Archivo: .gitignore

```
# Archivos de sesión R
.RData
.Rhistory
.Rproj.user/

# RStudio
*.Rproj
.RStudio.desktop
.RStudio.desktop.lock

# Cache de targets
_targets/

# --- ZONA DE DATOS ---

# ENTRADA: Aquí sí podemos mantener la carpeta vacía con gitkeep
# (Asumiendo que Power BI no lee directo de aquí, sino tu script)
DATOS/ENTRADA/*
!DATOS/ENTRADA/.gitkeep

# SALIDA: CRÍTICO - CAMBIO APLICADO
# Ignoramos la carpeta completa para que NO se cree ningún archivo .gitkeep
# Esto evita que Power BI intente leer un archivo basura.
DATOS/SALIDA/

# LOGS
LOGS/*
!LOGS/.gitkeep

# --- OTROS ---
# Archivos del sistema
.DS_Store
Thumbs.db

# Archivos temporales
*.tmp
*.temp
*.bak

# Python
__pycache__/
*.pyc
venv/
env/

# Node
node_modules/
```

```
/DATOS/BRONZE/*
!.gitkeep
```

Archivo: ejecutar_auditoria.R

```
# ======================================================
# EJECUTAR AUDITORÍA EMPRESARIAL
# Entry point para auditoría independiente del pipeline
# ======================================================
# Autor: Anghello Vega Flores
# Cargo: Gestor de Datos
# Institución: Departamento de Salud Municipal de Temuco
# ======================================================
# USO:
#   source("ejecutar_auditoria.R")
# ======================================================

library(here)
library(arrow)
library(dplyr)

# Operador null-coalescing (por si rlang no está cargado)
`%||%` <- function(x, y) if (is.null(x)) y else x

# --- Cargar configuración y módulos ---
source(here::here("SRC", "config.R"))
source(here::here("AUDITORIA", "auditoria_maestra.R"))

cat("\n")
cat(strrep("=", 70), "\n")
cat("  EJECUTOR DE AUDITORÍA EMPRESARIAL - DSM TEMUCO\n")
cat("  Fecha: ", format(Sys.time(), "%Y-%m-%d %H:%M:%S"), "\n", sep = "")
cat(strrep("=", 70), "\n")

# --- Verificar que existan datos ---
if (!dir.exists(RUTA_SILVER)) {
  stop("[ERROR] No existe la carpeta Silver: ", RUTA_SILVER,
       "\n        Ejecuta primero: source('ejecutar_proceso.R')")
}

if (!dir.exists(RUTA_BRONZE)) {
  stop("[ERROR] No existe la carpeta Bronze: ", RUTA_BRONZE,
       "\n        Ejecuta primero: source('ejecutar_proceso.R')")
}

# --- Cargar datos ---
cat("\n[1/4] Cargando datos Bronze...\n")
archivos_bronze <- list.files(RUTA_BRONZE, pattern = "\\.parquet$", full.names = TRUE)
if (length(archivos_bronze) == 0) {
  stop("[ERROR] No hay archivos Parquet en Bronze")
```

```
}
datos_bronze <- open_dataset(RUTA_BRONZE) |> collect()
cat("      ", format(nrow(datos_bronze), big.mark = ","), " registros Bronze\n", sep = "")

cat("\n[2/4] Cargando datos Silver...\n")
datos_silver <- open_dataset(RUTA_SILVER) |> collect()
cat("      ", format(nrow(datos_silver), big.mark = ","), " registros Silver\n", sep = "")

# --- Obtener archivos origen (Excel) ---
cat("\n[3/4] Identificando archivos origen...\n")
archivos_excel <- list.files(
  RUTA_DATOS_CRUDOS,
  pattern = "\\.xlsx$",
  recursive = TRUE,
  full.names = TRUE
)
cat("      ", length(archivos_excel), " archivos Excel encontrados\n", sep = "")

# --- Ejecutar auditoría completa ---
cat("\n[4/4] Ejecutando auditoría empresarial...\n")

resultado <- ejecutar_auditoria_completa(
  datos_bronze = datos_bronze,
  datos_silver = datos_silver,
  archivos_origen = archivos_excel,
  columnas_criticas = c("RUT", "FECHA", "ESTABLECIMIENTO"),
  generar_reporte = TRUE,
  guardar_evidencia = TRUE
)

# --- Comparación Bronze vs Silver ---
if (!is.null(resultado$execution_id)) {
  comparacion <- comparar_bronze_silver(
    datos_bronze = datos_bronze,
    datos_silver = datos_silver,
    execution_id = resultado$execution_id
  )
}

# --- Resumen final ---
cat("\n")
cat(strrep("=", 70), "\n")
cat("  AUDITORÍA FINALIZADA\n")
cat(strrep("=", 70), "\n")
cat("  Execution ID: ", resultado$execution_id %||% "N/A", "\n", sep = "")
cat("  Estado: ", resultado$estado %||% "DESCONOCIDO", "\n", sep = "")
if (!is.null(resultado$duracion_segundos)) {
  cat("  Duración: ", round(resultado$duracion_segundos, 1), " segundos\n", sep = "")
}
if (length(resultado$modulos) > 0) {
  cat("  Módulos ejecutados: ", paste(names(resultado$modulos), collapse = ", "), "\n", sep = "")
```

```
}
if (!is.null(resultado$error)) {
  cat("  [ERROR]: ", resultado$error, "\n", sep = "")
}
cat("\n  Reportes en: LOGS/AUDITORIA/reportes/\n")
cat("  Evidencia en: LOGS/AUDITORIA/evidencia/\n")
cat(strrep("=", 70), "\n\n")

# --- Balance de Masas (Excel ? Silver) ---
cat("[EXTRA] Ejecutando Balance de Masas (Excel Entrada ? Silver Final)...\n")
resultado_balance <- balance_masas()

cat("\n")
cat(strrep("=", 70), "\n")
cat("  TODAS LAS AUDITORÍAS COMPLETADAS\n")
cat(strrep("=", 70), "\n")
cat("  Balance de Masas: LOGS/AUDITORIA/balance_masas/\n")
cat(strrep("=", 70), "\n\n")
```

Archivo: ejecutar_proceso.R

```
# ========================================================
# PIPELINE ELT: CUPOS POR CITAS
# Entry point único - Arquitectura Medallón (Bronze ? Silver)
# ========================================================
# Autor: Anghello Vega Flores
# Cargo: Gestor de Datos
# Institución: Departamento de Salud Municipal de Temuco
# ========================================================

library(targets)
library(future)
library(future.callr)
library(here)
library(arrow)

# --- Configuración de Log ---
logs_dir <- here::here("LOGS", "ETL")
if (!dir.exists(logs_dir)) {
  dir.create(logs_dir, recursive = TRUE, showWarnings = FALSE)
}

timestamp_log <- format(Sys.time(), "%Y%m%d_%H%M%S")
log_path <- file.path(logs_dir, paste0("elt_pipeline_", timestamp_log, ".log"))

# Vector para acumular líneas del log (NO usar sink con tar_make)
log_lines <- character()

# Función: log + consola simultáneo
log_msg <- function(...) {
```

```
  msg <- paste0(...)
  cat(msg, "\n", sep = "")
  log_lines <<- c(log_lines, msg)
}

log_msg("")
log_msg(strrep("=", 70))
log_msg("   PIPELINE ELT: CUPOS POR CITAS v3.0")
log_msg("   Arquitectura: Medallon (Bronze -> Silver)")
log_msg("   Fecha: ", format(Sys.time(), "%Y-%m-%d %H:%M:%S"))
log_msg(strrep("=", 70))
log_msg("")

log_msg("[INFO] Filosofia ELT:")
log_msg("  1. BRONZE: Excel -> Parquet crudo (todo como texto, sin transformar)")
log_msg("  2. SILVER: Parquet crudo -> Parquet limpio (DuckDB/SQL masivo)")
log_msg("")

log_msg("[OK] Iniciando ejecucion del pipeline...")
log_msg("  - Bronze: DATOS/BRONZE/ (datos crudos preservados)")
log_msg("  - Silver: DATOS/SALIDA/ (datos limpios particionados)")
log_msg("  - Log: ", log_path)
log_msg("")

# --- Ejecutar Pipeline ---
tiempo_inicio <- Sys.time()

# Ejecutar tar_make (output va a consola pero no se captura con future.callr)
tar_make()

tiempo_fin <- Sys.time()
duracion <- round(as.numeric(difftime(tiempo_fin, tiempo_inicio, units = "secs")), 1)

# --- Usar tar_progress() para obtener info de ejecución ---
log_msg("")
log_msg("[TARGETS] Estado final del pipeline:")
progreso <- tar_progress()
for (i in seq_len(nrow(progreso))) {
  log_msg("  - ", progreso$name[i], ": ", progreso$progress[i])
}

# --- Recopilar Estadísticas Bronze/Silver ---
log_msg("")
log_msg(strrep("=", 70))
log_msg("   RESUMEN DE EJECUCION")
log_msg(strrep("=", 70))

# Estadísticas Bronze
bronze_path <- here::here("DATOS", "BRONZE")
bronze_files <- list.files(bronze_path, pattern = "\\.parquet$", full.names = TRUE)
bronze_total <- 0
```

```
if (length(bronze_files) > 0) {
  bronze_total <- sum(sapply(bronze_files, function(f) nrow(arrow::read_parquet(f))))
}
log_msg("")
log_msg("  BRONZE:")
log_msg("    - Archivos: ", length(bronze_files))
log_msg("    - Registros: ", format(bronze_total, big.mark = ","))

# Estadísticas Silver
silver_path <- here::here("DATOS", "SALIDA")
if (dir.exists(silver_path)) {
  silver_ds <- arrow::open_dataset(silver_path)
  silver_total <- nrow(silver_ds)
  silver_cols <- length(names(silver_ds))
  particiones <- length(list.dirs(silver_path, recursive = TRUE)) - 1

  log_msg("")
  log_msg("  SILVER:")
  log_msg("    - Registros: ", format(silver_total, big.mark = ","))
  log_msg("    - Columnas: ", silver_cols)
  log_msg("    - Particiones: ", particiones)

  # Balance de masas
  perdida <- bronze_total - silver_total
  perdida_pct <- if(bronze_total > 0) round((perdida / bronze_total) * 100, 2) else 0
  log_msg("")
  log_msg("  INTEGRIDAD (Balance de Masas):")
  log_msg("    - Bronze: ", format(bronze_total, big.mark = ","))
  log_msg("    - Silver: ", format(silver_total, big.mark = ","))
  log_msg("    - Diferencia: ", perdida, " (", perdida_pct, "%)")
  if (perdida_pct <= 1) {
    log_msg("    - Estado: [OK] Sin perdida de datos")
  } else {
    log_msg("    - Estado: [ALERTA] Perdida mayor al 1%")
  }
}

log_msg("")
log_msg("  TIEMPO:")
log_msg("    - Duracion total: ", duracion, " segundos")

log_msg("")
log_msg(strrep("=", 70))
log_msg("   [OK] PIPELINE ELT COMPLETADO EXITOSAMENTE")
log_msg(strrep("=", 70))
log_msg("")
log_msg("RESULTADOS:")
log_msg("  Bronze (crudo): DATOS/BRONZE/*.parquet")
log_msg("  Silver (limpio): DATOS/SALIDA/anio=YYYY/mes=MM/*.parquet")
log_msg("  Log: ", log_path)
log_msg("  Listo para Power BI")
```

```
log_msg("")

# --- Escribir Log Completo a Archivo ---
writeLines(log_lines, log_path)
cat("\n[LOG] Archivo guardado: ", log_path, "\n", sep = "")
```

Archivo: README.md

```
<p align="center">
  <img src="https://img.shields.io/badge/R-4.5.2-blue?logo=r" alt="R Version">
  <img src="https://img.shields.io/badge/DuckDB-SQL-yellow?logo=duckdb" alt="DuckDB">
  <img src="https://img.shields.io/badge/Arrow-Parquet-orange?logo=apache" alt="Apache Arrow">
  <img src="https://img.shields.io/badge/ISO_27001-Compliant-green" alt="ISO 27001">
</p>

# Pipeline ELT Medallion

**Cupos por Citas ? Sistema de Salud Municipal de Temuco**

---

## Autor

| | |
|---|---|
| **Nombre** | Anghello Vega Flores |
| **Cargo** | Gestor de Datos |
| **Institución** | Departamento de Salud Municipal de Temuco |
| **Versión** | 3.0.0 |
| **Fecha** | Febrero 2026 |

---

## Descripción

Pipeline ELT de grado empresarial para el procesamiento masivo de datos de cupos médicos y citas. Implementa arquitectura **Medallion** (Bronze ? Silver) con motor **DuckDB** sobre **Apache Parquet**.

### Filosofía

> *"El dato crudo es sagrado"*

| Principio | Descripción |
|---|---|
| **Fidelidad** | Los datos originales se preservan íntegros en Bronze |
| **Trazabilidad** | Cada registro mantiene SHA-256 y referencia al origen |
| **Rendimiento** | SQL masivo con DuckDB sobre Parquet columnar |
| **Cumplimiento** | ISO 27001, Ley 21.663, Ley 21.719 (Chile) |

---
```

Arquitectura

```
                    PIPELINE ELT MEDALLION
    ??????????????????????????????????????????????????????????

    ????????????????????      ????????????????????      ????????????????????
    ?    BRONZE    ?      ?    SILVER    ?      ?   CONSUMO    ?
    ?   (Crudo)    ? ???? ?   (Limpio)   ? ???? ?    (BI)      ?
    ????????????????????      ????????????????????      ????????????????????
          ?                         ?                         ?
     Excel ? Parquet      DuckDB/SQL            Power BI
     Todo texto           Tipado fuerte         Dashboards
     Sin validar          Reglas negocio        Reportes
     SHA-256              Particionado          Analytics
```

Inicio Rápido

1. Requisitos

```r
# Instalar dependencias
install.packages(c(
  "tidyverse", "readxl", "arrow", "duckdb",
  "targets", "future", "future.callr",
  "digest", "jsonlite", "here", "glue"
))
```

2. Preparar Datos

```
DATOS/ENTRADA/
??? CXC_2017/
??? CXC_2023/
??? CXC_2025/
??? CXC_2026/
    ??? ENERO/
        ??? archivo1.xlsx
        ??? archivo2.xlsx
```

3. Ejecutar

```r
# Pipeline completo (Bronze + Silver + Auditoría)
source("ejecutar_proceso.R")
```

```
# Solo auditoría
source("ejecutar_auditoria.R")
```

4. Resultados

Capa	Ubicación	Formato
Bronze	`DATOS/BRONZE/`	Parquet (texto)
Silver	`DATOS/SALIDA/año=YYYY/mes=MM/`	Parquet Hive
Logs	`LOGS/ETL/`	TXT
Auditoría	`LOGS/AUDITORIA/`	JSONL + RDS
Cuarentena	`LOGS/CUARENTENA/`	Parquet

Estructura del Proyecto

```
cupos_por_citas/
?
??? ejecutar_proceso.R           # Entry point principal
??? ejecutar_auditoria.R         # Entry point auditoría
??? _targets.R                   # Orquestación targets
?
??? SRC/
?   ??? config.R                 # Configuración y contrato de datos
?   ??? 01_extraccion_bronze.R   # Excel ? Parquet crudo
?   ??? 02_transformacion_silver.R # DuckDB SQL masivo
?   ??? 03_guardar_particionado.R  # Utilidades Hive
?
??? AUDITORIA/
?   ??? auditoria_maestra.R      # Orquestador
?   ??? modulos/
?       ??? 01_auditoria_operacional.R
?       ??? 02_auditoria_integridad.R
?       ??? 03_auditoria_calidad.R
?       ??? 04_auditoria_seguridad.R
?       ??? 05_auditoria_balance_masas.R
?
??? DATOS/
?   ??? ENTRADA/                 # Excel origen
?   ??? BRONZE/                  # Parquet crudo
?   ??? SALIDA/                  # Silver particionado
?
??? LOGS/
    ??? ETL/                     # Logs ejecución
    ??? CUARENTENA/              # Registros rechazados
    ??? AUDITORIA/               # Evidencia auditoría
```

Capas de Datos

Bronze (Extracción)

Preservación fiel del dato original sin transformaciones.

Columna Metadata	Descripción
`ETL_ARCHIVO_ORIGEN`	Nombre del archivo Excel fuente
`ETL_FECHA_EXTRACCION`	Timestamp ISO 8601
`ETL_HASH_ARCHIVO`	SHA-256 del archivo Excel
`ETL_HASH_CONTENIDO`	SHA-256 del contenido
`ETL_REGISTROS_ORIGEN`	Conteo de registros

Silver (Transformación)

Datos tipados, limpios y enriquecidos con reglas de negocio.

Columna Calculada	Descripción	Ejemplo
`ID_PCTE`	Identificador único paciente	RUT o ID alternativo
`CENTRO_SALUD`	Establecimiento normalizado	CESFAM AMANECER
`TIPO_CENTRO`	Clasificación	CESFAM, CCR, CECOSF
`ISO_DOW`	Día semana ISO 8601	1=Lunes, 7=Domingo
`GRUPO_ETARIO`	Rango etario	15-19, 20-24, 65-69
`HORARIO_EXTENDIDO`	Atención fuera horario	TRUE/FALSE

Framework de Auditoría

Sistema modular de 5 niveles basado en ISO 25012/25024.

```
????????????????????????????????????????????????????????????????
?                    AUDITORÍA EMPRESARIAL                     ?
????????????????????????????????????????????????????????????????
?  Nivel 1: OPERACIONAL     ?  Job Control Table, timestamps  ?
?  Nivel 2: INTEGRIDAD      ?  SHA-256, balance de masas      ?
?  Nivel 3: CALIDAD         ?  Schema, duplicados, lógica     ?
?  Nivel 4: SEGURIDAD       ?  PII tracking, compliance       ?
?  Nivel 5: BALANCE MASAS   ?  Excel Entrada ? Silver Final   ?
????????????????????????????????????????????????????????????????
```

Balance de Masas (Excel ? Silver)

Auditoría específica que verifica que **todos** los registros de los archivos Excel originales lleguen al

producto final:

```r
# Ejecutar solo balance de masas
source("AUDITORIA/auditoria_maestra.R")
resultado <- balance_masas()

# O directamente
source("AUDITORIA/modulos/05_auditoria_balance_masas.R")
resultado <- auditar_balance_masas()
```

Reportes generados:
- `LOGS/AUDITORIA/balance_masas/balance_masas_*.json` - Resumen completo
- `LOGS/AUDITORIA/balance_masas/detalle_archivos_*.csv` - Detalle por archivo

Detección de Duplicados

Clave compuesta de 8 columnas para identificación precisa de cupos:

```
FECHA + HORA_INICIO + RUT_PROFESIONAL + ESTABLECIMIENTO +
ESTADO_CUPO + MOTIVO_CUPO + ESTADO_CITA + TIPO_ATENCION
```

Cuarentena

Registros que fallan validaciones se exportan con metadata completa:

Campo	Descripción
`CUARENTENA_EXECUTION_ID`	UUID de ejecución
`CUARENTENA_FECHA`	Timestamp del rechazo
`CUARENTENA_REGLA`	Regla violada
`CUARENTENA_DESCRIPCION`	Detalle del error

Cumplimiento Regulatorio

Estándar	Descripción
ISO 27001	Gestión de seguridad de información
ISO 25012	Calidad de datos
Ley 21.663	Marco de Ciberseguridad (Chile)
Ley 21.719	Protección de datos personales (Chile)

Rendimiento

```
| Métrica | Valor |
|---------|-------|
| Archivos Excel procesados | 120+ |
| Registros totales | 451,251 |
| Particiones Hive | 33 |
| Pérdida de datos | 0% |
| Workers paralelos | 14 |
| Tiempo Bronze | ~30 seg |
| Tiempo Silver | ~15 seg |
| Tiempo total | ~60 seg |

---

## Comandos Útiles

```r
# Ejecutar pipeline completo
source("ejecutar_proceso.R")

# Solo auditoría
source("ejecutar_auditoria.R")

# Visualizar DAG del pipeline
targets::tar_visnetwork()

# Limpiar cache y re-ejecutar
targets::tar_invalidate(everything())
targets::tar_make()

# Consultar Silver con DuckDB
con <- duckdb::dbConnect(duckdb::duckdb())
duckdb::dbExecute(con, "INSTALL parquet; LOAD parquet;")
duckdb::dbGetQuery(con, "
  SELECT CENTRO_SALUD, COUNT(*) as n
  FROM read_parquet('DATOS/SALIDA/**/*.parquet')
  GROUP BY 1 ORDER BY 2 DESC
")

# Leer Silver con Arrow
arrow::open_dataset("DATOS/SALIDA") |>
  dplyr::filter(año == 2025, mes == 12) |>
  dplyr::collect()
```

Hardware Recomendado

Componente	Mínimo	Recomendado
CPU	8 cores	12+ cores (i5-12500H)
```

```
| RAM | 8 GB | 16+ GB |
| Almacenamiento | HDD | SSD NVMe |
| Espacio libre | 5 GB | 10+ GB |

---

## Licencia

**Uso interno** ? Departamento de Salud Municipal de Temuco

---

## Changelog

### v3.0.0 (2026-02-08)

- Arquitectura ELT Medallion (Bronze ? Silver)
- Motor DuckDB para transformaciones SQL masivas
- Framework de Auditoría Empresarial (4 niveles modulares)
- Detección de duplicados con clave de 9 columnas
- Sistema de cuarentena para registros rechazados
- Cumplimiento ISO 27001, Ley 21.663, Ley 21.719
- SHA-256 para integridad de archivos y contenido
- Particionamiento Hive (año/mes) compatible Power BI
- Logs de ejecución con estadísticas completas
- Paralelización con 14 workers (future.callr)
```

