usuarios por unidad organizacional.
- Gestión de tareas con estado, prioridad y asignación.
- Flujo de solicitudes de cambio (`CREATE`, `UPDATE`, `COMPLETE`, `DELETE`).
- Aprobación/rechazo por supervisor.
- Trazabilidad y auditoría de eventos.

El modelo separa claramente:

- Dato operativo: `tasks`.
- Intención de cambio: `task_change_requests`.
- Evidencia de acciones: `task_events`.

---

2) Catálogos base

2.1 `organizational_units`

Propósito:

- Define el catálogo de unidades de negocio (ej. Finanzas, RRHH, BI).

Campos clave:

- `id` (PK)
- `code` (único, en mayúsculas)
- `name` (único)
- `is_active`
- `created_at`, `updated_at`

Justificación:

- Todas las reglas de visibilidad y aprobación están condicionadas por unidad.

2.2 `roles`

Propósito:

- Define roles globales del sistema (`STANDARD`, `SUPERVISOR`).

Campos clave:

- `id` (PK)
- `code` (único, mayúsculas)
- `name`
- `is_active`
- `created_at`, `updated_at`

2.3 `permissions`

Propósito:

- Catálogo de permisos funcionales (`TASK_VIEW_ALL`, `TASK_APPROVE_CHANGES`, etc.).

Campos clave:

- `id` (PK)
- `code` (único)
- `name`
- `description`
- `created_at`, `updated_at`

2.4 `role_permissions`

Propósito:

- Relación muchos-a-muchos entre roles y permisos (RBAC).

Campos clave:

- `role_id` (FK -> `roles.id`)
- `permission_id` (FK -> `permissions.id`)
- PK compuesta (`role_id`, `permission_id`)

Justificación:

- Permite crecer el sistema sin hardcodear autorizaciones por rol en cada endpoint.

---

3) Identidad y acceso

3.1 `users`

Propósito:

- Almacena identidad de usuarios y su contexto organizacional.

Campos clave:

- `id` (PK, BIGSERIAL)
- `username` (CITEXT, único, opcional)

- `full_name`
- `email` (CITEXT, único)
- `password_hash`
- `organizational_unit_id` (FK -> `organizational_units.id`)
- `role_id` (FK -> `roles.id`)
- `is_active`
- `last_login_at`
- `created_at`, `updated_at`

Restricciones relevantes:

- `username` opcional pero, si existe, mínimo 3 caracteres.
- `email` único case-insensitive por uso de `CITEXT`.

Justificación:

- Une autenticación (`password_hash`) con autorización contextual (rol + unidad).

---

4) Núcleo operativo de tareas

4.1 Tipo `task_status` (ENUM)

Valores:

- `PENDING`
- `IN_PROGRESS`
- `COMPLETED`

4.2 Tipo `task_priority` (ENUM)

Valores:

- `LOW`
- `MEDIUM`
- `HIGH`

4.3 `tasks`

Propósito:

- Entidad operativa principal del sistema.

Campos clave:

- `id` (PK)
- `organizational_unit_id` (FK -> unidad de la tarea)
- `title`
- `description`
- `status` (`task_status`)

- `priority` (`task_priority`)
- `due_date`
- `assigned_to_user_id` (FK -> `users.id`, opcional)
- `created_by_user_id` (FK -> `users.id`)
- `approved_by_user_id` (FK -> `users.id`, opcional)
- `completed_at`
- `is_active` (soft delete)
- `created_at`, `updated_at`

Restricciones relevantes:

- `title` con longitud mínima.
- `due_date` con límite inferior razonable.
- Soft delete vía `is_active` para preservar historial.

Regla automática de consistencia:

- Trigger `enforce_task_completion_fields`:
  - si `status = COMPLETED` y `completed_at` es null -> set `NOW()`
  - si `status != COMPLETED` -> `completed_at = null`

Justificación:

- Asegura coherencia temporal entre estado y fecha de completado.

---

5) Flujo de solicitudes de cambio

5.1 Tipo `change_type` (ENUM)

Valores:

- `CREATE`
- `UPDATE`
- `COMPLETE`
- `DELETE`

5.2 Tipo `approval_status` (ENUM)

Valores:

- `PENDING`
- `APPROVED`
- `REJECTED`

5.3 `task_change_requests`

Propósito:

- Representa la solicitud formal de cambio sobre una tarea.