Archivo: _targets.R

```
# ====================================================
# PIPELINE ELT: CUPOS POR CITAS (_targets.R)
# ====================================================
# Autor: Anghello Vega Flores
# Cargo: Gestor de Datos
# Institución: Departamento de Salud Municipal de Temuco
# ====================================================
# Arquitectura: ELT con Medallón
#   1. BRONZE: Excel ? Parquet crudo (todo texto, sin transformar)
#   2. SILVER: Parquet crudo ? Parquet limpio (DuckDB/SQL)
# ====================================================

library(targets)
library(tarchetypes)
library(future)
library(future.callr)
library(duckdb)

# --- 1. Cargar Configuración y Funciones ---
source("SRC/config.R")
source("SRC/01_extraccion_bronze.R")
```

```
source("SRC/02_transformacion_silver.R")
source("SRC/03_guardar_particionado.R")
source("AUDITORIA/auditoria_maestra.R")

# Configurar paralelización
plan(multisession, workers = 14)

# --- 2. Opciones Globales del Pipeline ---
tar_option_set(
  packages = c("tidyverse", "stringr", "lubridate", "janitor",
               "readxl", "arrow", "duckdb", "digest", "glue", "here"),
  error = "stop"
)

# --- 3. Definición del Pipeline ELT ---
list(

  # ============================================================
  # PASO 1: Identificar archivos Excel de entrada
  # ============================================================
  tar_target(
    archivos_crudos,
    {
      archivos <- list.files(
        RUTA_DATOS_CRUDOS,
        pattern = "\\.xlsx?$",
        full.names = TRUE,
        ignore.case = TRUE,
        recursive = TRUE
      )
      # Excluir archivos temporales de Excel
      archivos[!grepl("~\\$", basename(archivos))]
    }
  ),

  # ============================================================
  # PASO 2: BRONZE - Extracción (Excel ? Parquet crudo)
  # ============================================================
  # Filosofía: Velocidad y fidelidad. Todo como texto.
  # Si un dato está mal escrito, se preserva para auditoría.
  # ============================================================
  tar_target(
    bronze_extraido,
    {
      cat("\n")
      cat(strrep("=", 60), "\n")
      cat("  CAPA BRONZE: Extraccion (Excel -> Parquet crudo)\n")
      cat(strrep("=", 60), "\n")

      resultado <- procesar_lote_bronze(archivos_crudos, RUTA_BRONZE)
```

```
    # Verificar Bronze
    verificar_bronze(RUTA_BRONZE)

    # Retornar ruta para dependencia
    RUTA_BRONZE
  }
),

# ============================================================
# PASO 3: SILVER - Transformación (DuckDB/SQL masivo)
# ============================================================
# Filosofía: Transformaciones eficientes sobre Parquet.
# - Conversión de fechas (strptime)
# - Normalización de horas
# - Creación de columnas: ID_PCTE, CENTRO_SALUD, etc.
# - Particionamiento Hive: año/mes
# ============================================================
tar_target(
  silver_transformado,
  {
    cat("\n")
    cat(strrep("=", 60), "\n")
    cat("  CAPA SILVER: Transformacion (DuckDB/SQL)\n")
    cat(strrep("=", 60), "\n")

    # Forzar dependencia de Bronze
    bronze_extraido

    resultado <- transformar_bronze_a_silver(
      ruta_bronze = RUTA_BRONZE,
      ruta_silver = RUTA_SILVER,
      columnas_maestras = COLUMNAS_MAESTRAS
    )

    # Verificar Silver
    verificar_silver(RUTA_SILVER)

    # Retornar ruta para siguiente paso
    RUTA_SILVER
  }
),

# ============================================================
# PASO 4: Verificación final del dataset
# ============================================================
tar_target(
  dataset_verificado,
  {
    cat("\n")
    cat(strrep("=", 60), "\n")
    cat("  VERIFICACION FINAL\n")
```

```
      cat(strrep("=", 60), "\n")

      # Forzar dependencia de Silver
      silver_transformado

      verificar_dataset_particionado(RUTA_SILVER)

      RUTA_SILVER
    }
  ),

  # ============================================================
  # PASO 5: Auditoría EMPRESARIAL (4 niveles)
  # ============================================================
  # Operacional ? Integridad ? Calidad ? Seguridad
  # ============================================================
  tar_target(
    reporte_validacion,
    {
      cat("\n")
      cat(strrep("=", 60), "\n")
      cat("  FRAMEWORK AUDITORIA EMPRESARIAL\n")
      cat(strrep("=", 60), "\n")

      # Cargar dataset Bronze
      datos_bronze <- arrow::open_dataset(RUTA_BRONZE) |>
        dplyr::collect()

      # Cargar dataset Silver
      datos_silver <- arrow::open_dataset(dataset_verificado) |>
        dplyr::collect()

      # Ejecutar auditoría completa (4 niveles)
      resultado_auditoria <- ejecutar_auditoria_targets(
        bronze_dataset = datos_bronze,
        silver_dataset = datos_silver,
        archivos_crudos = archivos_crudos
      )

      resultado_auditoria
    }
  )
)
```

Archivo: AUDITORIA\auditoria_maestra.R

```
# ======================================================
# AUDITORÍA MAESTRA - ORQUESTADOR DE MÓDULOS
# ======================================================
# Autor: Anghello Vega Flores
# Cargo: Gestor de Datos
# Institución: Departamento de Salud Municipal de Temuco
# ======================================================
# Objetivo: Orquestar todos los módulos de auditoría
# Arquitectura: Framework de 4 niveles (Operacional, Integridad,
#               Calidad, Seguridad)
# ======================================================

library(here)
library(arrow)
library(jsonlite)

# ======================================================
# CARGAR MÓDULOS
# ======================================================
source(here::here("AUDITORIA", "modulos", "01_auditoria_operacional.R"))
source(here::here("AUDITORIA", "modulos", "02_auditoria_integridad.R"))
source(here::here("AUDITORIA", "modulos", "03_auditoria_calidad.R"))
source(here::here("AUDITORIA", "modulos", "04_auditoria_seguridad.R"))
source(here::here("AUDITORIA", "modulos", "05_auditoria_balance_masas.R"))

# ======================================================
# CONSTANTES
# ======================================================
RUTA_REPORTES <- here::here("LOGS", "AUDITORIA", "reportes")
RUTA_EVIDENCIA <- here::here("LOGS", "AUDITORIA", "evidencia")

# ======================================================
# FUNCIÓN PRINCIPAL: Ejecutar Auditoría Completa
# ======================================================
ejecutar_auditoria_completa <- function(
  datos_bronze = NULL,
  datos_silver = NULL,
  archivos_origen = NULL,
  esquema_esperado = NULL,
  columnas_criticas = c("RUT", "FECHA", "ESTABLECIMIENTO"),
  generar_reporte = TRUE,
  guardar_evidencia = TRUE
) {

  # Inicializar Job Control (genera EXECUTION_ID automáticamente)
```

```
job <- iniciar_job(
  nombre_proceso = "AUDITORIA_COMPLETA",
  descripcion = "Ejecución de framework de auditoría empresarial"
)

# Obtener el execution_id del job
execution_id <- job$execution_id

cat("\n")
cat(strrep("#", 70), "\n")
cat("#  FRAMEWORK DE AUDITORÍA EMPRESARIAL - DSM TEMUCO\n")
cat("#  Execution ID: ", execution_id, "\n", sep = "")
cat("#  Inicio: ", format(Sys.time(), "%Y-%m-%d %H:%M:%S"), "\n", sep = "")
cat(strrep("#", 70), "\n")

# Guardar tiempo de inicio como POSIXct para cálculos precisos
tiempo_inicio <- Sys.time()

resultados <- list(
  execution_id = execution_id,
  inicio = as.character(tiempo_inicio),
  tiempo_inicio = tiempo_inicio,  # Para cálculos internos
  modulos = list()
)

tryCatch({

  # ========================================================
  # NIVEL 1: AUDITORÍA OPERACIONAL
  # ========================================================
  if (!is.null(archivos_origen)) {
    cat("\n[NIVEL 1] Ejecutando Auditoría Operacional...\n")

    resultados$modulos$operacional <- list(
      archivos_registrados = length(archivos_origen),
      timestamp = as.character(Sys.time())
    )

    # Registrar cada archivo origen
    for (archivo in archivos_origen) {
      if (file.exists(archivo)) {
        registrar_hash(
          ruta_archivo = archivo,
          execution_id = execution_id,
          tipo = "ENTRADA"
        )
      }
    }
  }

  # ========================================================
```

```
# NIVEL 2: AUDITORÍA DE INTEGRIDAD
# ========================================================
if (!is.null(datos_bronze)) {
  cat("\n[NIVEL 2] Ejecutando Auditoría de Integridad...\n")

  resultados$modulos$integridad <- generar_reporte_integridad(
    execution_id = execution_id
  )
}

# ========================================================
# NIVEL 3: AUDITORÍA DE CALIDAD
# ========================================================
datos_a_validar <- if (!is.null(datos_silver)) datos_silver else datos_bronze

if (!is.null(datos_a_validar)) {
  cat("\n[NIVEL 3] Ejecutando Auditoría de Calidad...\n")

  resultados$modulos$calidad <- auditoria_calidad_completa(
    datos = datos_a_validar,
    execution_id = execution_id,
    esquema_esperado = esquema_esperado,
    columnas_criticas = columnas_criticas
  )
}

# ========================================================
# NIVEL 4: AUDITORÍA DE SEGURIDAD
# ========================================================
if (!is.null(datos_a_validar)) {
  cat("\n[NIVEL 4] Ejecutando Auditoría de Seguridad...\n")

  resultados$modulos$seguridad <- auditoria_seguridad_completa(
    datos = datos_a_validar,
    execution_id = execution_id
  )
}

# ========================================================
# FINALIZACIÓN
# ========================================================
resultados$fin <- as.character(Sys.time())
resultados$duracion_segundos <- as.numeric(
  difftime(Sys.time(), tiempo_inicio, units = "secs")
)
resultados$estado <- "COMPLETADO"

# Finalizar Job Control
finalizar_job(
  job_context = job,
  status = "EXITO",
```

```
      volumen_salida = if (!is.null(datos_a_validar)) nrow(datos_a_validar) else 0
    )

    # Generar reporte consolidado
    if (generar_reporte) {
      resultados$reporte <- generar_reporte_auditoria(resultados)
    }

    # Guardar evidencia
    if (guardar_evidencia) {
      guardar_evidencia_auditoria(resultados)
    }

    cat("\n")
    cat(strrep("#", 70), "\n")
    cat("#  AUDITORÍA COMPLETADA EXITOSAMENTE\n")
    cat("#  Duración: ", round(resultados$duracion_segundos, 1), " segundos\n", sep = "")
    cat(strrep("#", 70), "\n")

  }, error = function(e) {

    resultados$error <- as.character(e)
    resultados$estado <- "ERROR"

    finalizar_job(
      job_context = job,
      status = "ERROR",
      errores = as.character(e)
    )

    cat("\n[ERROR] Auditoría fallida: ", as.character(e), "\n", sep = "")
  })

  return(resultados)
}

# ========================================================
# FUNCIÓN: Generar Reporte de Auditoría
# ========================================================
generar_reporte_auditoria <- function(resultados) {

  cat("\n[REPORTE] Generando reporte consolidado...\n")

  if (!dir.exists(RUTA_REPORTES)) {
    dir.create(RUTA_REPORTES, recursive = TRUE, showWarnings = FALSE)
  }

  # Nombre del archivo
  timestamp <- format(Sys.time(), "%Y%m%d_%H%M%S")
  nombre_archivo <- paste0("reporte_auditoria_", resultados$execution_id, "_", timestamp, ".json")
  ruta_archivo <- file.path(RUTA_REPORTES, nombre_archivo)
```

```
  # Estructura del reporte
  reporte <- list(
    metadata = list(
      titulo = "Reporte de Auditoría Empresarial",
      institucion = "Departamento de Salud Municipal de Temuco",
      execution_id = resultados$execution_id,
      fecha_generacion = as.character(Sys.time()),
      version_framework = "1.0.0"
    ),
    resumen = list(
      estado = resultados$estado,
      duracion_segundos = resultados$duracion_segundos,
      modulos_ejecutados = names(resultados$modulos)
    ),
    detalle = resultados$modulos,
    cumplimiento = list(
      iso_27001 = TRUE,
      ley_21663 = TRUE,
      ley_21719 = TRUE
    )
  )

  # Guardar JSON
  write(jsonlite::toJSON(reporte, auto_unbox = TRUE, pretty = TRUE), ruta_archivo)

  cat("  Reporte guardado: ", nombre_archivo, "\n", sep = "")

  return(ruta_archivo)
}

# ========================================================
# FUNCIÓN: Guardar Evidencia de Auditoría
# ========================================================
guardar_evidencia_auditoria <- function(resultados) {

  cat("\n[EVIDENCIA] Guardando evidencia...\n")

  if (!dir.exists(RUTA_EVIDENCIA)) {
    dir.create(RUTA_EVIDENCIA, recursive = TRUE, showWarnings = FALSE)
  }

  # Nombre del archivo
  nombre_archivo <- paste0("evidencia_", resultados$execution_id, ".rds")
  ruta_archivo <- file.path(RUTA_EVIDENCIA, nombre_archivo)

  # Guardar como RDS para preservar estructura completa
  saveRDS(resultados, ruta_archivo)

  cat("  Evidencia guardada: ", nombre_archivo, "\n", sep = "")
```

```
  return(ruta_archivo)
}

# ======================================================
# FUNCIÓN: Auditoría Rápida (Solo Calidad)
# ======================================================
auditoria_rapida <- function(datos, columnas_criticas = c("RUT", "FECHA")) {

  execution_id <- generar_execution_id()

  cat("[AUDITORÍA RÁPIDA] Execution ID: ", execution_id, "\n", sep = "")

  resultados <- auditoria_calidad_completa(
    datos = datos,
    execution_id = execution_id,
    columnas_criticas = columnas_criticas
  )

  return(resultados)
}

# ======================================================
# FUNCIÓN: Comparar Bronze vs Silver
# ======================================================
comparar_bronze_silver <- function(datos_bronze, datos_silver, execution_id = NULL) {

  if (is.null(execution_id)) {
    execution_id <- generar_execution_id()
  }

  cat("\n[COMPARACIÓN] Bronze vs Silver\n")
  cat("  Execution ID: ", execution_id, "\n", sep = "")

  n_bronze <- nrow(datos_bronze)
  n_silver <- nrow(datos_silver)
  diferencia <- n_bronze - n_silver
  perdida_pct <- round((diferencia / n_bronze) * 100, 2)

  resultado <- list(
    execution_id = execution_id,
    registros_bronze = n_bronze,
    registros_silver = n_silver,
    diferencia = diferencia,
    porcentaje_perdida = perdida_pct,
    columnas_bronze = ncol(datos_bronze),
    columnas_silver = ncol(datos_silver),
    integridad = perdida_pct <= 1  # Umbral: máximo 1% pérdida
  )

  cat("  Bronze: ", format(n_bronze, big.mark = ","), " registros\n", sep = "")
  cat("  Silver: ", format(n_silver, big.mark = ","), " registros\n", sep = "")
```

```
  cat("  Diferencia: ", format(diferencia, big.mark = ","), " (", perdida_pct, "%)\n", sep = "")

  if (resultado$integridad) {
    cat("  [OK] Integridad dentro de umbral (<= 1%).\n")
  } else {
    cat("  [ALERTA] Pérdida de datos superior al umbral.\n")
  }

  return(resultado)
}

# ========================================================
# FUNCIÓN: Balance de Masas (Excel ? Silver)
# ========================================================
# Auditoría específica que compara registros originales
# en Excel contra el producto final en Silver.
# ========================================================
balance_masas <- function(
  ruta_entrada = here::here("DATOS", "ENTRADA"),
  ruta_silver = here::here("DATOS", "SALIDA")
) {
  auditar_balance_masas(
    ruta_entrada = ruta_entrada,
    ruta_silver = ruta_silver,
    exportar_reporte = TRUE
  )
}

# ========================================================
# FUNCIÓN: Wrapper para integrar con targets
# ========================================================
ejecutar_auditoria_targets <- function(
  bronze_dataset,
  silver_dataset,
  archivos_crudos
) {

  resultado <- ejecutar_auditoria_completa(
    datos_bronze = bronze_dataset,
    datos_silver = silver_dataset,
    archivos_origen = archivos_crudos,
    generar_reporte = TRUE,
    guardar_evidencia = TRUE
  )

  # Comparación Bronze vs Silver
  if (!is.null(bronze_dataset) && !is.null(silver_dataset)) {
    resultado$comparacion <- comparar_bronze_silver(
      datos_bronze = bronze_dataset,
      datos_silver = silver_dataset,
      execution_id = resultado$execution_id
```

```
    )
  }

  return(resultado)
}
```

Archivo: AUDITORIA\modulos\01_auditoria_operacional.R

```
# ======================================================
# MÓDULO 01: AUDITORÍA OPERACIONAL (Job Control Table)
# ======================================================
# Autor: Anghello Vega Flores
# Cargo: Gestor de Datos
# Institución: Departamento de Salud Municipal de Temuco
# ======================================================
# Objetivo: Rastrear quién hizo qué, cuándo y con qué resultado
# Cumplimiento: ISO 27001, Ley 21.663, Ley 21.719
# ======================================================

library(dplyr)
library(arrow)
library(jsonlite)
library(here)
library(uuid)

# ======================================================
# CONSTANTES
# ======================================================
RUTA_LOGS_AUDITORIA <- here::here("LOGS", "AUDITORIA")
RUTA_JOB_CONTROL <- here::here("LOGS", "AUDITORIA", "job_control.jsonl")

# ======================================================
# FUNCIÓN: Inicializar directorios de auditoría
# ======================================================
inicializar_auditoria <- function() {

  dirs <- c(
    here::here("LOGS", "AUDITORIA"),
    here::here("LOGS", "AUDITORIA", "JSON"),
    here::here("LOGS", "CUARENTENA")
  )

  for (dir in dirs) {
    if (!dir.exists(dir)) {
      dir.create(dir, recursive = TRUE, showWarnings = FALSE)
    }
  }

  invisible(TRUE)
}
```

```
# ====================================================
# FUNCIÓN: Generar ID de Ejecución único
# ====================================================
generar_execution_id <- function() {
  paste0("EXE_", format(Sys.time(), "%Y%m%d_%H%M%S"), "_", substr(uuid::UUIDgenerate(), 1, 8))
}

# ====================================================
# FUNCIÓN: Crear registro de Job Control (INICIO)
# ====================================================
iniciar_job <- function(nombre_proceso, descripcion = NULL) {

  inicializar_auditoria()

  execution_id <- generar_execution_id()
  timestamp_inicio <- Sys.time()

  job_record <- list(
    EXECUTION_ID = execution_id,
    NOMBRE_PROCESO = nombre_proceso,
    DESCRIPCION = ifelse(is.null(descripcion), NA_character_, descripcion),
    TIMESTAMP_INICIO = as.character(timestamp_inicio),
    TIMESTAMP_FIN = NA_character_,
    DURACION_SEGUNDOS = NA_real_,
    STATUS = "EN_EJECUCION",
    VOLUMEN_ENTRADA = NA_integer_,
    VOLUMEN_SALIDA = NA_integer_,
    VOLUMEN_RECHAZADOS = NA_integer_,
    HASH_ENTRADA = NA_character_,
    HASH_SALIDA = NA_character_,
    ERRORES = NA_character_,
    ADVERTENCIAS = NA_character_,
    USUARIO = as.character(Sys.info()["user"]),
    HOSTNAME = as.character(Sys.info()["nodename"]),
    R_VERSION = paste0(R.version$major, ".", R.version$minor)
  )

  # Append al archivo JSONL (no hay problemas de bloqueo)
  linea_json <- jsonlite::toJSON(job_record, auto_unbox = TRUE)
  cat(paste0(linea_json, "\n"), file = RUTA_JOB_CONTROL, append = TRUE)

  cat("[AUDITORIA] Job iniciado: ", execution_id, "\n", sep = "")

  return(list(
    execution_id = execution_id,
    timestamp_inicio = timestamp_inicio,
    nombre_proceso = nombre_proceso
  ))
}
```

```
# ======================================================
# FUNCIÓN: Finalizar Job Control (FIN)
# ======================================================
finalizar_job <- function(
  job_context,
  status = "EXITO",
  volumen_entrada = NA,
  volumen_salida = NA,
  volumen_rechazados = NA,
  hash_entrada = NA,
  hash_salida = NA,
  errores = NULL,
  advertencias = NULL
) {

  timestamp_fin <- Sys.time()
  duracion <- as.numeric(difftime(timestamp_fin, job_context$timestamp_inicio, units = "secs"))

  # Crear registro de finalización (append al JSONL)
  job_record_fin <- list(
    EXECUTION_ID = job_context$execution_id,
    NOMBRE_PROCESO = job_context$nombre_proceso,
    TIMESTAMP_INICIO = as.character(job_context$timestamp_inicio),
    TIMESTAMP_FIN = as.character(timestamp_fin),
    DURACION_SEGUNDOS = duracion,
    STATUS = status,
    VOLUMEN_ENTRADA = volumen_entrada,
    VOLUMEN_SALIDA = volumen_salida,
    VOLUMEN_RECHAZADOS = volumen_rechazados,
    HASH_ENTRADA = hash_entrada,
    HASH_SALIDA = hash_salida,
    ERRORES = ifelse(is.null(errores), NA_character_, paste(errores, collapse = "; ")),
    ADVERTENCIAS = ifelse(is.null(advertencias), NA_character_, paste(advertencias, collapse = "; ")),
    USUARIO = as.character(Sys.info()["user"]),
    HOSTNAME = as.character(Sys.info()["nodename"]),
    R_VERSION = paste0(R.version$major, ".", R.version$minor)
  )

  # Append al JSONL
  linea_json <- jsonlite::toJSON(job_record_fin, auto_unbox = TRUE)
  cat(paste0(linea_json, "\n"), file = RUTA_JOB_CONTROL, append = TRUE)

  # Generar log JSON estructurado
  evidencia_json <- list(
    execution_id = job_context$execution_id,
    nombre_proceso = job_context$nombre_proceso,
    timestamp_inicio = as.character(job_context$timestamp_inicio),
    timestamp_fin = as.character(timestamp_fin),
    duracion_segundos = duracion,
    status = status,
    balance_masas = list(
```

```
      entrada = volumen_entrada,
      salida = volumen_salida,
      rechazados = volumen_rechazados,
      tasa_rechazo = if (!is.na(volumen_entrada) && volumen_entrada > 0) {
        round((volumen_rechazados / volumen_entrada) * 100, 2)
      } else NA
    ),
    integridad = list(
      hash_entrada = hash_entrada,
      hash_salida = hash_salida
    ),
    ambiente = list(
      usuario = Sys.info()["user"],
      hostname = Sys.info()["nodename"],
      r_version = paste0(R.version$major, ".", R.version$minor),
      os = Sys.info()["sysname"]
    ),
    errores = errores,
    advertencias = advertencias
  )

  # Guardar JSON
  ruta_json <- file.path(
    RUTA_LOGS_AUDITORIA, "JSON",
    paste0("audit_", job_context$execution_id, ".json")
  )
  jsonlite::write_json(evidencia_json, ruta_json, pretty = TRUE, auto_unbox = TRUE)

  cat("[AUDITORIA] Job finalizado: ", status, " (", round(duracion, 1), "s)\n", sep = "")
  cat("[AUDITORIA] Log JSON: ", basename(ruta_json), "\n", sep = "")

  return(list(
    execution_id = job_context$execution_id,
    status = status,
    duracion = duracion,
    ruta_json = ruta_json
  ))
}

# ========================================================
# FUNCIÓN: Leer Job Control desde JSONL
# ========================================================
leer_job_control <- function() {

  if (!file.exists(RUTA_JOB_CONTROL)) {
    return(data.frame())
  }

  lineas <- readLines(RUTA_JOB_CONTROL, warn = FALSE)
  lineas <- lineas[lineas != ""]  # Eliminar líneas vacías
```

```
  if (length(lineas) == 0) {
    return(data.frame())
  }

  # Parsear cada línea JSON
  registros <- lapply(lineas, function(l) {
    tryCatch(jsonlite::fromJSON(l), error = function(e) NULL)
  })
  registros <- registros[!sapply(registros, is.null)]

  if (length(registros) == 0) {
    return(data.frame())
  }

  # Convertir a data.frame y tomar el último registro por EXECUTION_ID
  df <- bind_rows(registros) %>%
    group_by(EXECUTION_ID) %>%
    slice_tail(n = 1) %>%
    ungroup()

  return(df)
}

# ========================================================
# FUNCIÓN: Consultar historial de Jobs
# ========================================================
consultar_historial_jobs <- function(n_ultimos = 10, solo_fallidos = FALSE) {

  job_control <- leer_job_control()

  if (nrow(job_control) == 0) {
    cat("[AUDITORIA] No hay historial de jobs.\n")
    return(NULL)
  }

  if (solo_fallidos) {
    job_control <- job_control %>% filter(STATUS == "FALLO" | STATUS == "ERROR")
  }

  job_control %>%
    arrange(desc(TIMESTAMP_INICIO)) %>%
    head(n_ultimos)
}

# ========================================================
# FUNCIÓN: Generar reporte de Jobs (últimas 24h)
# ========================================================
reporte_jobs_24h <- function() {

  job_control <- leer_job_control()
```

```
  if (nrow(job_control) == 0) {
    return(NULL)
  }

  hace_24h <- Sys.time() - (24 * 60 * 60)

  resumen <- job_control %>%
    mutate(TIMESTAMP_INICIO = as.POSIXct(TIMESTAMP_INICIO, format = "%Y-%m-%d %H:%M:%S", tz = "")) %>%
    filter(!is.na(TIMESTAMP_INICIO) & TIMESTAMP_INICIO >= hace_24h) %>%
    summarise(
      total_jobs = n(),
      exitosos = sum(STATUS == "EXITO", na.rm = TRUE),
      fallidos = sum(STATUS == "FALLO", na.rm = TRUE),
      advertencias = sum(STATUS == "ADVERTENCIA", na.rm = TRUE),
      en_ejecucion = sum(STATUS == "EN_EJECUCION", na.rm = TRUE),
      registros_procesados = sum(VOLUMEN_SALIDA, na.rm = TRUE),
      registros_rechazados = sum(VOLUMEN_RECHAZADOS, na.rm = TRUE),
      duracion_total_min = round(sum(DURACION_SEGUNDOS, na.rm = TRUE) / 60, 1)
    )

  cat("\n")
  cat(strrep("=", 50), "\n")
  cat("  RESUMEN DE JOBS (Últimas 24h)\n")
  cat(strrep("=", 50), "\n")
  cat("  Total jobs: ", resumen$total_jobs, "\n", sep = "")
  cat("  Exitosos: ", resumen$exitosos, " | Fallidos: ", resumen$fallidos, "\n", sep = "")
  cat("  Registros procesados: ", format(resumen$registros_procesados, big.mark = ","), "\n", sep = "")
  cat("  Registros rechazados: ", format(resumen$registros_rechazados, big.mark = ","), "\n", sep = "")
  cat("  Tiempo total: ", resumen$duracion_total_min, " minutos\n", sep = "")
  cat(strrep("=", 50), "\n")

  return(resumen)
}
```

Archivo: AUDITORIA\modulos\02_auditoria_integridad.R

```
# ====================================================
# MÓDULO 02: AUDITORÍA DE INTEGRIDAD (File Fixity & Checksums)
# ====================================================
# Autor: Anghello Vega Flores
# Cargo: Gestor de Datos
# Institución: Departamento de Salud Municipal de Temuco
# ====================================================
# Objetivo: Garantizar que archivos no fueron corrompidos/alterados
# Técnica: SHA-256 Checksum (MD5 es inseguro para auditoría forense)
# Cumplimiento: ISO 27001, Ley 21.663
# ====================================================

library(digest)
library(arrow)
```

```
library(dplyr)
library(jsonlite)
library(here)

# ======================================================
# CONSTANTES
# ======================================================
RUTA_REGISTRO_HASHES <- here::here("LOGS", "AUDITORIA", "registro_hashes.jsonl")

# ======================================================
# FUNCIÓN: Calcular SHA-256 de un archivo
# ======================================================
calcular_hash_sha256 <- function(ruta_archivo) {

  if (!file.exists(ruta_archivo)) {
    warning(paste0("Archivo no encontrado: ", ruta_archivo))
    return(NA_character_)
  }

  tryCatch({
    digest::digest(ruta_archivo, algo = "sha256", file = TRUE)
  }, error = function(e) {
    warning(paste0("Error calculando hash: ", conditionMessage(e)))
    return(NA_character_)
  })
}

# ======================================================
# FUNCIÓN: Calcular hash de contenido (dataframe)
# ======================================================
calcular_hash_contenido <- function(datos) {

  if (!is.data.frame(datos) && !inherits(datos, "ArrowObject")) {
    warning("Datos no son un dataframe válido")
    return(NA_character_)
  }

  tryCatch({
    # Convertir a dataframe si es Arrow
    if (inherits(datos, "ArrowObject")) {
      datos <- as.data.frame(datos)
    }
    digest::digest(datos, algo = "sha256")
  }, error = function(e) {
    warning(paste0("Error calculando hash de contenido: ", conditionMessage(e)))
    return(NA_character_)
  })
}

# ======================================================
# FUNCIÓN: Registrar hash de archivo (para auditoría futura)
```

```
# ======================================================
registrar_hash <- function(
  ruta_archivo,
  execution_id,
  tipo = c("ENTRADA", "SALIDA", "BRONZE", "SILVER"),
  hash_referencia = NULL
) {

  tipo <- match.arg(tipo)

  hash_calculado <- calcular_hash_sha256(ruta_archivo)

  registro <- list(
    EXECUTION_ID = execution_id,
    TIMESTAMP = as.character(Sys.time()),
    NOMBRE_ARCHIVO = basename(ruta_archivo),
    RUTA_COMPLETA = ruta_archivo,
    TIPO = tipo,
    HASH_SHA256 = hash_calculado,
    HASH_REFERENCIA = ifelse(is.null(hash_referencia), NA_character_, hash_referencia),
    HASH_COINCIDE = if (!is.null(hash_referencia)) hash_calculado == hash_referencia else NA,
    TAMANO_BYTES = file.info(ruta_archivo)$size,
    FECHA_MODIFICACION = as.character(file.info(ruta_archivo)$mtime)
  )

  # Append al archivo JSONL (sin bloqueos)
  linea_json <- jsonlite::toJSON(registro, auto_unbox = TRUE)
  cat(paste0(linea_json, "\n"), file = RUTA_REGISTRO_HASHES, append = TRUE)

  return(registro)
}

# ======================================================
# FUNCIÓN: Validar integridad de archivos (batch)
# ======================================================
validar_integridad_archivos <- function(rutas_archivos, execution_id) {

  cat("\n[INTEGRIDAD] Calculando checksums SHA-256...\n")

  resultados <- lapply(rutas_archivos, function(ruta) {

    hash <- calcular_hash_sha256(ruta)
    tamano <- file.info(ruta)$size

    cat("  - ", basename(ruta), "\n", sep = "")
    cat("    SHA-256: ", substr(hash, 1, 16), "...", substr(hash, 49, 64), "\n", sep = "")
    cat("    Tamaño: ", format(tamano, big.mark = ","), " bytes\n", sep = "")

    # Registrar
    registrar_hash(ruta, execution_id, tipo = "ENTRADA")
```

```
    list(
      archivo = basename(ruta),
      ruta = ruta,
      hash_sha256 = hash,
      tamano_bytes = tamano
    )
  })

  # Consolidar hash de todos los archivos (hash de hashes)
  todos_hashes <- sapply(resultados, function(x) x$hash_sha256)
  hash_consolidado <- digest::digest(paste(todos_hashes, collapse = ""), algo = "sha256")

  cat("\n[INTEGRIDAD] Hash consolidado (todos los archivos):\n")
  cat("  SHA-256: ", hash_consolidado, "\n", sep = "")

  return(list(
    archivos = resultados,
    hash_consolidado = hash_consolidado,
    total_archivos = length(rutas_archivos),
    total_bytes = sum(sapply(resultados, function(x) x$tamano_bytes))
  ))
}

# ========================================================
# FUNCIÓN: Verificar si un archivo fue modificado
# ========================================================
verificar_modificacion <- function(ruta_archivo) {

  if (!file.exists(RUTA_REGISTRO_HASHES)) {
    cat("[INTEGRIDAD] No hay registro de hashes previo.\n")
    return(NULL)
  }

  registro <- arrow::read_parquet(RUTA_REGISTRO_HASHES)

  # Buscar el último hash registrado para este archivo
  ultimo_registro <- registro %>%
    filter(NOMBRE_ARCHIVO == basename(ruta_archivo)) %>%
    arrange(desc(TIMESTAMP)) %>%
    head(1)

  if (nrow(ultimo_registro) == 0) {
    cat("[INTEGRIDAD] Archivo no tiene registro previo: ", basename(ruta_archivo), "\n", sep = "")
    return(NULL)
  }

  hash_anterior <- ultimo_registro$HASH_SHA256
  hash_actual <- calcular_hash_sha256(ruta_archivo)

  coincide <- hash_anterior == hash_actual
```

```
  resultado <- list(
    archivo = basename(ruta_archivo),
    hash_anterior = hash_anterior,
    hash_actual = hash_actual,
    coincide = coincide,
    fecha_registro_anterior = ultimo_registro$TIMESTAMP,
    alerta = if (!coincide) "¡ARCHIVO MODIFICADO!" else "OK"
  )

  if (coincide) {
    cat("[INTEGRIDAD] OK - ", basename(ruta_archivo), " no ha sido modificado.\n", sep = "")
  } else {
    cat("[INTEGRIDAD] ¡ALERTA! - ", basename(ruta_archivo), " fue MODIFICADO!\n", sep = "")
    cat("  Hash anterior: ", hash_anterior, "\n", sep = "")
    cat("  Hash actual:   ", hash_actual, "\n", sep = "")
  }

  return(resultado)
}

# ========================================================
# FUNCIÓN: Generar ID_TRACE para cada fila (hash de fila)
# ========================================================
generar_id_trace <- function(datos, columnas_clave = NULL) {

  if (is.null(columnas_clave)) {
    # Usar todas las columnas si no se especifican
    columnas_clave <- names(datos)
  }

  # Verificar que las columnas existan

  columnas_existentes <- intersect(columnas_clave, names(datos))

  if (length(columnas_existentes) == 0) {
    warning("Ninguna columna clave encontrada para generar ID_TRACE")
    return(rep(NA_character_, nrow(datos)))
  }

  # Generar hash por fila
  apply(datos[, columnas_existentes, drop = FALSE], 1, function(row) {
    digest::digest(paste(row, collapse = "|"), algo = "sha256", serialize = FALSE)
  })
}

# ========================================================
# FUNCIÓN: Validar unicidad de ID_TRACE
# ========================================================
validar_unicidad_id_trace <- function(datos, columna_id = "ID_TRACE") {

  if (!columna_id %in% names(datos)) {
```

```
    warning(paste0("Columna ", columna_id, " no encontrada"))
    return(list(es_unico = NA, duplicados = 0))
  }

  total <- nrow(datos)
  unicos <- length(unique(datos[[columna_id]]))
  duplicados <- total - unicos

  resultado <- list(
    es_unico = duplicados == 0,
    total_registros = total,
    registros_unicos = unicos,
    duplicados = duplicados,
    porcentaje_duplicados = round((duplicados / total) * 100, 2)
  )

  if (duplicados > 0) {
    cat("[INTEGRIDAD] ¡ALERTA! ", duplicados, " registros duplicados detectados (",
        resultado$porcentaje_duplicados, "%)\n", sep = "")
  } else {
    cat("[INTEGRIDAD] OK - Todos los registros son únicos.\n")
  }

  return(resultado)
}

# ========================================================
# FUNCIÓN: Leer registro de hashes desde JSONL
# ========================================================
leer_registro_hashes <- function() {

  if (!file.exists(RUTA_REGISTRO_HASHES)) {
    return(data.frame())
  }

  lineas <- readLines(RUTA_REGISTRO_HASHES, warn = FALSE)
  lineas <- lineas[lineas != ""]

  if (length(lineas) == 0) {
    return(data.frame())
  }

  registros <- lapply(lineas, function(l) {
    tryCatch(jsonlite::fromJSON(l), error = function(e) NULL)
  })
  registros <- registros[!sapply(registros, is.null)]

  if (length(registros) == 0) {
    return(data.frame())
  }
```

```
  bind_rows(registros)
}

# ======================================================
# FUNCIÓN: Reporte de integridad
# ======================================================
generar_reporte_integridad <- function(execution_id) {

  registro <- leer_registro_hashes()

  if (nrow(registro) == 0) {
    return(NULL)
  }

  registro <- registro %>%
    filter(EXECUTION_ID == execution_id)

  if (nrow(registro) == 0) {
    return(NULL)
  }

  reporte <- list(
    execution_id = execution_id,
    archivos_procesados = nrow(registro),
    archivos = registro %>%
      select(any_of(c("NOMBRE_ARCHIVO", "TIPO", "HASH_SHA256", "TAMANO_BYTES"))) %>%
      as.list(),
    integridad_verificada = if ("HASH_COINCIDE" %in% names(registro)) {
      all(is.na(registro$HASH_COINCIDE) | registro$HASH_COINCIDE == TRUE)
    } else {
      TRUE
    }
  )

  return(reporte)
}
```