Campos clave:

- `id` (PK)
- `task_id` (FK -> `tasks.id`, nullable en `CREATE`)
- `organizational_unit_id` (FK -> unidad de la solicitud)
- `requested_by_user_id` (FK -> solicitante)
- `reviewed_by_user_id` (FK -> supervisor revisor)
- `change_type` (`change_type`)
- `status` (`approval_status`, default `PENDING`)
- `reason`
- `review_comment`
- `payload` (JSONB)
- `requested_at`
- `reviewed_at`
- `created_at`, `updated_at`

Restricción crítica de integridad:

- Si `change_type = CREATE` entonces `task_id` debe ser `NULL`.
- Si `change_type IN (UPDATE, COMPLETE, DELETE)` entonces `task_id` debe existir.

Justificación:

- Obliga coherencia estructural entre tipo de solicitud y entidad afectada.

---

6) Auditoría y trazabilidad

6.1 `task_events`

Propósito:

- Bitácora de acciones relevantes en tareas y solicitudes.

Campos clave:

- `id` (PK)
- `task_id` (FK -> `tasks.id`)
- `change_request_id` (FK -> `task_change_requests.id`)
- `action`
- `actor_user_id` (FK -> `users.id`)
- `details` (JSONB)
- `created_at`

Justificación:

- Provee trazabilidad para soporte, auditoría y análisis histórico.

---

7) Procedimiento transaccional de aprobación

7.1 Función `apply_task_change_request(...)`

Propósito:

- Centralizar en BD la lógica de aprobar/rechazar solicitudes y aplicar cambios a `tasks`.

Parámetros:

- `p_request_id`
- `p_supervisor_user_id`
- `p_decision` (`APPROVED` o `REJECTED`)
- `p_review_comment` (opcional)

Validaciones internas clave:

- La solicitud existe.
- Está en estado `PENDING`.
- El usuario revisor existe y está activo.
- El revisor tiene rol `SUPERVISOR`.
- El supervisor pertenece a la misma unidad de la solicitud.

Comportamiento:

- Si decisión `REJECTED`: actualiza solicitud y termina.
- Si decisión `APPROVED`: aplica operación según `change_type`:
  - `CREATE`: inserta nueva tarea.
  - `UPDATE`: modifica campos en tarea existente.
  - `COMPLETE`: marca tarea como completada.
  - `DELETE`: aplica soft delete (`is_active = false`).
- Registra evento en `task_events`.

Salida:

- `change_request_id`, `status`, `task_id`.

Justificación:

- Evita inconsistencias por lógica repartida entre múltiples llamadas.
- Garantiza atomicidad de aprobación + aplicación + auditoría.

---

8) Vistas de lectura

8.1 `vw_tasks_public`

Propósito:

- Exponer tareas activas con información enriquecida (unidad, creador, aprobador, asignado).

Beneficio:

- Simplifica consultas de lectura para los consumidores de datos.
- Reduce duplicación de joins en múltiples endpoints.

8.2 `vw_pending_change_requests`

Propósito:

- Exponer solicitudes pendientes con datos listos para revisión.

Beneficio:

- Facilita paneles de aprobación y monitoreo de cola.

---

9) Índices y rendimiento

Índices principales:

- `users(email)`, `users(username)`
- `users(organizational_unit_id, role_id)`
- `tasks(organizational_unit_id, status)`
- `tasks(priority)`
- `tasks(created_at DESC)`
- `tasks(is_active)`
- `task_change_requests(status, organizational_unit_id)`
- `task_change_requests(requested_at DESC)`
- `task_change_requests(task_id)`
- `task_change_requests(requested_by_user_id, status, requested_at DESC)`

Justificación:

- Acelera consultas comunes por unidad, estado, historial y bandejas de aprobación.

---

10) Reglas de negocio reflejadas en el esquema

1. Separación de responsabilidades de rol
    - Standard solicita; supervisor decide.
2. Aislamiento por unidad organizacional
    - Las aprobaciones se limitan a la misma unidad.
3. Integridad de estados
    - `ENUM` y constraints evitan combinaciones inválidas.
4. No pérdida de historial
    - Soft delete + eventos + solicitudes auditables.

5. Escalabilidad del dominio
    - RBAC y payload JSONB permiten añadir reglas y nuevos casos con menor fricción.

---

11) Conclusión del modelo de datos

El modelo está orientado a control operativo y auditoría: permite gestionar tareas con un flujo formal de aprobación, manteniendo integridad transaccional y trazabilidad completa.

La combinación de tablas operativas, tablas de flujo y registros de eventos responde de forma directa a los requerimientos funcionales del sistema y deja base sólida para evolución futura.