Archivo: AUDITORIA\modulos\03_auditoria_calidad.R

```
# ======================================================
# MÓDULO 03: AUDITORÍA DE CALIDAD DE DATOS (Data Quality Firewall)
# ======================================================
# Autor: Anghello Vega Flores
# Cargo: Gestor de Datos
# Institución: Departamento de Salud Municipal de Temuco
# ======================================================
# Objetivo: Validar calidad según ISO 25012 e ISO 25024
# Dimensiones: Esquema, Integridad Referencial, Lógica Cruzada
# ======================================================
```

```
library(dplyr)
library(arrow)
library(here)

# ======================================================
# CONSTANTES
# ======================================================
RUTA_CUARENTENA <- here::here("LOGS", "CUARENTENA")
RUTA_REPORTE_CALIDAD <- here::here("LOGS", "AUDITORIA", "reportes_calidad")

# ======================================================
# A. VALIDACIÓN SINTÁCTICA Y DE ESQUEMA
# ======================================================

# FUNCIÓN: Validar esquema (Schema Validation)
validar_esquema <- function(datos, esquema_esperado = NULL, fail_fast = FALSE) {

  cat("\n[CALIDAD] Validando esquema...\n")

  columnas_actuales <- names(datos)
  tipos_actuales <- sapply(datos, class)

  resultados <- list(
    columnas_actuales = columnas_actuales,
    n_columnas = length(columnas_actuales),
    tipos = tipos_actuales,
    schema_drift = FALSE,
    columnas_nuevas = character(0),
    columnas_faltantes = character(0),
    errores_tipo = list()
  )

  if (!is.null(esquema_esperado)) {
    # Detectar Schema Drift
    columnas_esperadas <- names(esquema_esperado)

    # Columnas nuevas (no esperadas)
    columnas_nuevas <- setdiff(columnas_actuales, columnas_esperadas)
    if (length(columnas_nuevas) > 0) {
      resultados$columnas_nuevas <- columnas_nuevas
      resultados$schema_drift <- TRUE
      cat("  [ADVERTENCIA] Columnas nuevas detectadas: ", paste(columnas_nuevas, collapse = ", "), "\n", sep =
"")
    }

    # Columnas faltantes
    columnas_faltantes <- setdiff(columnas_esperadas, columnas_actuales)
    if (length(columnas_faltantes) > 0) {
      resultados$columnas_faltantes <- columnas_faltantes
      resultados$schema_drift <- TRUE
      cat("  [ALERTA] Columnas faltantes: ", paste(columnas_faltantes, collapse = ", "), "\n", sep = "")
```

```
      if (fail_fast) {
        stop("[CALIDAD] Schema Drift crítico detectado. Job abortado (Fail Fast).")
      }
    }

    # Validar tipos de datos
    for (col in intersect(columnas_actuales, columnas_esperadas)) {
      tipo_esperado <- esquema_esperado[[col]]
      tipo_actual <- class(datos[[col]])[1]

      if (tipo_actual != tipo_esperado) {
        resultados$errores_tipo[[col]] <- list(
          esperado = tipo_esperado,
          actual = tipo_actual
        )
        cat("  [ADVERTENCIA] Tipo incorrecto en ", col, ": esperado '", tipo_esperado,
            "', actual '", tipo_actual, "'\n", sep = "")
      }
    }
  }

  if (!resultados$schema_drift && length(resultados$errores_tipo) == 0) {
    cat("  [OK] Esquema válido. ", resultados$n_columnas, " columnas.\n", sep = "")
  }

  return(resultados)
}

# ========================================================
# B. INTEGRIDAD REFERENCIAL Y UNICIDAD
# ========================================================

# FUNCIÓN: Validar integridad referencial
validar_integridad_referencial <- function(datos, columna_fk, tabla_referencia, columna_pk) {

  cat("\n[CALIDAD] Validando integridad referencial: ", columna_fk, " -> ", columna_pk, "\n", sep = "")

  if (!columna_fk %in% names(datos)) {
    warning(paste0("Columna FK no encontrada: ", columna_fk))
    return(NULL)
  }

  valores_fk <- unique(na.omit(datos[[columna_fk]]))
  valores_pk <- unique(na.omit(tabla_referencia[[columna_pk]]))

  # Registros huérfanos (sin coincidencia en tabla maestra)
  huerfanos <- setdiff(valores_fk, valores_pk)

  resultado <- list(
    columna_fk = columna_fk,
```

```
    columna_pk = columna_pk,
    total_valores_fk = length(valores_fk),
    total_valores_pk = length(valores_pk),
    huerfanos = huerfanos,
    n_huerfanos = length(huerfanos),
    porcentaje_huerfanos = round((length(huerfanos) / length(valores_fk)) * 100, 2)
  )

  if (length(huerfanos) > 0) {
    cat("  [ADVERTENCIA] ", length(huerfanos), " valores huérfanos (",
        resultado$porcentaje_huerfanos, "%)\n", sep = "")
    if (length(huerfanos) <= 10) {
      cat("    Valores: ", paste(huerfanos, collapse = ", "), "\n", sep = "")
    } else {
      cat("    Primeros 10: ", paste(head(huerfanos, 10), collapse = ", "), "...\n", sep = "")
    }
  } else {
    cat("  [OK] Integridad referencial válida.\n")
  }

  return(resultado)
}

# FUNCIÓN: Detectar duplicados
detectar_duplicados <- function(datos, columnas_clave, enviar_cuarentena = TRUE, execution_id = NULL) {

  cat("\n[CALIDAD] Detectando duplicados en: ", paste(columnas_clave, collapse = ", "), "\n", sep = "")

  columnas_existentes <- intersect(columnas_clave, names(datos))

  if (length(columnas_existentes) == 0) {
    warning("Ninguna columna clave encontrada")
    return(NULL)
  }

  # Contar duplicados (todas las filas que pertenecen a grupos con más de 1 registro)
  duplicados <- datos %>%
    group_by(across(all_of(columnas_existentes))) %>%
    filter(n() > 1) %>%
    ungroup()

  n_duplicados <- nrow(duplicados)

  resultado <- list(
    columnas_clave = columnas_existentes,
    total_registros = nrow(datos),
    registros_duplicados = n_duplicados,
    porcentaje_duplicados = round((n_duplicados / nrow(datos)) * 100, 2),
    datos_duplicados = NULL,
    archivo_cuarentena = NULL
  )
```

```
  if (n_duplicados > 0) {
    cat("  [ALERTA] ", n_duplicados, " registros duplicados (",
        resultado$porcentaje_duplicados, "%)\n", sep = "")

    # Enviar a cuarentena si está habilitado
    if (enviar_cuarentena && n_duplicados > 0) {
      resultado$archivo_cuarentena <- exportar_duplicados_cuarentena(
        duplicados = duplicados,
        columnas_clave = columnas_existentes,
        execution_id = execution_id
      )
    }
  } else {
    cat("  [OK] No hay duplicados.\n")
  }

  return(resultado)
}

# FUNCIÓN: Exportar duplicados a cuarentena
exportar_duplicados_cuarentena <- function(duplicados, columnas_clave, execution_id = NULL) {

  if (nrow(duplicados) == 0) return(NULL)

  # Crear directorio de cuarentena si no existe
  if (!dir.exists(RUTA_CUARENTENA)) {
    dir.create(RUTA_CUARENTENA, recursive = TRUE, showWarnings = FALSE)
  }

  # Generar execution_id si no existe
  if (is.null(execution_id)) {
    execution_id <- paste0("DUP_", format(Sys.time(), "%Y%m%d_%H%M%S"))
  }

  # Agregar metadata de cuarentena
  duplicados_cuarentena <- duplicados %>%
    mutate(
      CUARENTENA_EXECUTION_ID = execution_id,
      CUARENTENA_FECHA = as.character(Sys.time()),
      CUARENTENA_REGLA = "DUPLICADO",
      CUARENTENA_CLAVE = paste(columnas_clave, collapse = " + "),
      CUARENTENA_DESCRIPCION = "Registro duplicado según clave de negocio"
    )

  # Nombre del archivo
  timestamp <- format(Sys.time(), "%Y%m%d_%H%M%S")
  nombre_archivo <- paste0("duplicados_", execution_id, "_", timestamp, ".parquet")
  ruta_archivo <- file.path(RUTA_CUARENTENA, nombre_archivo)

  # Guardar como Parquet
```

```
  arrow::write_parquet(duplicados_cuarentena, ruta_archivo)

  cat("  [CUARENTENA] ", nrow(duplicados), " duplicados exportados\n", sep = "")
  cat("    Archivo: ", nombre_archivo, "\n", sep = "")
  cat("    Clave: ", paste(columnas_clave, collapse = " + "), "\n", sep = "")

  return(list(
    ruta = ruta_archivo,
    nombre = nombre_archivo,
    registros = nrow(duplicados),
    columnas_clave = columnas_clave
  ))
}

# FUNCIÓN: Detectar y exportar cupos perdidos
# Cupos perdidos = ESTADO_CUPO "DISPONIBLE" que nunca fueron utilizados
detectar_cupos_perdidos <- function(datos, execution_id = NULL) {

  cat("\n[CALIDAD] Detectando cupos perdidos (DISPONIBLES no utilizados)...\n")

  # Verificar columnas necesarias
  if (!"ESTADO_CUPO" %in% names(datos)) {
    cat("  [ADVERTENCIA] Columna ESTADO_CUPO no encontrada\n")
    return(NULL)
  }

  # Filtrar cupos disponibles (no utilizados)
  cupos_perdidos <- datos %>%
    filter(toupper(ESTADO_CUPO) == "DISPONIBLE")

  n_perdidos <- nrow(cupos_perdidos)
  n_total <- nrow(datos)
  pct_perdidos <- round((n_perdidos / n_total) * 100, 2)

  resultado <- list(
    total_cupos = n_total,
    cupos_perdidos = n_perdidos,
    porcentaje_perdidos = pct_perdidos,
    archivo_log = NULL
  )

  if (n_perdidos > 0) {
    cat("  [INFO] ", n_perdidos, " cupos perdidos (", pct_perdidos, "% del total)\n", sep = "")

    # Crear directorio de logs si no existe
    ruta_logs <- here::here("LOGS", "CUPOS_PERDIDOS")
    if (!dir.exists(ruta_logs)) {
      dir.create(ruta_logs, recursive = TRUE, showWarnings = FALSE)
    }

    # Generar execution_id si no existe
```

```
    if (is.null(execution_id)) {
      execution_id <- paste0("LOST_", format(Sys.time(), "%Y%m%d_%H%M%S"))
    }

    # Agregar metadata
    cupos_perdidos_log <- cupos_perdidos %>%
      mutate(
        LOG_EXECUTION_ID = execution_id,
        LOG_FECHA = as.character(Sys.time()),
        LOG_TIPO = "CUPO_PERDIDO",
        LOG_DESCRIPCION = "Cupo disponible no utilizado - oportunidad de atención perdida"
      )

    # Nombre del archivo
    timestamp <- format(Sys.time(), "%Y%m%d_%H%M%S")
    nombre_archivo <- paste0("cupos_perdidos_", execution_id, "_", timestamp, ".parquet")
    ruta_archivo <- file.path(ruta_logs, nombre_archivo)

    # Guardar como Parquet
    arrow::write_parquet(cupos_perdidos_log, ruta_archivo)

    resultado$archivo_log <- ruta_archivo

    cat("  [LOG] Cupos perdidos exportados\n")
    cat("    Archivo: ", nombre_archivo, "\n", sep = "")

    # Resumen por establecimiento y tipo de atención
    if (all(c("ESTABLECIMIENTO", "TIPO_ATENCION") %in% names(cupos_perdidos))) {
      resumen <- cupos_perdidos %>%
        group_by(ESTABLECIMIENTO, TIPO_ATENCION) %>%
        summarise(n_perdidos = n(), .groups = "drop") %>%
        arrange(desc(n_perdidos)) %>%
        head(10)

      cat("  [TOP 10] Establecimientos/Tipos con más cupos perdidos:\n")
      for (i in seq_len(nrow(resumen))) {
        cat("    - ", resumen$ESTABLECIMIENTO[i], " | ",
            resumen$TIPO_ATENCION[i], ": ", resumen$n_perdidos[i], "\n", sep = "")
      }
      resultado$resumen_top10 <- resumen
    }

  } else {
    cat("  [OK] No hay cupos perdidos.\n")
  }

  return(resultado)
}

# ======================================================
# C. VALIDACIÓN LÓGICA (Cross-Referencing)
```

```
# ======================================================

# FUNCIÓN: Validar lógica de horarios (HORA_TERMINO > HORA_INICIO)
validar_logica_horarios <- function(datos, col_inicio = "HORA_INICIO", col_termino = "HORA_TERMINO") {

  cat("\n[CALIDAD] Validando lógica de horarios...\n")

  if (!all(c(col_inicio, col_termino) %in% names(datos))) {
    warning("Columnas de hora no encontradas")
    return(NULL)
  }

  # Convertir a tiempo si son caracteres
  datos_temp <- datos %>%
    mutate(
      hora_inicio_num = suppressWarnings(as.integer(substr(!!sym(col_inicio), 1, 2)) * 60 +
                                           as.integer(substr(!!sym(col_inicio), 4, 5))),
      hora_termino_num = suppressWarnings(as.integer(substr(!!sym(col_termino), 1, 2)) * 60 +
                                            as.integer(substr(!!sym(col_termino), 4, 5)))
    )

  # Detectar inconsistencias
  inconsistentes <- datos_temp %>%
    filter(!is.na(hora_inicio_num) & !is.na(hora_termino_num)) %>%
    filter(hora_termino_num <= hora_inicio_num)

  n_inconsistentes <- nrow(inconsistentes)

  resultado <- list(
    regla = "HORA_TERMINO > HORA_INICIO",
    total_evaluados = sum(!is.na(datos_temp$hora_inicio_num) & !is.na(datos_temp$hora_termino_num)),
    inconsistentes = n_inconsistentes,
    porcentaje_error = round((n_inconsistentes / nrow(datos)) * 100, 2),
    datos_inconsistentes = if (n_inconsistentes > 0 && n_inconsistentes <= 100) inconsistentes else NULL
  )

  if (n_inconsistentes > 0) {
    cat("  [ALERTA] ", n_inconsistentes, " registros con HORA_TERMINO <= HORA_INICIO (",
        resultado$porcentaje_error, "%)\n", sep = "")
  } else {
    cat("  [OK] Lógica de horarios válida.\n")
  }

  return(resultado)
}

# FUNCIÓN: Validar lógica de fechas
validar_logica_fechas <- function(datos, col_fecha = "FECHA", rango_min = NULL, rango_max = NULL) {

  cat("\n[CALIDAD] Validando lógica de fechas...\n")
```

```
if (!col_fecha %in% names(datos)) {
  warning(paste0("Columna de fecha no encontrada: ", col_fecha))
  return(NULL)
}

fechas_raw <- datos[[col_fecha]]

# Convertir a Date si es necesario (manejo robusto)
fechas <- tryCatch({
  if (inherits(fechas_raw, "Date")) {
    fechas_raw
  } else if (inherits(fechas_raw, "POSIXct") || inherits(fechas_raw, "POSIXlt")) {
    as.Date(fechas_raw)
  } else {
    # Intentar varios formatos comunes
    parsed <- as.Date(fechas_raw, format = "%Y-%m-%d")
    if (all(is.na(parsed[!is.na(fechas_raw)]))) {
      parsed <- as.Date(fechas_raw, format = "%d-%m-%Y")
    }
    if (all(is.na(parsed[!is.na(fechas_raw)]))) {
      parsed <- as.Date(fechas_raw, format = "%d/%m/%Y")
    }
    parsed
  }
}, error = function(e) {
  warning("No se pudo convertir columna de fecha: ", e$message)
  return(rep(NA, length(fechas_raw)))
})

# Fechas nulas
nulas <- sum(is.na(fechas))

# Fechas futuras (más de 1 día adelante)
hoy <- Sys.Date()
futuras <- sum(fechas > (hoy + 1), na.rm = TRUE)

# Fechas muy antiguas (antes de 2000)
antiguas <- sum(fechas < as.Date("2000-01-01"), na.rm = TRUE)

# Fuera de rango esperado
fuera_rango_min <- 0
fuera_rango_max <- 0

if (!is.null(rango_min)) {
  fuera_rango_min <- sum(fechas < rango_min, na.rm = TRUE)
}
if (!is.null(rango_max)) {
  fuera_rango_max <- sum(fechas > rango_max, na.rm = TRUE)
}

resultado <- list(
```

```
    columna = col_fecha,
    total_registros = length(fechas),
    nulas = nulas,
    futuras = futuras,
    antiguas = antiguas,
    fuera_rango_min = fuera_rango_min,
    fuera_rango_max = fuera_rango_max,
    rango_real = if (sum(!is.na(fechas)) > 0) c(min(fechas, na.rm = TRUE), max(fechas, na.rm = TRUE)) else NULL
  )

  cat("  Rango de fechas: ", as.character(resultado$rango_real[1]), " a ",
      as.character(resultado$rango_real[2]), "\n", sep = "")
  cat("  Nulas: ", nulas, " | Futuras: ", futuras, " | Antiguas (<2000): ", antiguas, "\n", sep = "")

  if (nulas == 0 && futuras == 0 && antiguas == 0) {
    cat("  [OK] Fechas válidas.\n")
  } else {
    cat("  [ADVERTENCIA] Revisar fechas anómalas.\n")
  }

  return(resultado)
}

# FUNCIÓN: Validar completitud de columnas críticas
validar_completitud <- function(datos, columnas_criticas) {

  cat("\n[CALIDAD] Validando completitud de columnas críticas...\n")

  resultados <- list()

  for (col in columnas_criticas) {
    if (col %in% names(datos)) {
      total <- nrow(datos)
      col_data <- datos[[col]]

      # Contar nulos/vacíos según el tipo de columna
      if (inherits(col_data, "Date") || inherits(col_data, "POSIXct") || inherits(col_data, "POSIXlt")) {
        # Para fechas, solo contar NA
        nulos <- sum(is.na(col_data))
      } else if (is.numeric(col_data)) {
        # Para numéricos, solo contar NA
        nulos <- sum(is.na(col_data))
      } else {
        # Para texto, contar NA y cadenas vacías
        nulos <- sum(is.na(col_data) | col_data == "")
      }

      completos <- total - nulos
      pct_completo <- round((completos / total) * 100, 1)

      resultados[[col]] <- list(
```

```
        columna = col,
        total = total,
        completos = completos,
        nulos = nulos,
        porcentaje_completo = pct_completo
      )

      estado <- if (pct_completo >= 99) "[OK]" else if (pct_completo >= 90) "[ADVERTENCIA]" else "[ALERTA]"
      cat("  ", col, ": ", pct_completo, "% completo ", estado, "\n", sep = "")
    } else {
      cat("  ", col, ": [NO EXISTE]\n", sep = "")
      resultados[[col]] <- list(columna = col, existe = FALSE)
    }
  }

  return(resultados)
}

# ========================================================
# FUNCIÓN: Enviar registros a Cuarentena
# ========================================================
enviar_a_cuarentena <- function(
  datos_rechazados,
  execution_id,
  regla_fallada,
  descripcion = NULL
) {

  if (nrow(datos_rechazados) == 0) {
    return(NULL)
  }

  # Crear directorio si no existe
  if (!dir.exists(RUTA_CUARENTENA)) {
    dir.create(RUTA_CUARENTENA, recursive = TRUE, showWarnings = FALSE)
  }

  # Agregar metadata de cuarentena
  datos_cuarentena <- datos_rechazados %>%
    mutate(
      CUARENTENA_EXECUTION_ID = execution_id,
      CUARENTENA_FECHA = as.character(Sys.time()),
      CUARENTENA_REGLA = regla_fallada,
      CUARENTENA_DESCRIPCION = ifelse(is.null(descripcion), NA_character_, descripcion)
    )

  # Nombre del archivo
  timestamp <- format(Sys.time(), "%Y%m%d_%H%M%S")
  nombre_archivo <- paste0("cuarentena_", execution_id, "_", timestamp, ".parquet")
  ruta_archivo <- file.path(RUTA_CUARENTENA, nombre_archivo)
```

```
  # Guardar
  arrow::write_parquet(datos_cuarentena, ruta_archivo)

  cat("[CUARENTENA] ", nrow(datos_rechazados), " registros enviados a cuarentena.\n", sep = "")
  cat("  Regla: ", regla_fallada, "\n", sep = "")
  cat("  Archivo: ", nombre_archivo, "\n", sep = "")

  return(list(
    registros = nrow(datos_rechazados),
    regla = regla_fallada,
    archivo = ruta_archivo
  ))
}

# ========================================================
# FUNCIÓN: Auditoría de Calidad Completa
# ========================================================
auditoria_calidad_completa <- function(
  datos,
  execution_id,
  esquema_esperado = NULL,
  columnas_criticas = c("RUT", "FECHA", "ESTABLECIMIENTO"),
  validar_horarios = TRUE
) {

  cat("\n")
  cat(strrep("=", 60), "\n")
  cat("  AUDITORÍA DE CALIDAD DE DATOS\n")
  cat(strrep("=", 60), "\n")

  resultados <- list()
  advertencias <- character(0)
  errores <- character(0)

  # 1. Validar esquema
  resultados$esquema <- validar_esquema(datos, esquema_esperado)
  if (resultados$esquema$schema_drift) {
    advertencias <- c(advertencias, "Schema Drift detectado")
  }

  # 2. Validar completitud
  resultados$completitud <- validar_completitud(datos, columnas_criticas)

  # 3. Validar lógica de fechas
  if ("FECHA" %in% names(datos)) {
    resultados$fechas <- validar_logica_fechas(datos)
    if (resultados$fechas$nulas > 0) {
      advertencias <- c(advertencias, paste0(resultados$fechas$nulas, " fechas nulas"))
    }
  }
```

```
  # 4. Validar lógica de horarios
  if (validar_horarios && all(c("HORA_INICIO", "HORA_TERMINO") %in% names(datos))) {
    resultados$horarios <- validar_logica_horarios(datos)
    if (resultados$horarios$inconsistentes > 0) {
      advertencias <- c(advertencias, paste0(resultados$horarios$inconsistentes, " horarios inconsistentes"))
    }
  }

  # 5. Detectar duplicados en columnas clave (clave única de negocio)
  # Un cupo es único por: Paciente + Fecha + Hora + Profesional + Establecimiento + TipoCupo + Estados + Motivo
+ Tipo + Detalle + Obs
  # ID_PCTE: Columna creada en Silver (RUT normalizado sin guiones)
  # TIPO_CUPO: Distingue "Cupo Programado" de "Sobrecupo" (slots diferentes en mismo horario)
  # DETALLE_CUPO: Contiene funcionario citador - distingue citas de mismo paciente por diferentes funcionarios
  # OBSERVACIONES: Notas adicionales que distinguen registros similares
  columnas_duplicados <- c("ID_PCTE", "FECHA", "HORA_INICIO", "RUT_PROFESIONAL",
                           "ESTABLECIMIENTO", "TIPO_CUPO", "ESTADO_CUPO", "MOTIVO_CUPO",
                           "ESTADO_CITA", "TIPO_ATENCION", "DETALLE_CUPO", "OBSERVACIONES")
  columnas_disponibles <- intersect(columnas_duplicados, names(datos))

  if (length(columnas_disponibles) >= 3) {
    resultados$duplicados <- detectar_duplicados(
      datos = datos,
      columnas_clave = columnas_disponibles,
      enviar_cuarentena = TRUE,
      execution_id = execution_id
    )
  }

  # 6. Detectar cupos perdidos (DISPONIBLES no utilizados)
  resultados$cupos_perdidos <- detectar_cupos_perdidos(
    datos = datos,
    execution_id = execution_id
  )

  # Resumen
  cat("\n[CALIDAD] Resumen:\n")
  cat("  - Advertencias: ", length(advertencias), "\n", sep = "")
  cat("  - Errores: ", length(errores), "\n", sep = "")

  resultados$advertencias <- advertencias
  resultados$errores <- errores
  resultados$execution_id <- execution_id

  return(resultados)
}
```

Archivo: AUDITORIA\modulos\04_auditoria_seguridad.R

```
# ====================================================
```

```
# MÓDULO 04: AUDITORÍA DE SEGURIDAD Y CUMPLIMIENTO
# ====================================================
# Autor: Anghello Vega Flores
# Cargo: Gestor de Datos
# Institución: Departamento de Salud Municipal de Temuco
# ====================================================
# Objetivo: Registro de accesos, PII tracking, logs inmutables
# Cumplimiento: ISO 27001, Ley 21.663, Ley 21.719 (Chile)
# ====================================================

library(digest)
library(jsonlite)
library(here)

# ====================================================
# CONSTANTES
# ====================================================
RUTA_LOGS_SEGURIDAD <- here::here("LOGS", "AUDITORIA", "seguridad")
RUTA_LOGS_ACCESO <- here::here("LOGS", "AUDITORIA", "acceso")
RUTA_RETENCION <- here::here("LOGS", "AUDITORIA", "retencion")

# Columnas con datos sensibles (PII - Personally Identifiable Information)
COLUMNAS_PII <- c(
  "RUT",
  "RUN",
  "NOMBRE",
  "NOMBRES",
  "APELLIDO",
  "APELLIDOS",
  "DIRECCION",
  "TELEFONO",
  "EMAIL",
  "DIAGNOSTICO",
  "CIE10"
)

# Período de retención (años)
RETENCION_ANIOS <- 6  # Según normativa de salud chilena

# ====================================================
# A. DETECCIÓN Y SEGUIMIENTO DE PII
# ====================================================

# FUNCIÓN: Detectar columnas con datos sensibles
detectar_columnas_pii <- function(datos, columnas_pii_adicionales = NULL) {

  cat("\n[SEGURIDAD] Detectando columnas con datos sensibles (PII)...\n")

  columnas_buscar <- c(COLUMNAS_PII, columnas_pii_adicionales)
  columnas_buscar <- toupper(columnas_buscar)
```

```
  columnas_datos <- toupper(names(datos))

  # Coincidencia exacta
  coincidencias_exactas <- intersect(columnas_datos, columnas_buscar)

  # Coincidencia parcial (contiene)
  coincidencias_parciales <- character(0)
  for (col in columnas_datos) {
    for (patron in columnas_buscar) {
      if (grepl(patron, col) && !col %in% coincidencias_exactas) {
        coincidencias_parciales <- c(coincidencias_parciales, col)
        break
      }
    }
  }

  pii_detectadas <- unique(c(coincidencias_exactas, coincidencias_parciales))

  resultado <- list(
    columnas_pii = pii_detectadas,
    n_columnas_pii = length(pii_detectadas),
    total_columnas = ncol(datos),
    fecha_deteccion = Sys.time()
  )

  if (length(pii_detectadas) > 0) {
    cat("  [ALERTA] ", length(pii_detectadas), " columnas con datos sensibles detectadas:\n", sep = "")
    for (col in pii_detectadas) {
      cat("    - ", col, "\n", sep = "")
    }
  } else {
    cat("  [OK] No se detectaron columnas PII.\n")
  }

  return(resultado)
}

# FUNCIÓN: Generar inventario de datos sensibles
generar_inventario_pii <- function(datos, execution_id) {

  pii_info <- detectar_columnas_pii(datos)

  if (pii_info$n_columnas_pii == 0) {
    return(NULL)
  }

  # Para cada columna PII, generar estadísticas (sin exponer datos)
  inventario <- list()

  for (col in pii_info$columnas_pii) {
    col_original <- names(datos)[toupper(names(datos)) == col][1]
```

```
    if (!is.null(col_original) && col_original %in% names(datos)) {
      valores <- datos[[col_original]]

      inventario[[col_original]] <- list(
        columna = col_original,
        tipo = class(valores)[1],
        total_registros = length(valores),
        registros_con_valor = sum(!is.na(valores) & valores != ""),
        registros_vacios = sum(is.na(valores) | valores == ""),
        valores_unicos = length(unique(na.omit(valores))),
        # NO incluir valores reales por seguridad
        muestra_hash = digest::digest(head(na.omit(valores), 5), algo = "sha256")
      )
    }
  }

  resultado <- list(
    execution_id = execution_id,
    fecha_inventario = as.character(Sys.time()),
    n_columnas_pii = pii_info$n_columnas_pii,
    inventario = inventario
  )

  return(resultado)
}

# ========================================================
# B. LOGS INMUTABLES (Append-Only)
# ========================================================

# FUNCIÓN: Crear log de auditoría inmutable
registrar_log_seguridad <- function(
  execution_id,
  evento,
  descripcion,
  usuario = Sys.info()["user"],
  nivel = c("INFO", "WARNING", "CRITICAL"),
  datos_adicionales = NULL
) {

  nivel <- match.arg(nivel)

  # Crear directorio si no existe
  if (!dir.exists(RUTA_LOGS_SEGURIDAD)) {
    dir.create(RUTA_LOGS_SEGURIDAD, recursive = TRUE, showWarnings = FALSE)
  }

  # Timestamp ISO 8601
  timestamp <- format(Sys.time(), "%Y-%m-%dT%H:%M:%S%z")
```

```
  # Crear entrada de log
  entrada <- list(
    timestamp = timestamp,
    execution_id = execution_id,
    evento = evento,
    nivel = nivel,
    descripcion = descripcion,
    usuario = as.character(usuario),
    hostname = Sys.info()["nodename"],
    datos = datos_adicionales
  )

  # Generar hash de la entrada (para verificar integridad)
  entrada$hash_entrada <- digest::digest(entrada, algo = "sha256")

  # Nombre del archivo de log (por año-mes para particionado)
  año_mes <- format(Sys.Date(), "%Y-%m")
  archivo_log <- file.path(RUTA_LOGS_SEGURIDAD, paste0("audit_log_", año_mes, ".jsonl"))

  # Append al archivo (inmutable - solo agregar)
  linea_json <- jsonlite::toJSON(entrada, auto_unbox = TRUE)
  cat(paste0(linea_json, "\n"), file = archivo_log, append = TRUE)

  if (nivel == "CRITICAL") {
    cat("[SEGURIDAD] [CRITICAL] ", descripcion, "\n", sep = "")
  }

  return(invisible(entrada))
}

# FUNCIÓN: Registrar acceso a datos
registrar_acceso <- function(
  execution_id,
  operacion = c("READ", "WRITE", "DELETE", "TRANSFORM"),
  recurso,
  n_registros = NULL,
  columnas_accedidas = NULL,
  contiene_pii = FALSE
) {

  operacion <- match.arg(operacion)

  # Crear directorio si no existe
  if (!dir.exists(RUTA_LOGS_ACCESO)) {
    dir.create(RUTA_LOGS_ACCESO, recursive = TRUE, showWarnings = FALSE)
  }

  entrada <- list(
    timestamp = format(Sys.time(), "%Y-%m-%dT%H:%M:%S%z"),
    execution_id = execution_id,
    operacion = operacion,
```

```
    recurso = recurso,
    n_registros = n_registros,
    columnas = if (!is.null(columnas_accedidas)) paste(columnas_accedidas, collapse = ",") else NULL,
    contiene_pii = contiene_pii,
    usuario = as.character(Sys.info()["user"]),
    proceso = "R_ETL_Pipeline"
  )

  # Archivo de accesos
  fecha <- format(Sys.Date(), "%Y-%m-%d")
  archivo <- file.path(RUTA_LOGS_ACCESO, paste0("access_log_", fecha, ".jsonl"))

  cat(paste0(jsonlite::toJSON(entrada, auto_unbox = TRUE), "\n"), file = archivo, append = TRUE)

  return(invisible(entrada))
}

# ========================================================
# C. VALIDACIÓN DE CUMPLIMIENTO
# ========================================================

# FUNCIÓN: Verificar cumplimiento regulatorio
verificar_cumplimiento <- function(datos, execution_id) {

  cat("\n[SEGURIDAD] Verificando cumplimiento regulatorio...\n")

  cumplimiento <- list(
    execution_id = execution_id,
    fecha_verificacion = as.character(Sys.time()),
    controles = list()
  )

  # Control 1: Logs de auditoría habilitados
  cumplimiento$controles$logs_auditoria <- list(
    control = "Logs de auditoría habilitados",
    referencia = "ISO 27001 - A.12.4.1",
    estado = dir.exists(RUTA_LOGS_SEGURIDAD),
     observacion = if (dir.exists(RUTA_LOGS_SEGURIDAD)) "Directorio de logs existe" else "CREAR directorio de
logs"
  )

  # Control 2: Detección de PII
  pii_info <- detectar_columnas_pii(datos)
  cumplimiento$controles$deteccion_pii <- list(
    control = "Identificación de datos personales",
    referencia = "Ley 21.719 Art. 3",
    estado = TRUE,
    n_columnas_pii = pii_info$n_columnas_pii,
    columnas = pii_info$columnas_pii
  )
```

```
  # Control 3: Hash de integridad
  cumplimiento$controles$integridad <- list(
    control = "Verificación de integridad (SHA-256)",
    referencia = "ISO 27001 - A.12.2.1",
    estado = TRUE,
    observacion = "SHA-256 implementado en capas Bronze y Silver"
  )

  # Control 4: Política de retención
  cumplimiento$controles$retencion <- list(
    control = paste0("Política de retención (", RETENCION_ANIOS, " años)"),
    referencia = "Ley 21.663 - Ciberseguridad",
    estado = TRUE,
    años = RETENCION_ANIOS
  )

  # Resumen
  controles_ok <- sum(sapply(cumplimiento$controles, function(x) x$estado))
  total_controles <- length(cumplimiento$controles)

  cumplimiento$resumen <- list(
    controles_evaluados = total_controles,
    controles_ok = controles_ok,
    porcentaje_cumplimiento = round((controles_ok / total_controles) * 100, 1)
  )

  cat("  Controles evaluados: ", total_controles, "\n", sep = "")
  cat("  Controles OK: ", controles_ok, "/", total_controles, " (",
      cumplimiento$resumen$porcentaje_cumplimiento, "%)\n", sep = "")

  # Registrar en log de seguridad
  registrar_log_seguridad(
    execution_id = execution_id,
    evento = "COMPLIANCE_CHECK",
     descripcion = paste0("Verificación de cumplimiento: ", controles_ok, "/", total_controles, " controles
OK"),
    nivel = "INFO",
    datos_adicionales = list(controles = names(cumplimiento$controles))
  )

  return(cumplimiento)
}

# ====================================================
# D. GESTIÓN DE RETENCIÓN
# ====================================================

# FUNCIÓN: Verificar política de retención
verificar_retencion <- function(ruta_datos) {

  cat("\n[SEGURIDAD] Verificando política de retención...\n")
```

```
  if (!dir.exists(ruta_datos)) {
    cat("  [ADVERTENCIA] Directorio no existe: ", ruta_datos, "\n", sep = "")
    return(NULL)
  }

  # Listar archivos
  archivos <- list.files(ruta_datos, recursive = TRUE, full.names = TRUE)

  if (length(archivos) == 0) {
    cat("  [INFO] No hay archivos para evaluar.\n")
    return(NULL)
  }

  # Info de archivos
  info_archivos <- file.info(archivos)

  # Calcular antigüedad
  fecha_corte <- Sys.Date() - (RETENCION_ANIOS * 365)

  archivos_antiguos <- rownames(info_archivos)[as.Date(info_archivos$mtime) < fecha_corte]

  resultado <- list(
    total_archivos = length(archivos),
    fecha_corte = as.character(fecha_corte),
    años_retencion = RETENCION_ANIOS,
    archivos_fuera_retencion = length(archivos_antiguos),
            lista_archivos_antiguos   =   if   (length(archivos_antiguos)   <=   20)   archivos_antiguos   else
head(archivos_antiguos, 20)
  )

  if (length(archivos_antiguos) > 0) {
    cat("  [ADVERTENCIA] ", length(archivos_antiguos), " archivos fuera de política de retención.\n", sep = "")
  } else {
    cat("  [OK] Todos los archivos dentro de política de retención.\n")
  }

  return(resultado)
}

# ========================================================
# FUNCIÓN: Auditoría de Seguridad Completa
# ========================================================
auditoria_seguridad_completa <- function(datos, execution_id, ruta_datos = NULL) {

  cat("\n")
  cat(strrep("=", 60), "\n")
  cat("  AUDITORÍA DE SEGURIDAD Y CUMPLIMIENTO\n")
  cat(strrep("=", 60), "\n")

  resultados <- list()
```

```
  # 1. Detectar e inventariar PII
  resultados$pii <- generar_inventario_pii(datos, execution_id)

  # 2. Verificar cumplimiento regulatorio
  resultados$cumplimiento <- verificar_cumplimiento(datos, execution_id)

  # 3. Verificar política de retención
  if (!is.null(ruta_datos)) {
    resultados$retencion <- verificar_retencion(ruta_datos)
  }

  # 4. Registrar acceso
  registrar_acceso(
    execution_id = execution_id,
    operacion = "READ",
    recurso = "dataset_silver",
    n_registros = nrow(datos),
    columnas_accedidas = names(datos),
    contiene_pii = !is.null(resultados$pii)
  )

  # Registrar evento de auditoría
  registrar_log_seguridad(
    execution_id = execution_id,
    evento = "SECURITY_AUDIT_COMPLETE",
    descripcion = "Auditoría de seguridad completada",
    nivel = "INFO",
    datos_adicionales = list(
      n_registros = nrow(datos),
      n_columnas_pii = if (!is.null(resultados$pii)) resultados$pii$n_columnas_pii else 0
    )
  )

  resultados$execution_id <- execution_id
  resultados$fecha <- as.character(Sys.time())

  cat("\n[SEGURIDAD] Auditoría completada.\n")

  return(resultados)
}
```

Archivo: AUDITORIA\modulos\05_auditoria_balance_masas.R

```
# ======================================================
# MÓDULO 05: AUDITORÍA DE BALANCE DE MASAS
# Excel Entrada ? Silver Final
# ======================================================
# Verifica que TODOS los registros de los Excel originales
# lleguen al producto final sin pérdida de datos.
```

```
# ======================================================

library(readxl)
library(arrow)
library(dplyr)
library(tidyr)
library(here)
library(jsonlite)

# --- Función Principal ---
auditar_balance_masas <- function(
    ruta_entrada = here::here("DATOS", "ENTRADA"),
    ruta_silver = here::here("DATOS", "SALIDA"),
    exportar_reporte = TRUE
) {

  cat("\n")
  cat(strrep("=", 70), "\n")

  cat("   AUDITORÍA DE BALANCE DE MASAS\n")
  cat("   Excel Entrada ? Silver Final\n")
  cat(strrep("=", 70), "\n\n")

  timestamp_inicio <- Sys.time()

  # --- 1. Escanear Excel de Entrada ---
  cat("[1/4] Escaneando archivos Excel de entrada...\n")

  archivos_excel <- list.files(
    ruta_entrada,
    pattern = "\\.xlsx?$",
    recursive = TRUE,
    full.names = TRUE
  )

  if (length(archivos_excel) == 0) {
    stop("No se encontraron archivos Excel en: ", ruta_entrada)
  }

  cat("      Archivos encontrados: ", length(archivos_excel), "\n")

  # --- 2. Contar Registros por Archivo ---
  # IMPORTANTE: Usar skip=8 igual que en Bronze para contar solo datos reales
  cat("[2/4] Contando registros por archivo Excel (skip=8, igual que Bronze)...\n")

  conteo_entrada <- lapply(archivos_excel, function(archivo) {
    tryCatch({
      # Contar registros en la primera hoja (igual que Bronze)
      # Bronze usa skip=8 para saltar encabezados del reporte
      registros <- tryCatch({
        df <- suppressMessages(readxl::read_excel(
```

```
        archivo,
        sheet = 1,        # Solo primera hoja (igual que Bronze)
        skip = 8,         # CRÍTICO: Igual que Bronze
        col_types = "text"
      ))
      nrow(df)
    }, error = function(e) 0)

    # Contar hojas disponibles (info)
    n_hojas <- length(readxl::excel_sheets(archivo))

    # Extraer año y mes del path
    path_parts <- strsplit(archivo, "[/\\\\]")[[1]]
    carpeta_anio <- grep("^CXC_", path_parts, value = TRUE)[1]
    anio <- if (!is.na(carpeta_anio)) gsub("CXC_", "", carpeta_anio) else "DESCONOCIDO"

    # Buscar carpeta de mes
    meses <- c("ENERO", "FEBRERO", "MARZO", "ABRIL", "MAYO", "JUNIO",
               "JULIO", "AGOSTO", "SEPTIEMBRE", "OCTUBRE", "NOVIEMBRE", "DICIEMBRE")
    mes_encontrado <- intersect(toupper(path_parts), meses)
    mes <- if (length(mes_encontrado) > 0) mes_encontrado[1] else "DESCONOCIDO"

    data.frame(
      archivo = basename(archivo),
      ruta_completa = archivo,
      anio = anio,
      mes = mes,
      hojas = n_hojas,
      registros = registros,
      estado = "OK",
      stringsAsFactors = FALSE
    )
  }, error = function(e) {
    data.frame(
      archivo = basename(archivo),
      ruta_completa = archivo,
      anio = NA,
      mes = NA,
      hojas = 0,
      registros = 0,
      estado = paste("ERROR:", e$message),
      stringsAsFactors = FALSE
    )
  })
})

df_entrada <- do.call(rbind, conteo_entrada)

# Resumen por año
resumen_entrada_anio <- df_entrada |>
  filter(estado == "OK") |>
```

```
  group_by(anio) |>
  summarise(
    archivos = n(),
    registros = sum(registros),
    .groups = "drop"
  )

cat("      Registros totales en Excel: ", format(sum(df_entrada$registros), big.mark = ","), "\n")
cat("      Archivos con error: ", sum(df_entrada$estado != "OK"), "\n\n")

# --- 3. Contar Registros en Silver ---
cat("[3/4] Contando registros en Silver...\n")

if (!dir.exists(ruta_silver)) {
  stop("No existe el directorio Silver: ", ruta_silver)
}

silver_ds <- arrow::open_dataset(ruta_silver)

# Conteo total
silver_total <- nrow(silver_ds)

# Conteo por año (usar columna de partición)
resumen_silver_anio <- silver_ds |>
  count(año) |>
  collect() |>
  rename(anio = año, registros_silver = n) |>
  mutate(anio = as.character(anio))

cat("      Registros totales en Silver: ", format(silver_total, big.mark = ","), "\n\n")

# --- 4. Calcular Balance ---
cat("[4/4] Calculando balance de masas...\n\n")

# Unir por año
balance_anio <- resumen_entrada_anio |>
  full_join(resumen_silver_anio, by = "anio") |>
  mutate(
    registros = replace_na(registros, 0),
    registros_silver = replace_na(registros_silver, 0),
    diferencia = registros - registros_silver,
    pct_diferencia = round((diferencia / registros) * 100, 4),
    estado = case_when(
      diferencia == 0 ~ "EXACTO",
      abs(pct_diferencia) <= 0.01 ~ "ACEPTABLE",
      abs(pct_diferencia) <= 1 ~ "REVISAR",
      TRUE ~ "ALERTA"
    )
  ) |>
  arrange(anio)
```

```
# Totales
total_entrada <- sum(df_entrada$registros[df_entrada$estado == "OK"])
total_silver <- silver_total
diferencia_total <- total_entrada - total_silver
pct_diferencia_total <- round((diferencia_total / total_entrada) * 100, 4)

# --- Imprimir Resultados ---
cat(strrep("-", 70), "\n")
cat("   BALANCE POR AÑO\n")
cat(strrep("-", 70), "\n\n")

for (i in seq_len(nrow(balance_anio))) {
  row <- balance_anio[i, ]
  cat(sprintf("   Año %s:\n", row$anio))
  cat(sprintf("     Excel:  %s registros\n", format(row$registros, big.mark = ",")))
  cat(sprintf("     Silver: %s registros\n", format(row$registros_silver, big.mark = ",")))
  cat(sprintf("     Diff:   %s (%s%%)\n", row$diferencia, row$pct_diferencia))
  cat(sprintf("     Estado: [%s]\n\n", row$estado))
}

cat(strrep("=", 70), "\n")
cat("   BALANCE TOTAL\n")
cat(strrep("=", 70), "\n\n")
cat(sprintf("   Excel Entrada:    %s registros\n", format(total_entrada, big.mark = ",")))
cat(sprintf("   Silver Final:     %s registros\n", format(total_silver, big.mark = ",")))
cat(sprintf("   Diferencia:       %s (%s%%)\n", diferencia_total, pct_diferencia_total))

if (diferencia_total == 0) {
  cat("\n   [OK] BALANCE EXACTO - Sin pérdida de datos\n")
} else if (abs(pct_diferencia_total) <= 0.01) {
  cat("\n   [OK] BALANCE ACEPTABLE - Diferencia mínima\n")
} else if (abs(pct_diferencia_total) <= 1) {
  cat("\n   [REVISAR] Diferencia menor al 1%\n")
} else {
  cat("\n   [ALERTA] Diferencia significativa - Investigar\n")
}

cat("\n")

# --- Exportar Reporte ---
timestamp_fin <- Sys.time()
duracion <- round(as.numeric(difftime(timestamp_fin, timestamp_inicio, units = "secs")), 1)

resultado <- list(
  metadata = list(
    tipo = "BALANCE_DE_MASAS",
    version = "1.0.0",
    timestamp = format(timestamp_inicio, "%Y-%m-%dT%H:%M:%S"),
    duracion_segundos = duracion
  ),
  entrada = list(
```

```
    ruta = ruta_entrada,
    archivos_total = nrow(df_entrada),
    archivos_ok = sum(df_entrada$estado == "OK"),
    archivos_error = sum(df_entrada$estado != "OK"),
    registros_total = total_entrada
  ),
  silver = list(
    ruta = ruta_silver,
    registros_total = total_silver,
    columnas = length(names(silver_ds)),
    particiones = length(list.dirs(ruta_silver, recursive = TRUE)) - 1
  ),
  balance = list(
    diferencia_absoluta = diferencia_total,
    diferencia_porcentaje = pct_diferencia_total,
    estado = if (diferencia_total == 0) "EXACTO"
             else if (abs(pct_diferencia_total) <= 0.01) "ACEPTABLE"
             else if (abs(pct_diferencia_total) <= 1) "REVISAR"
             else "ALERTA"
  ),
  detalle_por_anio = balance_anio,
  detalle_archivos = df_entrada
)

if (exportar_reporte) {
  reporte_dir <- here::here("LOGS", "AUDITORIA", "balance_masas")
  if (!dir.exists(reporte_dir)) {
    dir.create(reporte_dir, recursive = TRUE, showWarnings = FALSE)
  }

  # JSON
  reporte_json <- file.path(
    reporte_dir,
    paste0("balance_masas_", format(timestamp_inicio, "%Y%m%d_%H%M%S"), ".json")
  )
  jsonlite::write_json(resultado, reporte_json, pretty = TRUE, auto_unbox = TRUE)
  cat("[REPORTE] ", reporte_json, "\n")

  # CSV detalle archivos
  reporte_csv <- file.path(
    reporte_dir,
    paste0("detalle_archivos_", format(timestamp_inicio, "%Y%m%d_%H%M%S"), ".csv")
  )
  write.csv(df_entrada, reporte_csv, row.names = FALSE)
  cat("[DETALLE] ", reporte_csv, "\n")
}

cat("\n", strrep("=", 70), "\n", sep = "")
cat("   AUDITORÍA DE BALANCE COMPLETADA\n")
cat(strrep("=", 70), "\n\n")
```

```
  invisible(resultado)
}

# --- Función: Comparar Archivo Individual ---
comparar_archivo_excel <- function(archivo_excel, datos_silver) {
  # Útil para debugging cuando hay diferencias

  cat("\nComparando: ", basename(archivo_excel), "\n")

  # Leer Excel
  hojas <- readxl::excel_sheets(archivo_excel)
  df_excel <- lapply(hojas, function(h) readxl::read_excel(archivo_excel, sheet = h, col_types = "text"))
  df_excel <- do.call(rbind, df_excel)

  cat("  Registros Excel: ", nrow(df_excel), "\n")
  cat("  Columnas Excel:  ", ncol(df_excel), "\n")
  cat("  Columnas: ", paste(head(names(df_excel), 5), collapse = ", "), "...\n")

  invisible(df_excel)
}

# --- Ejecución Directa ---
if (interactive() && !exists("AUDITORIA_CARGADA")) {
  cat("\n[INFO] Módulo 05: Balance de Masas cargado\n")
  cat("       Usar: resultado <- auditar_balance_masas()\n\n")
}
```

Archivo: DOCS\LOGICA_ESTADOS.md

```
# Lógica de Negocio: Estados de un Cupo/Cita

## Resumen Ejecutivo

Este documento describe el ciclo de vida de un **cupo de atención** en el sistema de salud, desde su creación hasta su resolución final.

---

## 1. Entidades Principales

### 1.1 Cupo
Un **cupo** representa una unidad de tiempo disponible para atención médica, definida por:
- **Profesional** (RUT_PROFESIONAL)
- **Establecimiento** (ESTABLECIMIENTO)
- **Fecha** (FECHA)
- **Hora Inicio/Término** (HORA_INICIO, HORA_TERMINO)
- **Tipo de Cupo** (TIPO_CUPO)

### 1.2 Cita
Una **cita** es la asignación de un **paciente** (ID_PCTE) a un **cupo** específico.
```

---

2. Tipos de Cupo

Tipo	Descripción
`Cupo Programado`	Slot de agenda regular, planificado con anticipación
`Sobrecupo`	Slot adicional fuera de la agenda regular (emergencias, demanda espontánea)

> **Nota**: Un mismo horario puede tener AMBOS tipos (1 programado + 1 sobrecupo).

---

3. Estados del Cupo (ESTADO_CUPO)

```
??????????????????????????????????????????????????????????????????????
?                    CICLO DE VIDA DEL CUPO                          ?
??????????????????????????????????????????????????????????????????????

    ???????????????????
    ?  DISPONIBLE  ?  ? Estado inicial: cupo sin paciente asignado
    ???????????????????
           ?
           ? Paciente solicita cita
           ?
    ???????????????????
    ?    CITADO    ?  ? Paciente asignado al cupo
    ???????????????????
           ?
           ? Se bloquea el cupo
           ?
    ???????????????????
    ?   BLOQUEADO  ?  ? Cupo no disponible (razón administrativa)
    ???????????????????
```

3.1 Detalle de Estados

Estado	Código	Descripción	ID_PCTE
DISPONIBLE	`DISPONIBLE`	Cupo sin paciente, listo para ser asignado	`NULL`
CITADO	`CITADO`	Paciente asignado, esperando atención	Requerido
BLOQUEADO	`BLOQUEADO`	Cupo inhabilitado por motivo administrativo	Opcional

3.2 Transiciones Válidas

```
DISPONIBLE ? CITADO      (Agendamiento)
DISPONIBLE ? BLOQUEADO   (Bloqueo administrativo)
```

```
CITADO     ? DISPONIBLE  (Cancelación de cita)
CITADO     ? BLOQUEADO   (Bloqueo post-cita)
BLOQUEADO  ? DISPONIBLE  (Desbloqueo)
```

---

4. Estados de la Cita (ESTADO_CITA)

Una vez que el cupo está `CITADO`, la cita tiene su propio ciclo:

```
????????????????????????????????????????????????????????????????????????
?                    CICLO DE VIDA DE LA CITA                          ?
????????????????????????????????????????????????????????????????????????

    ???????????????????
    ?   Agendado   ?  ? Cita programada, paciente aún no llega
    ???????????????????
           ?
           ? Paciente se presenta
           ?
    ???????????????????
    ?   Iniciado   ?  ? Atención en curso
    ???????????????????
           ?
           ? Atención finaliza
           ?
    ???????????????????
    ?  Completado  ?  ? Atención realizada exitosamente
    ???????????????????
```

4.1 Detalle de Estados de Cita

Estado	Descripción	Cupo Utilizado
Agendado	Cita programada, paciente no ha llegado	No
Iniciado	Paciente en atención	Parcial
Completado	Atención finalizada exitosamente	Sí
NULL	Sin cita (cupo DISPONIBLE o BLOQUEADO)	No aplica

4.2 Transiciones Válidas de Cita

```
NULL       ? Agendado    (Se agenda cita)
Agendado   ? Iniciado    (Paciente llega, comienza atención)
Agendado   ? NULL        (Cita cancelada)
Iniciado   ? Completado  (Atención finaliza)
Iniciado   ? Agendado    (Atención interrumpida, reagendar)
```

---

5. Matriz de Combinaciones Estado_Cupo + Estado_Cita

ESTADO_CUPO	ESTADO_CITA	Significado	Cupo Perdido?
DISPONIBLE	NULL	Cupo libre, sin paciente	**SÍ** (si fecha pasó)
CITADO	Agendado	Paciente citado, no atendido aún	Potencial
CITADO	Iniciado	Paciente en atención	No
CITADO	Completado	Atención exitosa	No
BLOQUEADO	NULL	Cupo inhabilitado	Depende del motivo

---

6. Cupos Perdidos (Oportunidades de Atención Perdidas)

Un cupo se considera **perdido** cuando:

```sql
ESTADO_CUPO = 'DISPONIBLE'
AND FECHA < CURRENT_DATE
AND ID_PCTE IS NULL
```

6.1 Causas Comunes de Cupos Perdidos

Causa	Descripción
Baja demanda	No hubo pacientes para ese horario
Inasistencia	Paciente no agendó la cita disponible
Error sistema	Cupo no visible en plataforma de agendamiento
Horario inconveniente	Horarios que pacientes evitan

6.2 Impacto

- **Recurso humano**: Profesional disponible sin pacientes
- **Infraestructura**: Box de atención no utilizado
- **Lista de espera**: Pacientes esperando mientras hay cupos vacíos

---

7. Clave Única de un Registro (Deduplicación)

Un registro es **único** cuando se identifica por:

```
ID_PCTE + FECHA + HORA_INICIO + RUT_PROFESIONAL + ESTABLECIMIENTO +
TIPO_CUPO + ESTADO_CUPO + MOTIVO_CUPO + ESTADO_CITA + TIPO_ATENCION
```

7.1 Justificación de Cada Campo

Campo	Razón de Inclusión
`ID_PCTE`	Mismo paciente puede tener múltiples citas
`FECHA`	Distingue citas en diferentes días
`HORA_INICIO`	Distingue citas en mismo día
`RUT_PROFESIONAL`	Mismo paciente puede ver diferentes profesionales
`ESTABLECIMIENTO`	Mismo profesional puede atender en múltiples centros
`TIPO_CUPO`	"Cupo Programado" ? "Sobrecupo" en mismo horario
`ESTADO_CUPO`	Mismo cupo puede cambiar de estado
`MOTIVO_CUPO`	Distingue razones de estado
`ESTADO_CITA`	Refleja progreso de la atención
`TIPO_ATENCION`	Mismo paciente puede tener citas de diferente tipo

---

8. Diagrama de Estado Completo

```
                              ????????????????????????????????????????
                              ?         CREACIÓN DE CUPO              ?
                              ?   (Agenda del profesional)            ?
                              ????????????????????????????????????????
                                            ?
                                            ?
                    ??????????????????????????????????????????????????????
                    ?                 DISPONIBLE                          ?
                    ?  ? ID_PCTE = NULL                                   ?
                    ?  ? ESTADO_CITA = NULL                               ?
                    ?  ? Cupo listo para ser asignado                     ?
                    ??????????????????????????????????????????????????????
                             ?                    ?
               Paciente      ?                    ?  Motivo
               solicita cita ?                    ?  administrativo
                             ?                    ?
      ????????????????????????????????????    ????????????????????????????????
      ?           CITADO             ?    ?        BLOQUEADO         ?
      ?  ? ID_PCTE = RUT paciente    ?    ?  ? MOTIVO_BLOQUEO        ?
      ?  ? Esperando atención        ?    ?  ? Cupo inhabilitado     ?
      ????????????????????????????????????    ????????????????????????????????
                     ?
                     ?
      ????????????????????????????????????????????????????????????????????????
      ?                    ESTADO_CITA                                       ?
      ????????????????????????????????????????????????????????????????????????
      ?                                                                      ?
      ?   Agendado ??????? Iniciado ??????? Completado                      ?
      ?       ?                                    ?                    ?
      ?       ?                                    ?                    ?
      ?       ?                                    ?                    ?
```

```
     ?   (Cancelar)                          CUPO UTILIZADO          ?
     ?                                       EXITOSAMENTE            ?
     ??????????????????????????????????????????????????????????????????????
```

---

9. Reglas de Negocio Críticas

9.1 Validaciones

Regla	Descripción	Severidad
R001	`HORA_TERMINO > HORA_INICIO`	Error
R002	Si `ESTADO_CUPO = 'CITADO'` entonces `ID_PCTE NOT NULL`	Error
R003	Si `ESTADO_CITA = 'Completado'` entonces `CUPOS_UTILIZADOS >= 1`	Advertencia
R004	`FECHA_BLOQUEO` solo si `ESTADO_CUPO = 'BLOQUEADO'`	Advertencia

9.2 Métricas Derivadas

Métrica	Cálculo
Tasa de Utilización	`Cupos CITADO / Total Cupos * 100`
Tasa de Pérdida	`Cupos DISPONIBLE (fecha < hoy) / Total Cupos * 100`
Tasa de Completitud	`Citas Completado / Citas Agendado * 100`

---

10. Glosario

Término	Definición
Cupo	Unidad de tiempo disponible para atención
Sobrecupo	Cupo adicional fuera de agenda regular
Cita	Asignación de paciente a un cupo
Cupo Perdido	Cupo DISPONIBLE que pasó sin ser utilizado
Rendimiento	Duración planificada del cupo (ej: 0.:30:0. = 30 min)
Horario Extendido	Atención fuera de horario laboral estándar

Historial de Cambios

Fecha	Versión	Autor	Descripción
2026-02-08	1.0	Anghello Vega Flores	Creación inicial

> **Departamento de Salud Municipal de Temuco**
> Gestión de Datos - 2026

Archivo: LOGS\AUDITORIA\balance_masas\balance_masas_20260208_032744.json


```
{
  "metadata": {
    "tipo": "BALANCE_DE_MASAS",
    "version": "1.0.0",
    "timestamp": "2026-02-08T03:27:44",
    "duracion_segundos": 11.8
  },
  "entrada": {
    "ruta": "C:/Users/Anghello/Documents/GitHub/cupos_por_citas_DSM_TCO-main/DATOS/ENTRADA",
    "archivos_total": 4,
    "archivos_ok": 4,
    "archivos_error": 0,
    "registros_total": 451251
  },
  "silver": {
    "ruta": "C:/Users/Anghello/Documents/GitHub/cupos_por_citas_DSM_TCO-main/DATOS/SALIDA",
    "registros_total": 451251,
    "columnas": 47,
    "particiones": 37
  },
  "balance": {
    "diferencia_absoluta": 0,
    "diferencia_porcentaje": 0,
    "estado": "EXACTO"
  },
  "detalle_por_anio": [
    {
      "anio": "2017",
      "archivos": 1,
      "registros": 21288,
      "registros_silver": 21288,
      "diferencia": 0,
      "pct_diferencia": 0,
      "estado": "EXACTO"
    },
    {
      "anio": "2023",
      "archivos": 1,
      "registros": 370592,
      "registros_silver": 370592,
      "diferencia": 0,
      "pct_diferencia": 0,
      "estado": "EXACTO"
    },
    {
      "anio": "2025",
      "archivos": 1,
      "registros": 56209,
```

```
      "registros_silver": 56209,
      "diferencia": 0,
      "pct_diferencia": 0,
      "estado": "EXACTO"
    },
    {
      "anio": "2026",
      "archivos": 1,
      "registros": 3162,
      "registros_silver": 3162,
      "diferencia": 0,
      "pct_diferencia": 0,
      "estado": "EXACTO"
    }
  ],
  "detalle_archivos": [
    {
      "archivo": "2017_ES_Cantidad_de_Cupos_por_Citas.xlsx",
                                                                                "ruta_completa":
"C:/Users/Anghello/Documents/GitHub/cupos_por_citas_DSM_TCO-main/DATOS/ENTRADA/CXC_2017/2017_ES_Cantidad_de_Cup
os_por_Citas.xlsx",
      "anio": "2017",
      "mes": "DESCONOCIDO",
      "hojas": 3,
      "registros": 21288,
      "estado": "OK"
    },
    {
      "archivo": "2023_AMA_Cantidad_de_Cupos_por_Citas.xlsx",
                                                                                "ruta_completa":
"C:/Users/Anghello/Documents/GitHub/cupos_por_citas_DSM_TCO-main/DATOS/ENTRADA/CXC_2023/2023_AMA_Cantidad_de_Cu
pos_por_Citas.xlsx",
      "anio": "2023",
      "mes": "DESCONOCIDO",
      "hojas": 3,
      "registros": 370592,
      "estado": "OK"
    },
    {
      "archivo": "2025_ARQ_Cantidad_de_Cupos_por_Citas.xlsx",
                                                                                "ruta_completa":
"C:/Users/Anghello/Documents/GitHub/cupos_por_citas_DSM_TCO-main/DATOS/ENTRADA/CXC_2025/2025_ARQ_Cantidad_de_Cu
pos_por_Citas.xlsx",
      "anio": "2025",
      "mes": "DESCONOCIDO",
      "hojas": 3,
      "registros": 56209,
      "estado": "OK"
    },
    {
      "archivo": "2026_ENE_CCR_Cantidad_de_Cupos_por_Citas.xlsx",
```

```
                                                                "ruta_completa":
"C:/Users/Anghello/Documents/GitHub/cupos_por_citas_DSM_TCO-main/DATOS/ENTRADA/CXC_2026/ENERO/2026_ENE_CCR_Cant
idad_de_Cupos_por_Citas.xlsx",
      "anio": "2026",
      "mes": "ENERO",
      "hojas": 3,
      "registros": 3162,
      "estado": "OK"
    }
  ]
}
```


Archivo: LOGS\AUDITORIA\JSON\audit_EXE_20260208_032738_e7119bd8.json


```
{
  "execution_id": "EXE_20260208_032738_e7119bd8",
  "nombre_proceso": "AUDITORIA_COMPLETA",
  "timestamp_inicio": "2026-02-08 03:27:38.090728",
  "timestamp_fin": "2026-02-08 03:27:44.146425",
  "duracion_segundos": 6.0557,
  "status": "EXITO",
  "balance_masas": {
    "entrada": null,
    "salida": 451251,
    "rechazados": null,
    "tasa_rechazo": null
  },
  "integridad": {
    "hash_entrada": null,
    "hash_salida": null
  },
  "ambiente": {
    "usuario": "Anghello",
    "hostname": "DESKTOP-CVO232P",
    "r_version": "4.5.2",
    "os": "Windows"
  },
  "errores": {},
  "advertencias": {}
}
```


Archivo: LOGS\AUDITORIA\reportes\reporte_auditoria_EXE_20260208_032738_e7119bd8_20260208


```
{
  "metadata": {
    "titulo": "Reporte de Auditoría Empresarial",
    "institucion": "Departamento de Salud Municipal de Temuco",
    "execution_id": "EXE_20260208_032738_e7119bd8",
    "fecha_generacion": "2026-02-08 03:27:44.17896",
```

```
    "version_framework": "1.0.0"
  },
  "resumen": {
    "estado": "COMPLETADO",
    "duracion_segundos": 6.0327,
    "modulos_ejecutados": ["operacional", "integridad", "calidad", "seguridad"]
  },
  "detalle": {
    "operacional": {
      "archivos_registrados": 4,
      "timestamp": "2026-02-08 03:27:38.099087"
    },
    "integridad": {
      "execution_id": "EXE_20260208_032738_e7119bd8",
      "archivos_procesados": 4,
      "archivos": {
                                            "NOMBRE_ARCHIVO":      ["2017_ES_Cantidad_de_Cupos_por_Citas.xlsx",
"2023_AMA_Cantidad_de_Cupos_por_Citas.xlsx",                        "2025_ARQ_Cantidad_de_Cupos_por_Citas.xlsx",
"2026_ENE_CCR_Cantidad_de_Cupos_por_Citas.xlsx"],
        "TIPO": ["ENTRADA", "ENTRADA", "ENTRADA", "ENTRADA"],
                          "HASH_SHA256":    ["b266e6d659e6551cb206f2ad5884df265f7c07669d2989456dbd550c2efe1e59",
"07ff04660daafbaba79034c6523b0a8f4ee4b4639792b2d391a1525b37531860",
"9bdbe3fec2c14a2dc96ebeb4762dc454a5c164c92fdabc6eddd71f3273458d85",
"0f10f48f1f69914795d748a0db8791b3a8a78b238c108e6d87f5faae23ece398"],
        "TAMANO_BYTES": [2846197, 51088449, 7333690, 426213]
      },
      "integridad_verificada": true
    },
    "calidad": {
      "esquema": {
                "columnas_actuales":  ["NUMERO_TIPO_IDENTIFICACION",  "RUT",  "NOMBRE",  "NOMBRE_SOCIAL",  "SS",
"ESTABLECIMIENTO",  "FECHA",  "TIPO_ATENCION",  "ESPECIALIDAD",  "INSTRUMENTO",  "TIPO_CUPO",  "HORA_INICIO",
"HORA_TERMINO",   "ESTADO_CUPO",   "HABILITADO",   "TIPO_DE_AGENDAMIENTO",   "TELECONSULTA",   "MOTIVO_CUPO",
"DETALLE_CUPO",  "OBSERVACIONES",  "EDAD_ANO",  "EDAD_MES",  "EDAD_DIA",  "FECHA_DE_NACIMIENTO",  "TELEFONOS",
"SECTOR",        "RUT_PROFESIONAL",        "PROFESIONAL",        "FUNCIONARIO_CITADOR",        "MOTIVO_BLOQUEO",
"FUNCIONARIO_REALIZA_BLOQUEO",     "FECHA_BLOQUEO",     "HORA_BLOQUEO",     "ESTADO_CITA",     "RENDIMIENTO",
"CUPOS_UTILIZADOS", "ID_PCTE", "CENTRO_SALUD", "TIPO_CENTRO", "ISO_DOW", "GRUPO_ETARIO", "HORARIO_EXTENDIDO",
"ETL_ARCHIVO_ORIGEN", "ETL_FECHA_EXTRACCION", "ETL_HASH_ARCHIVO", "año", "mes"],
        "n_columnas": 47,
        "tipos": {
          "NUMERO_TIPO_IDENTIFICACION": "character",
          "RUT": "character",
          "NOMBRE": "character",
          "NOMBRE_SOCIAL": "character",
          "SS": "character",
          "ESTABLECIMIENTO": "character",
          "FECHA": ["POSIXct", "POSIXt"],
          "TIPO_ATENCION": "character",
          "ESPECIALIDAD": "character",
          "INSTRUMENTO": "character",
          "TIPO_CUPO": "character",
```

```
    "HORA_INICIO": "character",
    "HORA_TERMINO": "character",
    "ESTADO_CUPO": "character",
    "HABILITADO": "character",
    "TIPO_DE_AGENDAMIENTO": "character",
    "TELECONSULTA": "character",
    "MOTIVO_CUPO": "character",
    "DETALLE_CUPO": "character",
    "OBSERVACIONES": "character",
    "EDAD_ANO": "integer",
    "EDAD_MES": "integer",
    "EDAD_DIA": "integer",
    "FECHA_DE_NACIMIENTO": ["POSIXct", "POSIXt"],
    "TELEFONOS": "character",
    "SECTOR": "character",
    "RUT_PROFESIONAL": "character",
    "PROFESIONAL": "character",
    "FUNCIONARIO_CITADOR": "character",
    "MOTIVO_BLOQUEO": "character",
    "FUNCIONARIO_REALIZA_BLOQUEO": "character",
    "FECHA_BLOQUEO": ["POSIXct", "POSIXt"],
    "HORA_BLOQUEO": "character",
    "ESTADO_CITA": "character",
    "RENDIMIENTO": "character",
    "CUPOS_UTILIZADOS": "integer",
    "ID_PCTE": "character",
    "CENTRO_SALUD": "character",
    "TIPO_CENTRO": "character",
    "ISO_DOW": "numeric",
    "GRUPO_ETARIO": "character",
    "HORARIO_EXTENDIDO": "character",
    "ETL_ARCHIVO_ORIGEN": "character",
    "ETL_FECHA_EXTRACCION": "character",
    "ETL_HASH_ARCHIVO": "character",
    "año": "integer",
    "mes": "integer"
  },
  "schema_drift": false,
  "columnas_nuevas": [],
  "columnas_faltantes": [],
  "errores_tipo": []
},
"completitud": {
  "RUT": {
    "columna": "RUT",
    "total": 451251,
    "completos": 168131,
    "nulos": 283120,
    "porcentaje_completo": 37.3
  },
  "FECHA": {
```

```
          "columna": "FECHA",
          "total": 451251,
          "completos": 451251,
          "nulos": 0,
          "porcentaje_completo": 100
        },
        "ESTABLECIMIENTO": {
          "columna": "ESTABLECIMIENTO",
          "total": 451251,
          "completos": 451251,
          "nulos": 0,
          "porcentaje_completo": 100
        }
      },
      "fechas": {
        "columna": "FECHA",
        "total_registros": 451251,
        "nulas": 0,
        "futuras": 0,
        "antiguas": 0,
        "fuera_rango_min": 0,
        "fuera_rango_max": 0,
        "rango_real": ["2017-05-01", "2026-01-31"]
      },
      "horarios": {
        "regla": "HORA_TERMINO > HORA_INICIO",
        "total_evaluados": 451251,
        "inconsistentes": 0,
        "porcentaje_error": 0,
        "datos_inconsistentes": {}
      },
      "duplicados": {
               "columnas_clave": ["ID_PCTE", "FECHA", "HORA_INICIO", "RUT_PROFESIONAL", "ESTABLECIMIENTO",
"TIPO_CUPO", "ESTADO_CUPO", "MOTIVO_CUPO", "ESTADO_CITA", "TIPO_ATENCION", "DETALLE_CUPO", "OBSERVACIONES"],
        "total_registros": 451251,
        "registros_duplicados": 139,
        "porcentaje_duplicados": 0.03,
        "datos_duplicados": {},
        "archivo_cuarentena": {
                                                                                                  "ruta":
"C:/Users/Anghello/Documents/GitHub/cupos_por_citas_DSM_TCO-main/LOGS/CUARENTENA/duplicados_EXE_20260208_032738
_e7119bd8_20260208_032742.parquet",
          "nombre": "duplicados_EXE_20260208_032738_e7119bd8_20260208_032742.parquet",
          "registros": 139,
                "columnas_clave": ["ID_PCTE", "FECHA", "HORA_INICIO", "RUT_PROFESIONAL", "ESTABLECIMIENTO",
"TIPO_CUPO", "ESTADO_CUPO", "MOTIVO_CUPO", "ESTADO_CITA", "TIPO_ATENCION", "DETALLE_CUPO", "OBSERVACIONES"]
        }
      },
      "cupos_perdidos": {
        "total_cupos": 451251,
        "cupos_perdidos": 136002,
```

```
        "porcentaje_perdidos": 30.14,
                                                                                "archivo_log":
"C:/Users/Anghello/Documents/GitHub/cupos_por_citas_DSM_TCO-main/LOGS/CUPOS_PERDIDOS/cupos_perdidos_EXE_2026020
8_032738_e7119bd8_20260208_032743.parquet",
        "resumen_top10": [
          {
            "ESTABLECIMIENTO": "Amanecer [CESFAM]",
            "TIPO_ATENCION": "procedimientos",
            "n_perdidos": 15250
          },
          {
            "ESTABLECIMIENTO": "Amanecer [CESFAM]",
            "TIPO_ATENCION": "TELEMEDICINA MORBILIDAD",
            "n_perdidos": 13649
          },
          {
            "ESTABLECIMIENTO": "Amanecer [CESFAM]",
            "TIPO_ATENCION": "Inyectables",
            "n_perdidos": 7768
          },
          {
            "ESTABLECIMIENTO": "CECOF Arquenco",
            "TIPO_ATENCION": "Procedimientos",
            "n_perdidos": 5450
          },
          {
            "ESTABLECIMIENTO": "Amanecer [CESFAM]",
            "TIPO_ATENCION": "Administración",
            "n_perdidos": 4968
          },
          {
            "ESTABLECIMIENTO": "Amanecer [CESFAM]",
            "TIPO_ATENCION": "Visita Domiciliaria a Postrados",
            "n_perdidos": 4807
          },
          {
            "ESTABLECIMIENTO": "Amanecer [CESFAM]",
            "TIPO_ATENCION": "Curación Avanzada",
            "n_perdidos": 4118
          },
          {
            "ESTABLECIMIENTO": "Amanecer [CESFAM]",
            "TIPO_ATENCION": "Curación Simple",
            "n_perdidos": 4116
          },
          {
            "ESTABLECIMIENTO": "Amanecer [CESFAM]",
            "TIPO_ATENCION": "Copia de recetas y/o exámenes",
            "n_perdidos": 3801
          },
          {
```

```
        "ESTABLECIMIENTO": "Amanecer [CESFAM]",
        "TIPO_ATENCION": "Control Ginecológico",
        "n_perdidos": 3396
      }
    ]
  },
  "advertencias": [],
  "errores": [],
  "execution_id": "EXE_20260208_032738_e7119bd8"
},
"seguridad": {
  "pii": {
    "execution_id": "EXE_20260208_032738_e7119bd8",
    "fecha_inventario": "2026-02-08 03:27:44.085356",
    "n_columnas_pii": 5,
    "inventario": {
      "RUT": {
        "columna": "RUT",
        "tipo": "character",
        "total_registros": 451251,
        "registros_con_valor": 168131,
        "registros_vacios": 283120,
        "valores_unicos": 25012,
        "muestra_hash": "e24a0093fb79e53a406065ea2f5d75fce5421af46135fa21a9cf27c9b190cace"
      },
      "NOMBRE": {
        "columna": "NOMBRE",
        "tipo": "character",
        "total_registros": 451251,
        "registros_con_valor": 197508,
        "registros_vacios": 253743,
        "valores_unicos": 29598,
        "muestra_hash": "f90e226b517fc98450b20171e6932410677276ea150ea46a1151a93104d00346"
      },
      "NOMBRE_SOCIAL": {
        "columna": "NOMBRE_SOCIAL",
        "tipo": "character",
        "total_registros": 451251,
        "registros_con_valor": 158764,
        "registros_vacios": 292487,
        "valores_unicos": 3379,
        "muestra_hash": "8a5447e773ad363b1e27f2a6d309cd6e061dc3fbcd71409aeec110a0ded2bad3"
      },
      "TELEFONOS": {
        "columna": "TELEFONOS",
        "tipo": "character",
        "total_registros": 451251,
        "registros_con_valor": 197253,
        "registros_vacios": 253998,
        "valores_unicos": 22567,
        "muestra_hash": "8991e7353091d09cdb51bad56b9b6eaaee07a1a290002eca5d2b6f19b07d12bf"
```

```
      },
      "RUT_PROFESIONAL": {
        "columna": "RUT_PROFESIONAL",
        "tipo": "character",
        "total_registros": 451251,
        "registros_con_valor": 451251,
        "registros_vacios": 0,
        "valores_unicos": 326,
        "muestra_hash": "0a43a67af41cf9b898695d2ed91ee71cebe8e34c0a9a2d6323211d7812ac2d06"
      }
    }
  },
  "cumplimiento": {
    "execution_id": "EXE_20260208_032738_e7119bd8",
    "fecha_verificacion": "2026-02-08 03:27:44.098709",
    "controles": {
      "logs_auditoria": {
        "control": "Logs de auditoría habilitados",
        "referencia": "ISO 27001 - A.12.4.1",
        "estado": false,
        "observacion": "CREAR directorio de logs"
      },
      "deteccion_pii": {
        "control": "Identificación de datos personales",
        "referencia": "Ley 21.719 Art. 3",
        "estado": true,
        "n_columnas_pii": 5,
        "columnas": ["RUT", "NOMBRE", "NOMBRE_SOCIAL", "TELEFONOS", "RUT_PROFESIONAL"]
      },
      "integridad": {
        "control": "Verificación de integridad (SHA-256)",
        "referencia": "ISO 27001 - A.12.2.1",
        "estado": true,
        "observacion": "SHA-256 implementado en capas Bronze y Silver"
      },
      "retencion": {
        "control": "Política de retención (6 años)",
        "referencia": "Ley 21.663 - Ciberseguridad",
        "estado": true,
        "años": 6
      }
    },
    "resumen": {
      "controles_evaluados": 4,
      "controles_ok": 3,
      "porcentaje_cumplimiento": 75
    }
  },
  "execution_id": "EXE_20260208_032738_e7119bd8",
  "fecha": "2026-02-08 03:27:44.130007"
}
```

```
  },
  "cumplimiento": {
    "iso_27001": true,
    "ley_21663": true,
    "ley_21719": true
  }
}
```

Archivo: SRC\01_extraccion_bronze.R

```
# ======================================================
# CAPA BRONZE: Extracción y Carga (ELT - Extract & Load)
# ======================================================
# Autor: Anghello Vega Flores
# Cargo: Gestor de Datos
# Institución: Departamento de Salud Municipal de Temuco
# ======================================================
# Filosofía:
#   - Velocidad y fidelidad. No se corrige nada.
#   - Todo se lee como TEXTO para evitar fallos de conversión.
#   - Se guarda inmediatamente en Parquet crudo.
#   - Si hay un dato mal escrito, se preserva para auditoría.
# ======================================================

library(readxl)
library(janitor)
library(arrow)
library(dplyr)
library(digest)

# ======================================================
# FUNCIÓN: Extraer Excel a Bronze (Sin transformaciones)
# ======================================================
extraer_a_bronze <- function(archivo_excel, ruta_bronze) {

  tryCatch({
    cat("  [BRONZE] Extrayendo: ", basename(archivo_excel), "\n", sep = "")

    # Crear directorio Bronze si no existe
    if (!dir.exists(ruta_bronze)) {
      dir.create(ruta_bronze, recursive = TRUE, showWarnings = FALSE)
    }

    # --- LECTURA: Todo como texto (velocidad + fidelidad) ---
    datos_crudos <- readxl::read_excel(
      archivo_excel,
      skip = 8,
      col_types = "text"  # CRÍTICO: Todo a texto
    ) %>%
      janitor::clean_names(case = "screaming_snake")
```

```
    if (nrow(datos_crudos) == 0) {
      cat("    [ALERTA] Archivo vacio. Saltando.\n")
      return(list(
        exito = TRUE,
        registros = 0,
        archivo = basename(archivo_excel),
        ruta_parquet = NA_character_,
        hash_archivo = NA_character_,
        hash_contenido = NA_character_
      ))
    }

    # --- HASHING: Calcular hashes para trazabilidad ---
    hash_archivo <- digest::digest(archivo_excel, algo = "md5", file = TRUE)
    hash_contenido <- digest::digest(datos_crudos, algo = "md5")

    # --- METADATA: Agregar trazabilidad ---
    datos_crudos <- datos_crudos %>%
      mutate(
        ETL_ARCHIVO_ORIGEN = basename(archivo_excel),
        ETL_RUTA_ORIGEN = archivo_excel,
        ETL_FECHA_EXTRACCION = format(Sys.time(), "%Y-%m-%d %H:%M:%S"),
        ETL_HASH_ARCHIVO = hash_archivo,
        ETL_HASH_CONTENIDO = hash_contenido,
        ETL_REGISTROS_ORIGEN = nrow(datos_crudos)
      )

    # --- GUARDADO: Parquet crudo (sin particiones, rápido) ---
    nombre_parquet <- paste0(
      tools::file_path_sans_ext(basename(archivo_excel)),
      "_bronze.parquet"
    )
    ruta_parquet <- file.path(ruta_bronze, nombre_parquet)

    arrow::write_parquet(datos_crudos, ruta_parquet)

     cat("    [OK] Bronze guardado: ", nrow(datos_crudos), " registros | Hash: ", substr(hash_archivo, 1, 8),
"...\n", sep = "")

    return(list(
      exito = TRUE,
      registros = nrow(datos_crudos),
      archivo = basename(archivo_excel),
      ruta_parquet = ruta_parquet,
      hash_archivo = hash_archivo,
      hash_contenido = hash_contenido
    ))

  }, error = function(e) {
    cat("    [ERROR] ", conditionMessage(e), "\n", sep = "")
```

```
    return(list(
      exito = FALSE,
      registros = 0,
      archivo = basename(archivo_excel),
      ruta_parquet = NA_character_,
      error = conditionMessage(e)
    ))
  })
}

# ======================================================
# FUNCIÓN: Procesar todos los archivos a Bronze
# ======================================================
procesar_lote_bronze <- function(archivos_excel, ruta_bronze) {

  cat("\n[BRONZE] Iniciando extraccion de ", length(archivos_excel), " archivos...\n", sep = "")

  resultados <- lapply(archivos_excel, function(archivo) {
    extraer_a_bronze(archivo, ruta_bronze)
  })

  # Resumen
  exitosos <- sum(sapply(resultados, function(x) x$exito))
  total_registros <- sum(sapply(resultados, function(x) x$registros))

  cat("\n[BRONZE] Resumen:\n")
  cat("  - Archivos procesados: ", exitosos, "/", length(archivos_excel), "\n", sep = "")
  cat("  - Registros totales: ", total_registros, "\n", sep = "")

  return(resultados)
}

# ======================================================
# FUNCIÓN: Verificar integridad de Bronze
# ======================================================
verificar_bronze <- function(ruta_bronze) {

  archivos_parquet <- list.files(ruta_bronze, pattern = "\\.parquet$", full.names = TRUE)

  if (length(archivos_parquet) == 0) {
    cat("[BRONZE] No hay archivos Parquet en Bronze.\n")
    return(FALSE)
  }

  cat("\n[BRONZE] Verificando ", length(archivos_parquet), " archivos Parquet...\n", sep = "")

  total_registros <- 0
  for (archivo in archivos_parquet) {
    tabla <- arrow::read_parquet(archivo)
    total_registros <- total_registros + nrow(tabla)
  }
```

```
  cat("  - Total registros en Bronze: ", total_registros, "\n", sep = "")
  return(TRUE)
}
```

Archivo: SRC\02_transformacion_silver.R

```
# ======================================================
# CAPA SILVER: Transformación (ELT - Transform)
# ======================================================
# Autor: Anghello Vega Flores
# Cargo: Gestor de Datos
# Institución: Departamento de Salud Municipal de Temuco
# ======================================================
# Filosofía:
#   - Lee desde Bronze (Parquet crudo, rápido).
#   - Aplica transformaciones masivas con DuckDB/SQL.
#   - Crea columnas calculadas: ID_PCTE, CENTRO_SALUD, etc.
#   - Valida y marca registros problemáticos.
#   - Guarda particionado en Silver.
# ======================================================

library(duckdb)
library(dplyr)
library(arrow)
library(glue)

# ======================================================
# FUNCIÓN: Conectar a DuckDB y registrar Bronze
# ======================================================
crear_conexion_duckdb <- function(ruta_bronze) {

  # Crear conexión en memoria (eficiente para transformaciones)
  con <- dbConnect(duckdb::duckdb(), dbdir = ":memory:")

  # Registrar todos los Parquet de Bronze como una vista
  archivos_bronze <- list.files(ruta_bronze, pattern = "\\.parquet$", full.names = TRUE)

  if (length(archivos_bronze) == 0) {
    stop("[SILVER] No hay archivos Parquet en Bronze para procesar.")
  }

  # Crear vista unificada de todos los archivos Bronze
  rutas_glob <- file.path(ruta_bronze, "*.parquet")
  query_registro <- glue::glue("
    CREATE VIEW bronze_raw AS
    SELECT * FROM read_parquet('{rutas_glob}', union_by_name=true)
  ")

  dbExecute(con, query_registro)
```

```
  cat("[SILVER] Conectado a DuckDB. Bronze registrado.\n")
  return(con)
}

# ======================================================
# SQL: Mapeo de ESTABLECIMIENTO ? CENTRO_SALUD
# ======================================================
SQL_MAPEO_CENTRO_SALUD <- "
  CASE
    -- Amanecer
    WHEN ESTABLECIMIENTO IN ('CESFAM Amanecer', 'Amanecer [CESFAM]') THEN 'Amanecer'

    -- Arquenco
    WHEN ESTABLECIMIENTO = 'CECOF Arquenco' THEN 'Arquenco'

    -- CCR Temuco
     WHEN ESTABLECIMIENTO IN ('CCR Temuco', 'Centro de Tratamiento Temuco', 'Centro Tratamiento Temuco') THEN
'CCR Temuco'

    -- El Carmen
    WHEN ESTABLECIMIENTO = 'CESFAM El Carmen' THEN 'El Carmen'

    -- El Salar
    WHEN ESTABLECIMIENTO = 'CECOF El Salar' THEN 'El Salar'

    -- Las Quilas
    WHEN ESTABLECIMIENTO = 'CECOF Las Quilas' THEN 'Las Quilas'

    -- Labranza
    WHEN ESTABLECIMIENTO IN ('CESFAM Labranza', 'Centro de Salud Labranza') THEN 'Labranza'

    -- Microcentro Amanecer
    WHEN ESTABLECIMIENTO = 'Anexo Nuevo Amanecer' THEN 'Microcentro Amanecer'

    -- Sergio Valech
    WHEN ESTABLECIMIENTO IN ('CGR Sergio Valech', 'Mun. de Temuco - CGR Sergio Valech') THEN 'Sergio Valech'

    -- Pueblo Nuevo
    WHEN ESTABLECIMIENTO IN ('CESFAM Pueblo Nuevo', 'Pueblo Nuevo [CESFAM]') THEN 'Pueblo Nuevo'

    -- Pedro de Valdivia
    WHEN ESTABLECIMIENTO IN ('CESFAM Pedro de Valdivia', 'Pedro de Valdivia [CESFAM]') THEN 'Pedro De Valdivia'

    -- Santa Rosa
    WHEN ESTABLECIMIENTO IN ('CGU Santa Rosa', 'Santa Rosa [CGU]') THEN 'Santa Rosa'

    -- Villa Alegre
    WHEN ESTABLECIMIENTO IN ('CGU Villa Alegre', 'Villa Alegre [CGU]') THEN 'Villa Alegre'

    -- Alo Red
```

```
    WHEN ESTABLECIMIENTO = 'ESTRATEGIA ALÓRED' THEN 'Alo Red'

    -- Centro Comunitario de Rehabilitación Temuco
       WHEN ESTABLECIMIENTO ILIKE '%Centro Comunitario de Rehabilita%Temuco%' THEN 'Centro Comunitario de Rehabilitacion Temuco'

    ELSE TRIM(ESTABLECIMIENTO)
  END
"

# ======================================================
# SQL: Clasificación de TIPO_CENTRO
# ======================================================
SQL_TIPO_CENTRO <- "
  CASE
    WHEN CENTRO_SALUD = 'CCR Temuco' THEN 'CCR'
    WHEN CENTRO_SALUD IN ('Arquenco', 'Las Quilas', 'El Salar') THEN 'CECOSF'
    WHEN CENTRO_SALUD = 'Microcentro Amanecer' THEN 'ANEXO'
    WHEN UPPER(CENTRO_SALUD) LIKE '%CENTRO COMUNITARIO DE REHABILITA%TEMUCO%' THEN 'ANEXO'
    WHEN CENTRO_SALUD = 'Alo Red' THEN 'ESTRATEGIA'
    ELSE 'CESFAM'
  END
"

# ======================================================
# SQL: Grupo Etario
# ======================================================
SQL_GRUPO_ETARIO <- "
  CASE
    WHEN EDAD_ANO_NUM < 1 THEN 'Menores de 1 año'
    WHEN EDAD_ANO_NUM >= 1 AND EDAD_ANO_NUM < 5 THEN '1-4 años'
    WHEN EDAD_ANO_NUM >= 5 AND EDAD_ANO_NUM < 10 THEN '5-9 años'
    WHEN EDAD_ANO_NUM >= 10 AND EDAD_ANO_NUM < 15 THEN '10-14 años'
    WHEN EDAD_ANO_NUM >= 15 AND EDAD_ANO_NUM < 20 THEN '15-19 años'
    WHEN EDAD_ANO_NUM >= 20 AND EDAD_ANO_NUM < 25 THEN '20-24 años'
    WHEN EDAD_ANO_NUM >= 25 AND EDAD_ANO_NUM < 30 THEN '25-29 años'
    WHEN EDAD_ANO_NUM >= 30 AND EDAD_ANO_NUM < 35 THEN '30-34 años'
    WHEN EDAD_ANO_NUM >= 35 AND EDAD_ANO_NUM < 40 THEN '35-39 años'
    WHEN EDAD_ANO_NUM >= 40 AND EDAD_ANO_NUM < 45 THEN '40-44 años'
    WHEN EDAD_ANO_NUM >= 45 AND EDAD_ANO_NUM < 50 THEN '45-49 años'
    WHEN EDAD_ANO_NUM >= 50 AND EDAD_ANO_NUM < 55 THEN '50-54 años'
    WHEN EDAD_ANO_NUM >= 55 AND EDAD_ANO_NUM < 60 THEN '55-59 años'
    WHEN EDAD_ANO_NUM >= 60 AND EDAD_ANO_NUM < 65 THEN '60-64 años'
    WHEN EDAD_ANO_NUM >= 65 AND EDAD_ANO_NUM < 70 THEN '65-69 años'
    WHEN EDAD_ANO_NUM >= 70 AND EDAD_ANO_NUM < 75 THEN '70-74 años'
    WHEN EDAD_ANO_NUM >= 75 AND EDAD_ANO_NUM < 80 THEN '75-79 años'
    WHEN EDAD_ANO_NUM >= 80 AND EDAD_ANO_NUM < 85 THEN '80-84 años'
    WHEN EDAD_ANO_NUM >= 85 THEN '85 años y más'
    ELSE NULL
  END
"
```

```
# ====================================================
# SQL: Horario Extendido
# ====================================================
SQL_HORARIO_EXTENDIDO <- "
  CASE
    WHEN ISO_DOW >= 6 THEN 'Si'                              -- Sábado y Domingo
    WHEN ISO_DOW = 5 AND HORA_INICIO_NUM >= 16 THEN 'Si'     -- Viernes >= 16:00
    WHEN ISO_DOW <= 4 AND HORA_INICIO_NUM >= 17 THEN 'Si'    -- Lunes-Jueves >= 17:00
    ELSE 'No'
  END
"

# ====================================================
# FUNCIÓN PRINCIPAL: Transformar Bronze ? Silver
# ====================================================
transformar_bronze_a_silver <- function(ruta_bronze, ruta_silver, columnas_maestras) {

  cat("\n[SILVER] Iniciando transformacion...\n")

  # Crear conexión DuckDB
  con <- crear_conexion_duckdb(ruta_bronze)
  on.exit(dbDisconnect(con, shutdown = TRUE), add = TRUE)

  # Crear directorio Silver si no existe
  if (!dir.exists(ruta_silver)) {
    dir.create(ruta_silver, recursive = TRUE, showWarnings = FALSE)
  }

  # --- QUERY DE TRANSFORMACIÓN MASIVA ---
  query_transformacion <- glue::glue("
    WITH
    -- Paso 1: Normalizar columnas (corregir typos, asegurar existencia)
    datos_normalizados AS (
      SELECT
        *,
        -- Corregir RENDIMIMENTO ? RENDIMIENTO (el typo viene del Excel original)
        COALESCE(TRY_CAST(RENDIMIMENTO AS VARCHAR), '') AS RENDIMIENTO_CORR,
        -- Asegurar columnas de identificación
          COALESCE(TRY_CAST(NUMERO_TIPO_IDENTIFICACION AS VARCHAR), COALESCE(TRY_CAST(NUMERO_IDENTIFICACION AS
VARCHAR), '')) AS NUMERO_TIPO_ID_CLEAN,
        COALESCE(RUT, '') AS RUT_CLEAN
      FROM bronze_raw
    ),

    -- Paso 2: Conversiones de tipos
    datos_tipados AS (
      SELECT
        *,
        -- Fechas: Intentar desde número Excel, luego desde texto
        CASE
```

```
  WHEN TRY_CAST(FECHA AS DOUBLE) IS NOT NULL
    THEN DATE '1899-12-30' + INTERVAL (CAST(FECHA AS INTEGER)) DAY
  ELSE TRY_CAST(STRPTIME(FECHA, '%d-%m-%Y') AS DATE)
END AS FECHA_CONV,

CASE
  WHEN TRY_CAST(FECHA_DE_NACIMIENTO AS DOUBLE) IS NOT NULL
    THEN DATE '1899-12-30' + INTERVAL (CAST(FECHA_DE_NACIMIENTO AS INTEGER)) DAY
  ELSE TRY_CAST(STRPTIME(FECHA_DE_NACIMIENTO, '%d-%m-%Y') AS DATE)
END AS FECHA_NACIMIENTO_CONV,

CASE
  WHEN TRY_CAST(FECHA_BLOQUEO AS DOUBLE) IS NOT NULL
    THEN DATE '1899-12-30' + INTERVAL (CAST(FECHA_BLOQUEO AS INTEGER)) DAY
  ELSE TRY_CAST(STRPTIME(FECHA_BLOQUEO, '%d-%m-%Y') AS DATE)
END AS FECHA_BLOQUEO_CONV,

-- Horas: Normalizar a HH:MM (desde fracción de día o texto)
CASE
  WHEN TRY_CAST(HORA_INICIO AS DOUBLE) IS NOT NULL THEN
    LPAD(CAST(FLOOR(CAST(HORA_INICIO AS DOUBLE) * 24) AS VARCHAR), 2, '0') || ':' ||
    LPAD(CAST(FLOOR(MOD(CAST(HORA_INICIO AS DOUBLE) * 1440, 60)) AS VARCHAR), 2, '0')
  WHEN HORA_INICIO LIKE '%:%' THEN
    LPAD(SPLIT_PART(HORA_INICIO, ':', 1), 2, '0') || ':' ||
    LPAD(SPLIT_PART(HORA_INICIO, ':', 2), 2, '0')
  ELSE NULL
END AS HORA_INICIO_NORM,

CASE
  WHEN TRY_CAST(HORA_TERMINO AS DOUBLE) IS NOT NULL THEN
    LPAD(CAST(FLOOR(CAST(HORA_TERMINO AS DOUBLE) * 24) AS VARCHAR), 2, '0') || ':' ||
    LPAD(CAST(FLOOR(MOD(CAST(HORA_TERMINO AS DOUBLE) * 1440, 60)) AS VARCHAR), 2, '0')
  WHEN HORA_TERMINO LIKE '%:%' THEN
    LPAD(SPLIT_PART(HORA_TERMINO, ':', 1), 2, '0') || ':' ||
    LPAD(SPLIT_PART(HORA_TERMINO, ':', 2), 2, '0')
  ELSE NULL
END AS HORA_TERMINO_NORM,

CASE
  WHEN TRY_CAST(HORA_BLOQUEO AS DOUBLE) IS NOT NULL THEN
    LPAD(CAST(FLOOR(CAST(HORA_BLOQUEO AS DOUBLE) * 24) AS VARCHAR), 2, '0') || ':' ||
    LPAD(CAST(FLOOR(MOD(CAST(HORA_BLOQUEO AS DOUBLE) * 1440, 60)) AS VARCHAR), 2, '0')
  WHEN HORA_BLOQUEO LIKE '%:%' THEN
    LPAD(SPLIT_PART(HORA_BLOQUEO, ':', 1), 2, '0') || ':' ||
    LPAD(SPLIT_PART(HORA_BLOQUEO, ':', 2), 2, '0')
  ELSE NULL
END AS HORA_BLOQUEO_NORM,

-- Enteros
TRY_CAST(EDAD_ANO AS INTEGER) AS EDAD_ANO_NUM,
TRY_CAST(EDAD_MES AS INTEGER) AS EDAD_MES_NUM,
```

```
    TRY_CAST(EDAD_DIA AS INTEGER) AS EDAD_DIA_NUM,
    TRY_CAST(CUPOS_UTILIZADOS AS INTEGER) AS CUPOS_UTILIZADOS_NUM,

    -- Rendimiento: Minutos a HH:MM:SS
    CASE
      WHEN TRY_CAST(RENDIMIENTO_CORR AS DOUBLE) IS NOT NULL THEN
        LPAD(CAST(FLOOR(CAST(RENDIMIENTO_CORR AS DOUBLE) / 60) AS VARCHAR), 2, '0') || ':' ||
        LPAD(CAST(FLOOR(MOD(CAST(RENDIMIENTO_CORR AS DOUBLE), 60)) AS VARCHAR), 2, '0') || ':' ||
        LPAD(CAST(ROUND(MOD(CAST(RENDIMIENTO_CORR AS DOUBLE), 1) * 60) AS VARCHAR), 2, '0')
      ELSE NULL
    END AS RENDIMIENTO_NORM

  FROM datos_normalizados
),

-- Paso 3: Crear columnas calculadas
datos_enriquecidos AS (
  SELECT
    *,
    -- ID_PCTE: Preferir NUMERO_TIPO_IDENTIFICACION, sino RUT
    REGEXP_REPLACE(
      UPPER(
        CASE
          WHEN NUMERO_TIPO_ID_CLEAN != '' THEN NUMERO_TIPO_ID_CLEAN
          WHEN RUT_CLEAN != '' THEN RUT_CLEAN
          ELSE NULL
        END
      ),
      '[^A-Z0-9K]', '', 'g'
    ) AS ID_PCTE,

    -- CENTRO_SALUD (mapeo)
    {SQL_MAPEO_CENTRO_SALUD} AS CENTRO_SALUD,

    -- ISO_DOW: Día de la semana (1=Lunes, 7=Domingo)
    DAYOFWEEK(FECHA_CONV) AS ISO_DOW,

    -- Hora numérica para clasificación
    TRY_CAST(SPLIT_PART(HORA_INICIO_NORM, ':', 1) AS INTEGER) AS HORA_INICIO_NUM

  FROM datos_tipados
),

-- Paso 4: Agregar TIPO_CENTRO, GRUPO_ETARIO, HORARIO_EXTENDIDO
datos_finales AS (
  SELECT
    -- Columnas originales (limpias) - usar las normalizadas
    NUMERO_TIPO_ID_CLEAN AS NUMERO_TIPO_IDENTIFICACION,
    RUT,
    NOMBRE,
    NOMBRE_SOCIAL,
```

```
  SS,
  ESTABLECIMIENTO,
  FECHA_CONV AS FECHA,
  TIPO_ATENCION,
  ESPECIALIDAD,
  INSTRUMENTO,
  TIPO_CUPO,
  HORA_INICIO_NORM AS HORA_INICIO,
  HORA_TERMINO_NORM AS HORA_TERMINO,
  ESTADO_CUPO,
  HABILITADO,
  TIPO_DE_AGENDAMIENTO,
  TELECONSULTA,
  MOTIVO_CUPO,
  DETALLE_CUPO,
  OBSERVACIONES,
  EDAD_ANO_NUM AS EDAD_ANO,
  EDAD_MES_NUM AS EDAD_MES,
  EDAD_DIA_NUM AS EDAD_DIA,
  FECHA_NACIMIENTO_CONV AS FECHA_DE_NACIMIENTO,
  TELEFONOS,
  SECTOR,
  RUT_PROFESIONAL,
  PROFESIONAL,
  FUNCIONARIO_CITADOR,
  MOTIVO_BLOQUEO,
  FUNCIONARIO_REALIZA_BLOQUEO,
  FECHA_BLOQUEO_CONV AS FECHA_BLOQUEO,
  HORA_BLOQUEO_NORM AS HORA_BLOQUEO,
  ESTADO_CITA,
  RENDIMIENTO_NORM AS RENDIMIENTO,
  CUPOS_UTILIZADOS_NUM AS CUPOS_UTILIZADOS,

  -- Columnas NUEVAS (calculadas)
  ID_PCTE,
  CENTRO_SALUD,
  {SQL_TIPO_CENTRO} AS TIPO_CENTRO,
  ISO_DOW,
  {SQL_GRUPO_ETARIO} AS GRUPO_ETARIO,
  {SQL_HORARIO_EXTENDIDO} AS HORARIO_EXTENDIDO,

  -- Columnas de TRAZABILIDAD (ETL Metadata)
  ETL_ARCHIVO_ORIGEN,
  ETL_FECHA_EXTRACCION,
  ETL_HASH_ARCHIVO,

  -- Columnas de particionamiento
  YEAR(FECHA_CONV) AS año,
  LPAD(CAST(MONTH(FECHA_CONV) AS VARCHAR), 2, '0') AS mes

FROM datos_enriquecidos
```

```
      WHERE FECHA_CONV IS NOT NULL  -- Filtrar registros sin fecha válida
    )

    SELECT * FROM datos_finales
  ")

  # Ejecutar transformación
  cat("[SILVER] Ejecutando transformacion SQL...\n")
  resultado <- dbGetQuery(con, query_transformacion)

  cat("[SILVER] Transformados: ", nrow(resultado), " registros\n", sep = "")

  # Convertir a Arrow Table y guardar particionado
  tabla_arrow <- arrow::as_arrow_table(resultado)

  cat("[SILVER] Guardando particionado (Hive: año/mes)...\n")

  arrow::write_dataset(
    dataset = tabla_arrow,
    path = ruta_silver,
    format = "parquet",
    partitioning = c("año", "mes"),
    existing_data_behavior = "delete_matching"
  )

  # Resumen de particiones
  particiones <- resultado %>%
    select(año, mes) %>%
    distinct() %>%
    arrange(año, mes)

  cat("[SILVER] Particiones creadas: ", nrow(particiones), "\n", sep = "")

  return(list(
    exito = TRUE,
    registros = nrow(resultado),
    particiones = nrow(particiones),
    ruta_salida = ruta_silver
  ))
}

# ======================================================
# FUNCIÓN: Verificar dataset Silver
# ======================================================
verificar_silver <- function(ruta_silver) {

  cat("\n[SILVER] Verificando dataset...\n")

  tryCatch({
    tabla <- arrow::open_dataset(ruta_silver, format = "parquet")
```

```
    registros_totales <- nrow(tabla)
    columnas <- names(tabla)

    cat("  - Registros totales: ", registros_totales, "\n", sep = "")
    cat("  - Columnas: ", length(columnas), "\n", sep = "")

    # Ver particiones
    particiones <- tabla %>%
      select(año, mes) %>%
      distinct() %>%
      collect() %>%
      arrange(año, mes)

    cat("  - Particiones: ", nrow(particiones), "\n", sep = "")
    for (i in 1:min(nrow(particiones), 10)) {
      cat("      año=", particiones$año[i], "/mes=", particiones$mes[i], "\n", sep = "")
    }
    if (nrow(particiones) > 10) {
      cat("      ... y ", nrow(particiones) - 10, " más\n", sep = "")
    }

    return(TRUE)

  }, error = function(e) {
    cat("  [ERROR] ", conditionMessage(e), "\n", sep = "")
    return(FALSE)
  })
}
```

Archivo: SRC\03_guardar_particionado.R

```
# ========================================================
# FUNCIONES AUXILIARES: Verificación y Utilidades
# ========================================================
# Autor: Anghello Vega Flores
# Cargo: Gestor de Datos
# Institución: Departamento de Salud Municipal de Temuco
# ========================================================
# Nota: En arquitectura ELT, el guardado particionado se
# realiza en 02_transformacion_silver.R con DuckDB.
# Este archivo contiene funciones de utilidad.
# ========================================================

library(arrow)
library(dplyr)

# ========================================================
# FUNCIÓN: Verificar integridad de dataset particionado
# ========================================================
verificar_dataset_particionado <- function(ruta_dataset) {
```

```
  cat("\n[VERIFICACION] Dataset particionado...\n")

  tryCatch({
    # Leer como tabla Arrow (sin cargar en memoria)
    tabla <- open_dataset(ruta_dataset, format = "parquet")

    registros_totales <- nrow(tabla)
    columnas <- names(tabla)

    cat("  Registros totales: ", format(registros_totales, big.mark = ","), "\n", sep = "")
    cat("  Columnas: ", length(columnas), "\n", sep = "")

    # Ver estructura de particiones
    particiones <- tabla %>%
      select(año, mes) %>%
      distinct() %>%
      collect() %>%
      arrange(año, mes)

    cat("  Particiones encontradas: ", nrow(particiones), "\n", sep = "")
    for (i in 1:nrow(particiones)) {
      cat("    - año=", particiones$año[i], "/mes=", particiones$mes[i], "\n", sep = "")
    }

    return(TRUE)

  }, error = function(e) {
    cat("  [ERROR] ", conditionMessage(e), "\n", sep = "")
    return(FALSE)
  })
}

# ========================================================
# FUNCIÓN: Obtener estadísticas del dataset
# ========================================================
obtener_estadisticas_dataset <- function(ruta_dataset) {

  cat("\n[ESTADISTICAS] Analizando dataset...\n")

  tryCatch({
    tabla <- open_dataset(ruta_dataset, format = "parquet")

    # Estadísticas por partición
    stats <- tabla %>%
      group_by(año, mes) %>%
      summarise(
        registros = n(),
        .groups = "drop"
      ) %>%
      collect() %>%
```

```
      arrange(año, mes)

    cat("  Registros por período:\n")
    for (i in 1:nrow(stats)) {
      cat("    ", stats$año[i], "-", stats$mes[i], ": ",
          format(stats$registros[i], big.mark = ","), " registros\n", sep = "")
    }

    return(stats)

  }, error = function(e) {
    cat("  [ERROR] ", conditionMessage(e), "\n", sep = "")
    return(NULL)
  })
}

# ====================================================
# FUNCIÓN: Leer dataset completo (Arrow, sin cargar en RAM)
# ====================================================
abrir_dataset <- function(ruta_dataset) {
  open_dataset(ruta_dataset, format = "parquet")
}

# ====================================================
# FUNCIÓN: Consultar dataset con filtros (eficiente)
# ====================================================
consultar_dataset <- function(ruta_dataset, filtro_año = NULL, filtro_mes = NULL) {

  tabla <- open_dataset(ruta_dataset, format = "parquet")

  if (!is.null(filtro_año)) {
    tabla <- tabla %>% filter(año == filtro_año)
  }

  if (!is.null(filtro_mes)) {
    tabla <- tabla %>% filter(mes == filtro_mes)
  }

  return(tabla)
}
```

Archivo: SRC\config.R

```
# ====================================================
# SCRIPT DE CONFIGURACIÓN GLOBAL
# ====================================================
# Autor: Anghello Vega Flores
# Cargo: Gestor de Datos
# Institución: Departamento de Salud Municipal de Temuco
```

```
# ======================================================
# Arquitectura: ELT con Medallón (Bronze ? Silver)
# ======================================================

library(here)

# --- 1. Rutas por Capas (Arquitectura Medallón) ---
# Entrada: Archivos Excel crudos
RUTA_DATOS_CRUDOS <- here::here("DATOS", "ENTRADA")

# Bronze: Parquet crudo (todo como texto, sin transformaciones)
RUTA_BRONZE <- here::here("DATOS", "BRONZE")

# Silver: Parquet limpio y transformado (particionado)
RUTA_SILVER <- here::here("DATOS", "SALIDA")

# Alias para compatibilidad
RUTA_DATOS_PROCESADOS <- RUTA_SILVER

# --- 2. CONTRATO MAESTRO (Según especificación DOC_TRANSFOR_ANGHELLO_VEGA.txt) ---
COLUMNAS_MAESTRAS <- c(
  # --- Columnas Originales del Excel (sin renombrar) ---
  "NUMERO_TIPO_IDENTIFICACION",
  "RUT",
  "NOMBRE",
  "NOMBRE_SOCIAL",
  "SS",
  "ESTABLECIMIENTO",
  "FECHA",
  "TIPO_ATENCION",
  "ESPECIALIDAD",
  "INSTRUMENTO",
  "TIPO_CUPO",
  "HORA_INICIO",
  "HORA_TERMINO",
  "ESTADO_CUPO",
  "HABILITADO",
  "TIPO_DE_AGENDAMIENTO",
  "TELECONSULTA",
  "MOTIVO_CUPO",
  "DETALLE_CUPO",
  "OBSERVACIONES",
  "EDAD_ANO",
  "EDAD_MES",
  "EDAD_DIA",
  "FECHA_DE_NACIMIENTO",
  "TELEFONOS",
  "SECTOR",
  "RUT_PROFESIONAL",
  "PROFESIONAL",
  "FUNCIONARIO_CITADOR",
```

```
  "MOTIVO_BLOQUEO",
  "FUNCIONARIO_REALIZA_BLOQUEO",
  "FECHA_BLOQUEO",
  "HORA_BLOQUEO",
  "ESTADO_CITA",
  "RENDIMIENTO",
  "CUPOS_UTILIZADOS",

  # --- Columnas Creadas (nuevas, según especificación) ---
  "ID_PCTE",
  "CENTRO_SALUD",
  "TIPO_CENTRO",
  "ISO_DOW",
  "GRUPO_ETARIO",
  "HORARIO_EXTENDIDO"
)

# --- 3. Nombre del Archivo de Salida (sin cambios) ---
NOMBRE_ARCHIVO_FINAL <- "cupos_por_citas_consolidados_actualizados.parquet"
```

Archivo: _targets\.gitignore

```
# CAUTION: do not edit this file by hand!
# _targets/objects/ may have large data files,
# and _targets/meta/process may have sensitive information.
# It is good pratice to either commit nothing from _targets/,
# or if your data is not too sensitive,
# commit only _targets/meta/meta.
*
!.gitignore
!meta
meta/*
!meta/meta
```